Mendelova univerzita v Brně
Provozně ekonomická fakulta

Systém pro kontrolu uživatelské aktivity a jeho integrace do systému SMART fakulty

Diplomová práce

Vedoucí práce:
Ing. Tomáš Koubek

Bc. Jan Král

Brno 2017

Rád bych na tomto místě poděkoval Ing. Tomáši Koubkovi za odborné vedení této práce, za časté a velmi podnětné konzultace a za ochotu pružně řešit vzniklé problémy, a to třeba i o víkendu. Dále mé poděkování patří Ing. Janu Kolomazníkovi, PhD. za podporu při vývoji SmartPEF. Na závěr bych rád poděkoval těm, bez kterých bych tuto práci možná ani nedopsal - mé rodině, přítelkyni a přátelům, kteří při mně stáli a byli mi neskutečnou oporou.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Systém pro kontrolu uživatelské aktivity a jeho integrace do systému SMART fakulty**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 22. května 2017 .....

Abstract

Král, J. User activity monitoring system and its integration into SMART faculty system. Diploma thesis. Brno, 2017.

The goal of this thesis is integration of a new data source into SmartPEF system. SmartPEF is the application of the concept of smart cities to the Faculty of Business and Economics of the Mendel University in Brno. As a new data source, a user activity monitoring system is being developed and implemented in order to collect statistical data about the traffic and utilization of university computers.

Keywords

Information system, software utilization analysis, API, Spring framework, SmartPEF, Morpheus.

Abstrakt

Král, J. Systém pro kontrolu uživatelské aktivity a jeho integrace do systému SMART fakulty. Diplomová práce. Brno, 2017.

Diplomová práce se zabývá integrací nového datového zdroje do systému SmartPEF. SmartPEF je aplikace koncepce chytrých měst do prostředí Provozně ekonomické fakulty Mendelovy univerzity v Brně. V rámci práce je jako nový datový zdroj navrhnut a implementován systém pro kontrolu uživatelské aktivity, jehož účelem je sbírat statistická data o provozu a využívanosti univerzitních počítačů.

Klíčová slova

Informační systém, analýza využívanosti softwaru, API, Spring framework, SmartPEF, Morpheus.

Obsah

1 Úvod a cíl **11**
1.1 Úvod . . . 11
1.2 Cíl . . . 11

2 Analýza současného stavu **12**
2.1 Systém Morpheus . . . 12
2.1.1 Architektura . . . 13
2.1.2 Databáze . . . 15
2.2 Meeting Room Booking System . . . 16
2.3 Systém SmartPEF . . . 17
2.4 Požadavky . . . 18
2.4.1 Funkční požadavky . . . 18
2.4.2 Nefunkční požadavky . . . 20

3 Metodika řešení **22**

4 Návrh řešení **23**
4.1 Systém Morpheus . . . 23
4.1.1 Architektura . . . 23
4.1.2 Modul MRBS . . . 23
4.1.3 Modul Refresh . . . 25
4.1.4 Návrh databáze . . . 26
4.1.5 Optimalizace frekvence obnovování informací . . . 27
4.1.6 Návrh API . . . 29
4.2 Systém SmartPEF - modul Morpheus . . . 31
4.2.1 Architektura . . . 33
4.2.2 Návrh databáze . . . 35
4.2.3 Návrh API . . . 40
4.3 Komunikace mezi systémy . . . 41
4.3.1 Analýza objemu dat . . . 41
4.3.2 Způsob výměny dat . . . 42
4.3.3 Agent pro nahrávání dat do SmartPEF . . . 43

5 Implementace **46**
5.1 Identifikace procesů . . . 46
5.1.1 Tasklist . . . 46
5.1.2 Wmic . . . 46
5.1.3 Get-Process . . . 47
5.1.4 Výběr vhodné metody . . . 47
5.2 Identifikace nainstalovaného softwaru . . . 47
5.2.1 Třída Win32Product . . . 47
5.2.2 Využití registrů . . . 48

5.2.3 Výběr vhodné metody 48
5.3 Matching software-proces 48
5.4 SmartPEF REST API 49
5.5 Autentizace a autorizace 51
5.5.1 HTTP Basic 51
5.5.2 OAuth 1.0 a OAuth 2.0 52
5.5.3 Výběr vhodné metody 53
5.5.4 Autorizace na úrovni metod 54
5.6 Tvorba dokumentace pomocí Swagger UI 54

6 Zhodnocení a závěr **56**

7 Literatura **57**

Přílohy **59**

A Elektronická příloha **60**

1 Úvod a cíl

1.1 Úvod

Na Provozně ekonomické fakultě Mendelovy Univerzity vzniká nový projekt SmartPEF, který se zabývá aplikací konceptu chytrých měst do prostředí fakulty. Koncepce chytrých měst si klade za cíl zvýšení kvality života jejich obyvatel, mimo jiné pomocí zavedení informačních a komunikačních technologií. SmartPEF staví na třech pilířích. Těmi jsou serverová infrastruktura, senzorická síť a mobilní zařízení pro zprostředkování informací uživatelům. Díky těmto třem pilířům bude možné studentům a zaměstnancům poskytnout informace o fakultě a jejím provozu. Příkladem může být aplikace, která pomůže studentovi zjistit, na kterých počítačích je nainstalovaný jím hledaný software, aby mohl dodělat semestrální projekt. Dalším případem využití může být analýza využívanosti jednotlivých programů, k nimž univerzita zakoupila licence.

Tento koncept nemůže fungovat bez relevantních datových zdrojů. Data se sbírají jak na fyzické úrovni (např. přes senzorickou síť – teploměry, kamery), tak na softwarové úrovni. Příkladem může být univerzitní informační systém UIS nebo systém Morpheus, který bude využit jako datový zdroj právě pro účely této práce.

Morpheus je systém vyvíjený Ústavem informatiky, který umožňuje hromadnou správu univerzitních počítačů. Systém umožňuje správcům vzdálené zapínání a vypínání stanic, spouštění příkazů nebo lze vzdáleně přihlásit uživatele, například pro účely správy stanice. Systém také poskytuje základní informace o aktuálním stavu stanic, včetně aktuálně nabootovaného operačního systému, přihlášeného uživatele nebo IP adresy. Právě oblast zobrazování základních informací poskytuje dostatek prostoru pro další rozšíření.

1.2 Cíl

Cílem diplomové práce je integrace nových datových zdrojů do systému SmartPEF. Zdrojem získávaných dat bude především systém Morpheus a rezervační systém MRBS. Za tímto účelem bude systém Morpheus rozšířen o dva moduly. Prvním bude modul získávající data o softwaru nainstalovaném na univerzitních počítačích a o využívanosti tohoto softwaru. Druhým bude modul pro získání informací z rezervačního systému MRBS. Následně bude potřeba zajistit komunikaci mezi systémy Morpheus a SmartPEF pomocí naimplementování API. Všechny zmíněné systémy budou podrobně popsány v následující kapitole.

Systém SmartPEF a jeho API by měly být po dokončení této práce připraveny tak, aby mohly sloužit jako platforma pro analýzu využívanosti nainstalovaného softwaru a vytíženosti počítačů v rámci dne, týdne či semestru.

2 Analýza současného stavu

Pro tuto práci jsou klíčové tři systémy – Morpheus, MRBS a SmartPEF. V současné době tyto systémy mezi sebou nejsou nijak provázané. Pro správný návrh jejich rozšíření o novou funkcionalitu a návrh formy komunikace mezi systémy je nejprve nutné se s těmito systémy seznámit.

2.1 Systém Morpheus

Morpheus je systém pro vzdálenou správu počítačů, který je vyvíjen Ústavem informatiky Provozně ekonomické fakulty. Systém v současné době umožňuje efektivní správu powermanagementu, vzdálené bootování systémů a provádění příkazů na vzdálených počítačích, provádění naplánovaných úloh a zobrazování základních informací o spuštěném OS. Jak bylo zmíněno v úvodu práce, právě poslední zmíněná oblast poskytuje prostor pro další rozšíření, o němž bude pojednáno v kapitole 4.1.

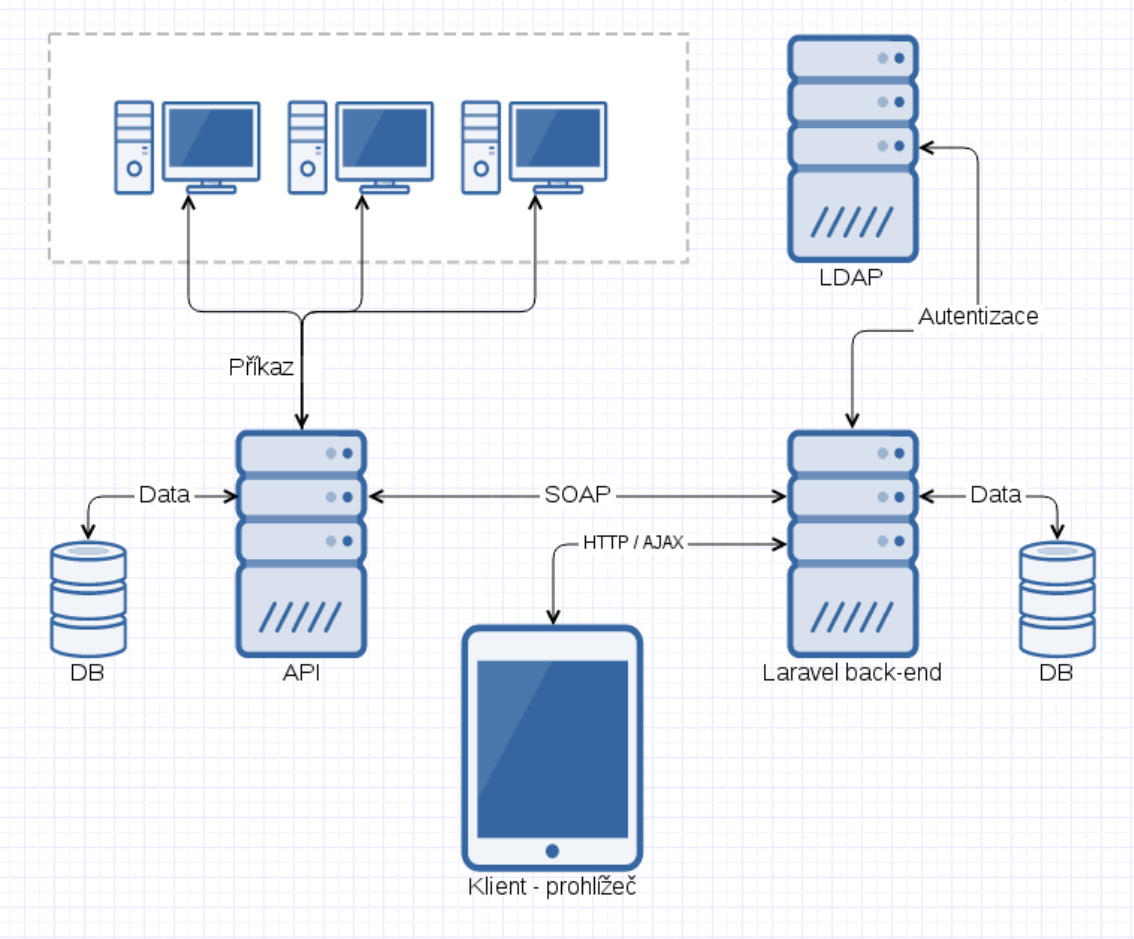
Systém je implementován v jazyce PHP. Základní schéma systému popisuje obr. 1. Jak ve své práci uvádí (Vyhnalík, 2016), nejčastější scénář komunikace v rámci aplikace je následující:

1. Klient odešle HTTP požadavek na server.
2. Server ověří oprávnění požadavku.
3. Server zpracuje požadavek a vytvoří nový, typu SOAP, který odešle na API.
4. a) API odešle daný příkaz počítačům pomocí skriptu.
   b) API aktualizuje databázi dle odezvy počítačů.
   c) API vybere z databáze data požadovaná serverem.
   d) API odešle odpověď na server.
5. Server zpracuje odpověď a odešle přes HTTP odpověď novou, ve formátu JSON nebo HTML.

Práce se bude dále zabývat především částí, která je v textu výše a na obr. 1 označována jako „API“[1] V textu bude využíváno vhodnější pojmenování „server systému Morpheus“ (nebo „server Morphea“). Tento server poskytuje SOAP API, které využívá například zmíněná webová aplikace. Vzhledem k tomu, že webová aplikace[2] je z tohoto pohledu klientem, který využívá služby serveru Morphea skrze jeho API, bude v textu dále označována jako „webový klient Morphea“. Je totiž také

[1] API (Application Programming Interface) je rozhraní aplikace (knihovny, programu či operačního systému), které ostatním programům umožňuje interakci s danou aplikací, podobně jako uživatelské rozhraní (UI) umožňuje interakci mezi člověkem a počítačem. V podstatě API definuje správný způsob, jakým může vývojář využívat služby této aplikace (Orenstein, 2000).

[2] Na obr. 1 se webová aplikace sestává z částí Laravel back-end a přidružené databáze.

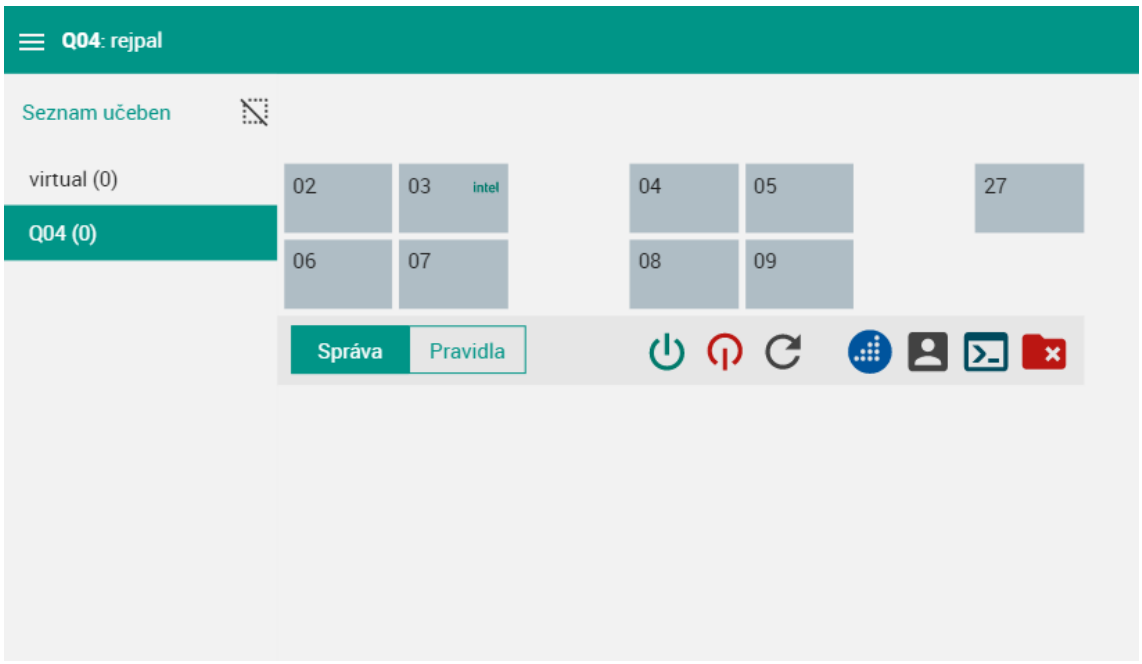


Obr. 1: Schéma komunikace v rámci systému Morpheus (Vyhnalík, 2016)

možné, že do budoucna vzniknou i další klienti (kromě webového klienta například mobilní či desktopový), kteří budou API využívat. Webový klient Morphea odstiňuje uživatele od přímého využití API serveru Morpheus a umožňuje počítače spravovat pohodlně v rámci webového prohlížeče. Uživatelské rozhraní webového klienta je zachyceno na obr. 2.

2.1.1 Architektura

Nyní si rozeberme více dopodrobna architekturu serveru systému Morpheus. Její pochopení bude stěžejní pro další rozšiřování systému. Server se sestává z několika modulů, z nichž se každý stará o specifickou část funkcionality. Každý modul je realizován pomocí tzv. traity, PHP mechanismu, který slouží k seskupování funkcionality (The PHP Group, 2017). Na rozdíl od tříd, které jsou v objektově orientovaném návrhu používány především, nemůžou být traity instanciovány. Traity jsou následně využívány ve třídách - můžeme takto simulovat vícenásobnou dědičnost,



Obr. 2: Webový klient Morphea - ukázka vykreslení počítačů do mřížky

kterou jazyk PHP nedisponuje. SOAP server Morphea se pak skládá právě z množiny veřejných metod jednotlivých modulů.

Prvním z modulů je `module_getCompInfo`, který poskytuje aktuální data o stavu počítačů. Z jeho API lze například zjistit, které počítače jsou v učebně zapnuté, jaký systém je na nich nabootovaný nebo kteří uživatelé jsou právě přihlášeni. Dalším z modulů je `module_login`, pomocí něhož lze na vzdálený počítač přihlásit libovolného uživatele. `Module_shutdown` zase umožňuje vzdálené vypínání počítačů.

Pro další práci bude důležitý `module_refresh`, který se stará o to, aby byly informace o stavu strojů v databázi Morphea aktuální. Tento modul bude v rámci práce dále rozšiřován. Pomocí programu `ping` modul zjišťuje, zda je stroj zapnutý a případně jaký systém je nabootovaný. Přes ssh[3] vykonává v příkazové řádce vzdálených počítačů příkazy, které zasílají zpět informace o přihlášeném uživateli a přítomnosti automatického přihlášení. Pro každý operační systém (Unix, Windows nebo MacOS) existuje jiná sada těchto příkazů.

`Module_refresh` obsahuje metodu `refresh`, která se v pravidelném intervalu provolává pomocí CRONu[4]. Tento interval je nastaven na čtyři minuty. Díky tomu se v databázi udržují poměrně aktuální informace o stavu strojů, z čehož profituje

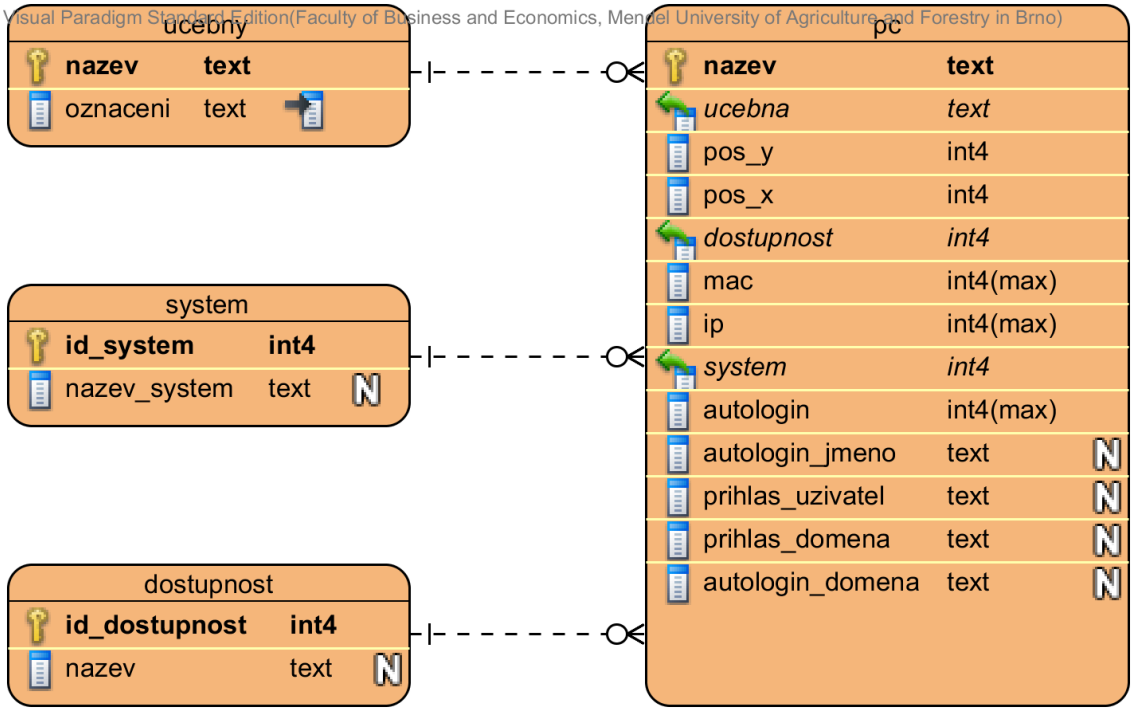
[3]Secure shell, komunikační protokol umožňující šifrovanou komunikaci mezi dvěma počítači. Zprostředkovává přístup k příkazovému řádku vzdáleného počítače (SSH Communications Security, 2017).

[4]CRON je softwarový démon, plánovač úloh, který se stará o spouštění skriptů (resp. příkazů) v naplánovaných časových intervalech (Blum, 2015).

i výše zmíněný webový klient Morphea. Délka intervalu byla zvolena experimentálně na základě měření. Do učeben spadajících pod správu Ústavu informatiky patří cca 300 počítačů. Všechny počítače jsou zpracovávány dávkově v rámci jednoho CRON jobu. Doba mezi odesláním příkazu na první počítač a zpracováním výsledku příkazu z posledního počítače se pro každé opakování CRON jobu liší. V nejhorších případech byl naměřen čas dosahující téměř čtyř minut. Proto není vhodné interval příliš zkracovat.

2.1.2 Databáze

Současné řešení serveru Morphea si vystačí s velmi jednoduchou databází PostgreSQL. Jak vidíme na obr. 3, databáze se sestává z hlavní tabulky `pc` a ze tří číselníků `ucebny`, `system` a `dostupnost`. Do `pc` se ukládají informace, které server získá ze stanic - jaký uživatel je právě přihlášený (jeho login a doména), jaký systém je nabootovaný nebo jestli je počítač aktuálně zapnutý. Kromě těchto dynamicky se měnících atributů se uchovávají v čase neměnné informace. Příkladem může být MAC adresa stroje, IP adresa[5], v jaké učebně se stroj nachází nebo poloha stroje v rámci učebny (`pos_x`, `pos_y`). Atributy `pos_x` a `pos_y` slouží pro vykreslení počítačů do mřížky ve webovém klientu Morphea (viz obr. 2). Tabulka `pc` uchovává pouze aktuální stav všech strojů - není navržena na ukládání historických informací.



Obr. 3: Schéma databáze systému Morpheus

[5]V prostředí fakulty lze považovat IP adresu za neměnnou. IP adresy jsou počítačům přidělovány DHCP serverem na základě jejich MAC adresy (tzv. statická alokace).

2.2 Meeting Room Booking System

MRBS (Meeting Room Booking System) je open-source webová aplikace vydávaná pod licencí GPL. Slouží k rezervování meetingových místností nebo jiných zdrojů. Je naprogramovaná v jazyce PHP a může využívat databázový systém MySQL nebo PostgreSQL (MRBS, 2017). MRBS je poměrně robustní aplikace. Ze zajímavých funkcí můžeme uvést například autentizaci pomocí NT domén, NIS nebo LDAP, reportování administrátorovi emailem v případě nových rezervací nebo zajištění toho, aby nemohla být zadána rezervace kolidující s rezervací jinou.

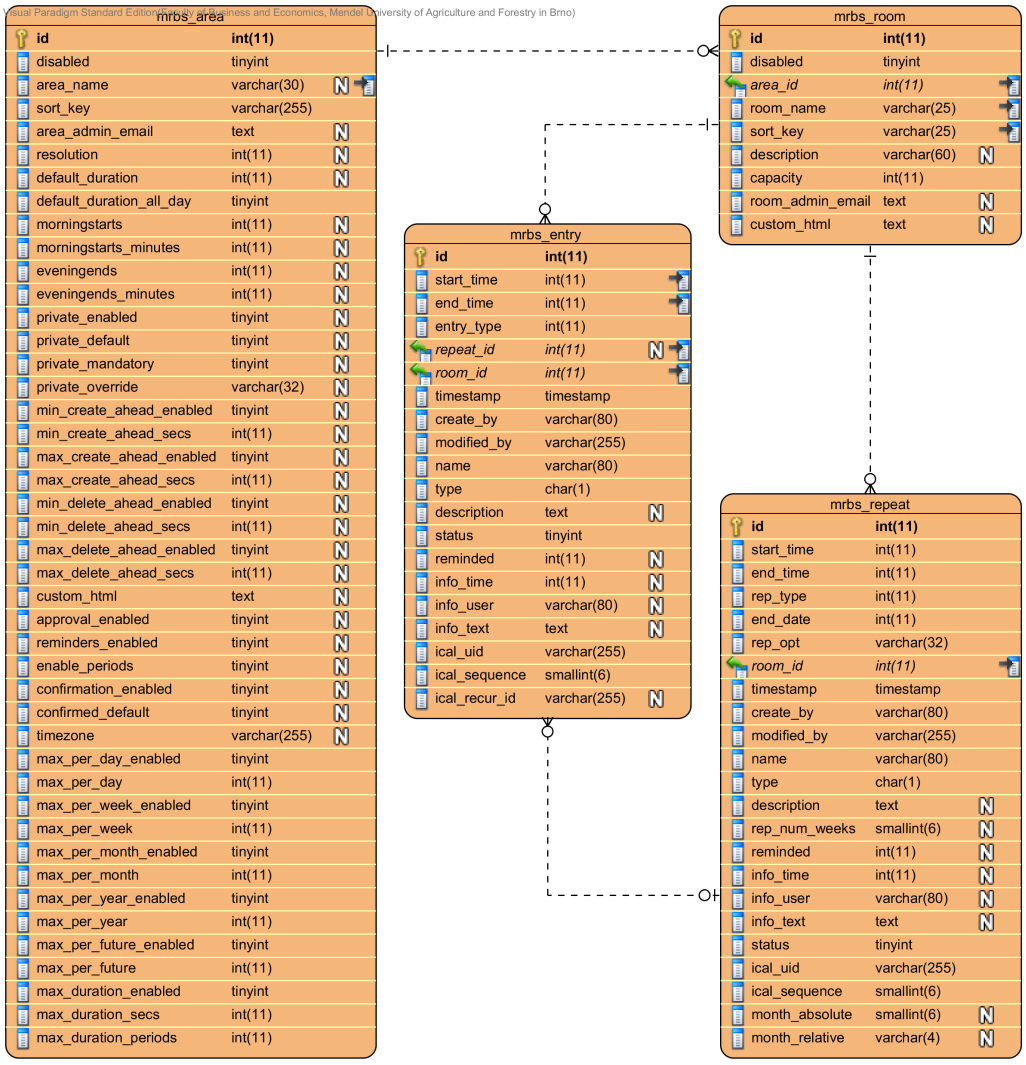
Na Ústavu informatiky je nasazená instance MRBS, která slouží pro rezervování místností ve správě Ústavu informatiky. Přes tento systém se rezervují všechny mimorozvrhové akce, tedy všechny akce kromě pravidelných cvičení a přednášek zanesených do běžného rozvrhu.

Jak vidíme na obr. 4, databáze aplikace MRBS je složena ze čtyř tabulek. Tabulka `mrbs_room` obsahuje objekty, které lze rezervovat (v našem případě místnosti na Ústavu informatiky, nicméně pomocí MRBS lze rezervovat libovolné objekty). Místnosti se sdružují do oblastí pomocí tabulky `mrbs_area`. Tato tabulka má množství atributů, ve kterých je zaznamenána konfigurace oblastí. Konfigurace je pak platná pro všechny objekty v dané oblasti. Zmínit můžeme například atribut `max_per_day`, který udává maximální možný počet rezervací denně, nebo atribut `area_admin_email` obsahující email na správce oblasti.

Stěžejní tabulky pro pochopení fungování systému jsou `mrbs_entry` a `mrbs_repeat`. Akce, které se konají v rezervovaných místnostech, mohou být buď jednorázové, nebo se mohou v pravidelných intervalech opakovat (např. jednou týdně). Pokud uživatel do systému zadá jednorázovou rezervaci, objeví se pouze v `mrbs_entry`. V případě akce, která se bude opakovat, se uloží jeden záznam do `mrbs_repeat` a poté se na základě vnitřní logiky aplikace MRBS dopočítají a vloží záznamy do `mrbs_entry`. Tak je zajištěno, že má každá akce z množiny opakujících se akcí vlastní záznam, který je samostatně editovatelný. Může tak například vzniknout série vyučovacích hodin konaných jednou týdně, s tou výjimkou, že závěrečná hodina bude o hodinu delší než ostatní. `Mrbs_repeat` pak obsahuje souhrnné informace platné pro celou sérii - např. atribut `rep_type` (typ opakování - 1 = denně, 2 = týdně, 3 = měsíčně, 4 = ročně) nebo atribut `rep_opt` ve formátu `xxxxxxx`, kde každé `x` představuje jeden den v týdnu. X je rovno jedné v případě, že v daný den rezervace platí. V opačném případě je x rovno nule. Pokud se celá série změní (například se změní frekvence rezervací z týdenní na denní), do `mrbs_entry` se napočítají nové záznamy. Staré, neplatné záznamy, se vymažou.

Tabulka `mrbs_entry` tedy obsahuje pro další práci stěžejní atributy `start_time` a `end_time` udávající začátek a konec rezervace. Čas je reprezentován jako unixový čas[6], ale s ohledem na časovou zónu platnou v České republice (UTC+1:00 v zimním období, UTC+2:00 v období letního času). Zatímco reálný unixový čas pro 5. 5. 2017

[6]Počet sekund uplynulých od 1. ledna 1970 00:00:00 UTC (The IEEE and The Open Group, 2016).



Obr. 4: Schéma databáze systému MRBS

14:30 UTC+2:00 by měl být 1493987400, do MRBS se uloží jako 1493994600 - tedy jako 5. 5. 2017 14:30 UTC+0:00. Na toto specifikum bude potřeba myslet při tvorbě modulu a jeho API.

2.3 Systém SmartPEF

SmartPEF je nově vznikající koncept a zároveň informační systém, který si klade za cíl studentům a zaměstnancům Provozně ekonomické fakulty poskytnout informace o fakultě a jejím provozu. Tento koncept staví na třech pilířích. Těmi jsou serverová infrastruktura, senzorická síť a mobilní zařízení pro zprostředkování informací uživatelům.

Systém SmartPEF sestává z několika samostatných modulů, z nichž některé mezi sebou komunikují. Vývojáři jednotlivých modulů se snaží dodržovat stejnou

architekturu, aby byly moduly snáze udržovatelné. Základem každého modulu jsou části `core` a `api`. `Core` obsahuje aplikační logiku a komunikuje s databází. `Api` využívá služby `core`. Veřejnou část modulu představuje REST API, které uživatelům poskytuje možnost manipulovat s daty shromažďovanými v databázi. Dopodrobna bude architektura popsána v kapitole 4.2. Je vhodné, aby bylo REST API dobře zdokumentované a snadno použitelné, protože data ze SmartPEFu budou poskytována vývojářům, kteří na jejich základě budou budovat nové aplikace.

Architektura sjednocená napříč moduly skýtá mnohé výhody, přestože je každý modul vyvíjen nezávisle. Všechny moduly například využívají společný autentizační OAuth 2.0 server. Pro vývojáře je navíc snazší pracovat na více různých modulech, které jsou založené na stejné architektuře.

V systému SmartPEF je momentálně zařazeno několik modulů. Jako příklad uveďme modul `endpoints`, který sbírá a zpracovává data ze senzorů umístěných na Provozně ekonomické fakultě (teploměry, tlakoměry...) nebo modul `FAQ`, který shromažďuje a zpracovává často kladené dotazy z prostředí fakulty.

2.4 Požadavky

Po úvodních konzultacích se zadavatelem vzešly následující funkční a nefunkční požadavky. Požadavky se týkají všech rozšiřovaných i nově vytvářených systémů a modulů, včetně komunikace, která mezi nimi má probíhat. Zadavatelem práce je oddělení infrastruktury Ústavu informatiky PEF Mendelovy Univerzity v Brně.

2.4.1 Funkční požadavky

- **Modul Refresh systému Morpheus** pro sběr dat z počítačů. Prostřednictvím tohoto modulu je potřeba sbírat následující data z provozu počítačů na Provozně ekonomické fakultě:
  - názvy počítačů v učebnách;
  - rozmístění počítačů v rámci učebny;
  - stav počítače (zapnuto/vypnuto, jaký operační systém je nabootovaný);
  - obsazenost počítače, tedy zda je někdo přihlášen - jak lokálně, tak vzdáleně (např. přes funkci Připojení ke vzdálené ploše na OS Windows). V jeden moment může být na počítači přihlášeno i více uživatelů. Tento fakt by měl systém Morpheus také podchytit;
  - software nainstalovaný na jednotlivých počítačích. Seznam evidovaného softwaru by mělo být možno omezit pomocí zařazení softwaru na blacklist. Blacklist zavádíme z toho důvodu, že není žádoucí zaznamenávat veškerý software. Z pohledu analýzy využívanosti softwaru nás například nezajímají systémové aplikace, ovladače ani systémové aktualizace;

  - procesy spuštěné na počítači, včetně jejich spárování se softwarem nainstalovaným na daném počítači. Seznam sledovaných procesů, podobně jako software, by měl jít omezit pomocí blacklistu. U procesů by se měl zaznamenávat i jejich vlastník.

- **Data pro analýzu**. Výše zmíněná nasbíraná data by měla umožňovat zodpovězení následujících otázek:
  - Kdy je učebna X vytěžována nejvíce (den v týdnu, doba ve dni, část semestru)?
  - Která učebna je nejvíce vytížená během dne?
  - Byl někdy na počítači X nainstalován software Y?
  - Jaký software je nainstalován na počítačích/učebnách X?
  - Kde je nainstalován software X?
  - Jaký počítač v učebně je nejčastěji využíván?
  - Jaký software je spouštěn nejčastěji na počítači/učebně X?
  - Kdo byl na počítači X přihlášen v době Y?
  - Jak často byl na počítači X spouštěn program Y?
  - Který software patří mezi nejčastěji využívané?
  - Jaký je nejčastěji bootovaný systém?
- **Modul MRBS systému Morpheus** pro komunikaci s rezervačním systémem MRBS. Díky tomuto modulu a jeho API musí být možné získat informace o obsazenosti učeben, např. která učebna je zarezervovaná pro akce mimo rozvrh a v jaké časy. V kombinaci s daty ze systému Morpheus půjde zjistit, kdy je počítač s požadovaným softwarem volný pro případnou práci studenta.
- **Modul Morpheus systému SmartPEF**. Bude potřeba vytvořit část systému SmartPEF, která bude čerpat data ze systémů Morpheus a MRBS. Tento modul bude sloužit jako platforma pro analýzu sesbíraných dat a jako podklad pro v budoucnu vyvíjené aplikace určené pro studenty a zaměstnance PEF.
- **Agent pro nahrávání dat do SmartPEF**. Výše zmíněná data budou pravidelně posílána do systému SmartPEF, který bude navržen pro ukládání velkého objemu historických dat. Při přenosu je nutné zajistit konzistenci dat. Agent by měl být plně automatizovaný pomocí plánovače úloh.
- **Rozhraní pro přístup k systémům**. Pro systémy Morpheus i SmartPEF by mělo být vytvořeno API, které umožní uživatelům (včetně ostatních systémů a aplikací) přistupovat k uloženým datům.

- **Autorizovaný přístup k datům**. Vzhledem k povaze uchovávaných dat je nutné kontrolovat, kdo k datům bude přistupovat. Uchovávat se budou např. přihlašovací jména uživatelů nebo IP adresy, což lze považovat za potenciálně zneužitelné osobní údaje. Proto bude většina dat přístupná jen autorizovaným osobám. Data je potřeba zabezpečit v systému Morpheus, v modulu Morpheus systému SmartPEF i během přenosu mezi systémy. API by mělo zajišťovat i anonymizaci předkládaných dat - tedy např. analytik bude moci přes API přistoupit pouze k agregovaným a anonymizovaným datům, ale oprávněný administrátor uvidí data neanonymizovaná.

2.4.2 Nefunkční požadavky

- **Minimalizování zátěže na klientských stanicích** (počítačích, ze kterých se sbírají data). Veškeré výpočty, pokud to bude možné, by se měly zpracovávat na serveru. Jako výpočty jsou zde myšleny například matchování procesů a nainstalovaného softwaru nebo omezení sledovaného softwaru pomocí blacklistu.
- **Minimalizování zátěže na univerzitní síti**. Univerzitní síť by neměla být přetěžována přenášením zbytečně velkého objemu dat nebo zahlcena velkým množstvím komunikace, ať už mezi počítači a serverem Morphea nebo mezi Morpheem a systémem SmartPEF. Proto je nutné rozlišit, která data je účelné sbírat a která již ne. Tento požadavek jde svou logikou proti požadavku minimalizování zátěže na klientských stanicích, zmíněnému výše. Proto bude potřeba nalézt rovnovážný stav, který bude vyhovovat oběma těmto požadavkům.
- **Morpheus jako agentless[7] systém**. Systém Morpheus by měl nadále fungovat bez nutnosti instalovat na klientské stanice jakýkoliv dodatečný software. Instalace a údržba softwaru na více než 300 stanicích by totiž vyžadovala značné úsilí správců. Morpheus by tedy měl využívat pouze prostředky, které jsou v současnosti dostupné v operačním systému klientských stanic.
- **Rozšiřitelnost**. Modul Morpheus systému SmartPEF by měl být snadno rozšiřitelný o nové datové zdroje a novou funkcionalitu. Je totiž pravděpodobné, že časem budou datové zdroje přibývat, stejně tak jako budou přibývat případy užití jeho API a do API bude tím pádem nutné přidávat nové operace.
- **Snadná použitelnost API**. Jak systém Morpheus, tak SmartPEF by měly poskytovat API, která budou ideálně samopopisná a pochopitelná pro programátorskou veřejnost. Systémy mezi sebou musí komunikovat pomocí obecně

[7] Jak zmiňuje (TechTarget, 2017), pod pojmem agentless se rozumí stav, kdy na stroji, na kterém se provádí daná operace (v našem případě tedy na vzdáleném počítači), neběží v pozadí žádná služba, démon nebo proces (neboli agent). V praxi je tento stav téměř nemožný, protože všechny počítače potřebují pro vykonání operace nějaký program, který lze nazvat agentem. Operace však mohou být obecně prováděny řídícím strojem, který používá agenta, zatímco cíle řídícího stroje lze označit jako agentless - nemusí se na ně instalovat nový software související s prováděnou operací.

známých protokolů a přijímat a poskytovat data ve známých a dobře strukturovaných formátech (např. JSON, csv, XML). Rozhraní by měla být dobře zdokumentovaná, včetně popisu struktury přijímaných a poskytovaných dat.

3 Metodika řešení

Pro dosažení stanovených cílů práce bude potřeba postupovat podle následujících kroků.

1. Prozkoumání možností, kterými je možné zjistit informace z provozu vzdálených počítačů, zejména informace o nainstalovaném softwaru a jeho využívanosti. Navržení způsobu, jakým se budou spojovat informace o nainstalovaném softwaru s běžícími procesy, abychom získali využitelná data.
2. Implementace skriptů, které budou spouštěny na vzdálených počítačích a které budou zjišťovat výše zmíněné informace. Testování skriptů na různých verzích operačního systému Microsoft Windows.
3. Návrh nových modulů systému Morpheus, návrh rozšíření databáze a návrh vhodného API na základě funkčních a nefunkčních požadavků.
4. Implementace modulů (s využitím výše zmíněných skriptů) a API v jazyce PHP na základě návrhu. Integrace nových modulů do systému Morpheus.
5. Návrh architektury, databáze a REST API modulu Morpheus systému SmartPEF na základě funkčních a nefunkčních požadavků.
6. Implementace navrženého modulu Morpheus systému SmartPEF v jazyce Java s využitím frameworku Spring. Implementace REST API včetně řešení autentizace a autorizace.
7. Návrh komunikace mezi systémy Morpheus a SmartPEF, včetně analýzy objemu přenášených dat. Návrh frekvence a formátu přenášení dat.
8. Implementace agenta, který realizuje výměnu dat mezi systémy Morpheus a SmartPEF. Agent bude implementován v jazyce PHP a jeho úlohy spouštěny pomocí CRONu.
9. Otestování a nasazení funkčního řešení do produkce.

4 Návrh řešení

Na základě analýzy současného stavu a požadavků od zadavatele bude navrhnuto nové řešení vyhovující aktuálním požadavkům. Navrhované řešení sestává z rozšíření systému Morpheus (kapitola 4.1) a vytvoření modulu Morpheus systému SmartPEF (kapitola 4.2). Na závěr se návrh bude zabývat zajištěním komunikace mezi zmíněnými systémy (kapitola 4.3).

4.1 Systém Morpheus

Současný stav systému Morpheus byl nastíněn v kapitole 2.1. Na základě analýzy funkčních požadavků v kapitole 2.4 je potřeba systém rozšířit o modul pro komunikaci s rezervačním systémem MRBS. Dále je nutné rozšířit existující `module_refresh` o funkcionalitu zajišťující sběr dat z klientských stanic o využívanosti softwaru a uživatelské aktivitě. Za účelem rozšíření systému Morpheus bude dále provedena revize současné databáze a navrhnuto nové řešení vyhovující novým požadavkům. V neposlední řadě je potřeba navrhnout API, pomocí kterého bude Morpheus komunikovat s okolními systémy.

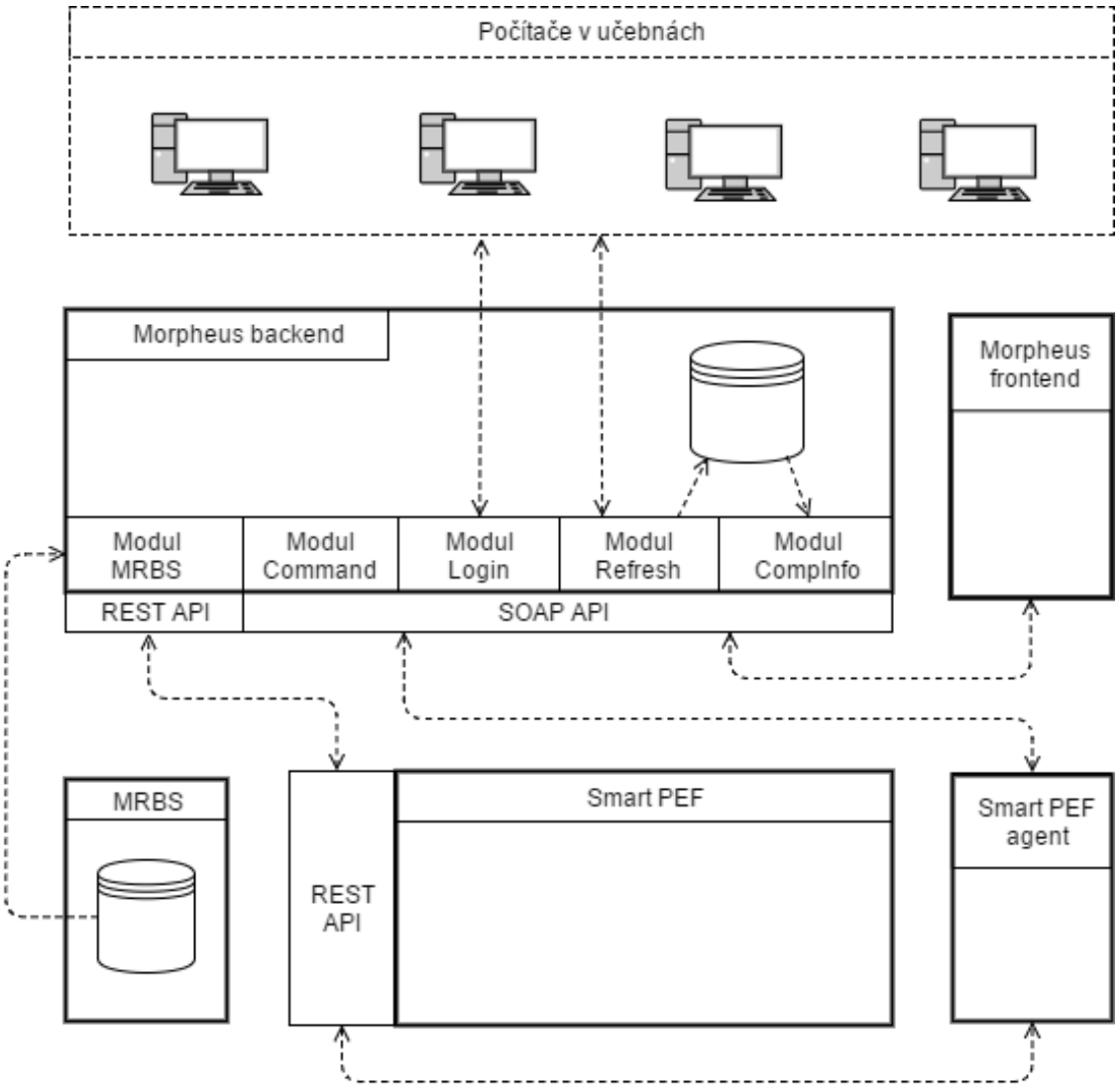
4.1.1 Architektura

V současnosti se server Morphea skládá z šesti modulů, z nichž ty nejdůležitější jsou zobrazeny na obr. 5 v části Morpheus backend a jejichž funkcionalita byla rozebrána v kapitole 2.1.1. Jak bylo zmíněno, každý modul je realizován pomocí PHP mechanismu trait. SOAP server Morphea se skládá z množiny veřejných metod jednotlivých modulů. Nový modul bude implementován v PHP, stejně jako celý systém Morpheus.

4.1.2 Modul MRBS

Nově vytvářený modul MRBS by měl umožnit komunikaci s rezervačním systémem MRBS. Protože MRBS je aplikace, která neposkytuje žádné API, bude nutné přistupovat k databázi MRBS přímo, jak znázorňuje obr. 5. V kapitole 2.2 byla provedena analýza této databáze. Bez této znalosti by nebylo možné postupovat dále v návrhu. Po diskuzi se zadavatelem byly navrhnuty čtyři metody poskytující informace o stavu rezervací. Tyto metody budou dostupné ostatním systémům skrze API (více v kapitole 4.1.6).

Pomocí metody `isPCFree` lze zjistit, zda je v požadovaném čase daný počítač volný. Slovo volný zde chápeme jako stav, kdy učebna, v níž se počítač nachází, není v daný čas zarezervována. Pro toto zjištění musíme zkombinovat data z Morphea (zjistit, v jaké učebně se počítač nachází) a z MRBS (zjistit, zda je učebna s dotazovaným počítačem v daný čas volná). Tato informace se hodí například studentům, kteří potřebují v určitém čase pracovat na semestrálním projektu na konkrétním počítači.



Obr. 5: Schematické znázornění komunikace mezi systémy Morpheus, MRBS a SmartPEF

Následuje trojice metod `getActualReservation`, `getPreviousReservation` a `getNextReservation`. Všechny tři přebírají dva parametry - `room` a `timestamp`. Pomocí parametru `room` určíme učebnu, na jejíž rezervace se dotazujeme. Parametr `timestamp` se vztahuje k času, pro který chceme zjistit existenci rezervace. Metoda `getActualReservation` nalezne rezervaci učebny pro zadaný čas (`timestamp`). Pokud v daný čas učebna zarezervovaná není, vrátí `false`. Metoda `getPreviousReservation` vrátí nejbližší rezervaci před zadaným časem. Pokud je tedy učebna rezervována od 10:00 do 12:00 a od 14:00 do 16:00 a dotaz míří na čas 13:00, metoda vrátí první rezervaci - od 10:00 do 12:00. Analogicky, metoda `getNextReservation` nalezne nejbližší rezervaci po zadaném čase. Informace získané z těchto metod budou později využity ve webovém klientu Morphea a v mo-

dulu Morpheus systému SmartPEF. Mohou pomoci například v případě, kdy chce správce učeben pomocí Morphea nastavit pravidla pro automatické vypínání a zapínání počítačů tak, aby před důležitými rezervacemi byly počítače vždy zapnuté. Potenciálně se tak může urychlit začátek semináře, protože účastníci nebudou muset počítače zapínat manuálně - budou zapnuté s předstihem.

S dotazováním na rezervaci učeben se pojí jeden problém. Systém Morpheus má seznam učeben uložen v tabulce `ucebny` (viz obr. 6), zatímco MRBS má svůj vlastní seznam v tabulce `mrbs_room` (viz obr. 4). Bohužel, na každém z těchto seznamů je název učeben uveden v rozdílném tvaru (např. „Q04“ „Q04 - Macy“). Pro snazší komunikaci mezi systémy bude nezbytné pojmenování sjednotit. Sjednocení názvů učeben je nutné také v případě, kdy se uživatel dotáže na volný počítač (metoda `isPCFree`). Informace o tom, v jaké učebně se nachází daný počítač, je totiž v systému Morpheus, zatímco informace o rezervacích konkrétních učeben je v MRBS. Abychom mohli tyto informace automatizovaně propojit, je nutné názvy sjednotit.

4.1.3 Modul Refresh

Druhým modulem, kterým se bude návrh zabývat, je modul Refresh. Tento v současné době umožňuje získávat z klientských stanic informace o stavu stroje - jestli je počítač zapnutý, jaký systém je nabootovaný a který uživatel je přihlášený. Modul je potřeba rozšířit o novou funkcionalitu dle funkčních požadavků v kapitole 2.4. Modul bude obsahovat dvě hlavní funkce - `refreshInfo` a `refreshSoftware`. `RefreshInfo` aktualizuje informace o stavu strojů, `refreshSoftware` získá informace o nainstalovaném softwaru.

`RefreshInfo` se spouští pomocí CRONu každé čtyři minuty. Zjistí stav všech počítačů na fakultě. U zapnutých počítačů zjistí přihlášené uživatele. Pokud je na některém stroji přihlášen uživatel, spustí skript na získání informací o procesech běžících pod daným uživatelským účtem. Vše následně uloží do databáze (viz kapitola 4.1.4). Procesy, které nechceme zaznamenávat, můžeme omezit pomocí blacklistu. Ten je realizován pomocí textového souboru uloženém v souborovém systému serveru Morpheus, ke kterému mají přístup pouze administrátoři Morphea. Formát souboru blacklistu je následující - na každém řádku se nachází název jednoho procesu, který má být zařazen na blacklist.

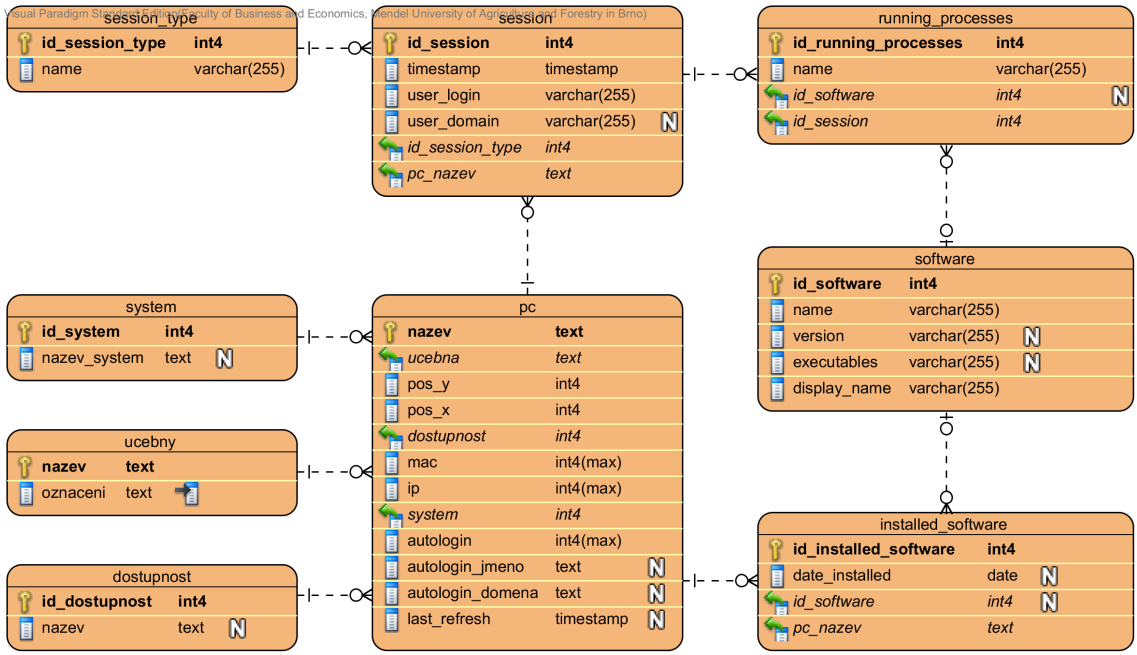
Funkce `refreshSoftware` se také spouští pomocí CRONu, ale pouze jednou týdně. Stará se o udržování aktuálních informací o softwaru, který je nainstalován na jednotlivých počítačích. Tyto informace není nutné na Morpheovi uchovávat historicky. Jak je nastíněno v kapitole 4.3.3, tato data se budou v týdenních intervalech nahrávat do SmartPEF, kde bude uložená i jejich historie pro potřeby analýzy. Proto se data o nainstalovaném softwaru můžou při každém spuštěním funkce `refreshSoftware` vymazat a nahradit novými, aktuálními. U automatizovaného spouštění funkce `refreshSoftware` je potřeba myslet na důležitý fakt - počítače musí být v době, kdy se funkce spouští, zapnuté. To je možné zajistit pomocí existující funkcionality Automatické spouštění akcí, kterou poskytuje webový klient Mor-

phea (Vyhnalík, 2016). Na den a čas, kdy má být spuštěna funkce `refreshSoftware`, bude nastaveno pravidlo udávající, že se v daný čas mají zapnout všechny počítače na fakultě. Analogicky nastavíme i čas, kdy se mají počítače vypnout (musíme však počítat s tím, že zpracování všech počítačů bude nějakou dobu trvat). Automatické zapínání a vypínání počítačů by mělo být nastaveno na dobu, kdy se s počítači nepracuje - tedy ideálně v nočních hodinách během víkendu. I přesto by se měla provést kontrola, zda k počítači není v daný moment přihlášen nějaký uživatel, kterému bychom počítač vypnuli pod rukama.

Podrobnosti o implementaci postupu získávání informací o běžících procesech a o nainstalovaném softwaru budou dále rozebrány v kapitolách 5.1 a 5.2.

4.1.4 Návrh databáze

Současná databáze Morphea (obr. 3) je vyhovující pro stávající řešení, ale vzhledem k plánovanému rozšíření systému Morpheus bude muset být značně upravena. Návrh nové struktury databáze je zobrazen na obr. 6.



Obr. 6: Návrh nové struktury databáze systému Morpheus

První úprava se týká přihlášených uživatelů. Současné řešení umožňuje uchovávat pro každý stroj nejvýše jednoho přihlášeného uživatele. Jak vyplynulo z funkčních požadavků, mohou nastat situace, kdy bude k jednomu počítači přihlášeno více uživatelů - např. přes vzdálenou plochu. Tyto situace v novém návrhu pokrývá tabulka `session`, která je s tab. `pc` ve vazbě 1:N. Díky této tabulce mohou být z tab. `pc` odstraněny atributy `prihlas_uzivatel` a `prihlas_domena`. Kromě loginu přihlášeného uživatele tabulka `session` uchovává i doménu, pod kterou se uživatel přihlásil

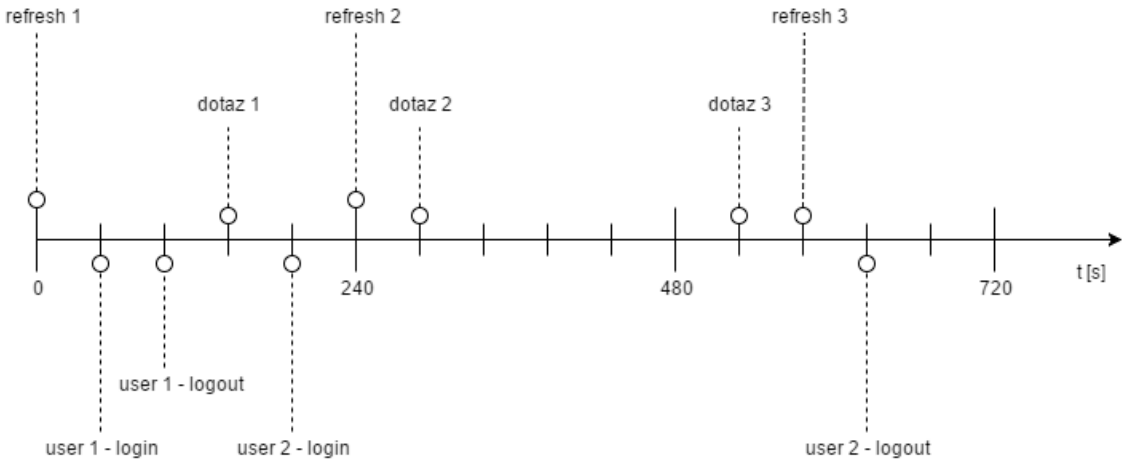
a také typ přihlášení (lokální/vzdálené). Tabulka je navržená tak, aby mohla uchovávat historické informace. Při každém skenu počítačů v univerzitní síti se do tabulky uloží nové záznamy o aktuálně přihlášených uživatelích. Staré záznamy se z tabulky nemažou. Nové záznamy jsou opatřené časovým razítkem z doby jejich pořízení, aby bylo později možné sestavit časový průběh přihlášení (např. odhadnout, jak dlouho byl uživatel na počítači přihlášen).

Dále je dle funkčních požadavků nutné uchovávat informace o nainstalovaném softwaru. K tomu slouží tabulky `software` a `installed_software`. Informace o softwaru jsou rozdělené do dvou tabulek, aby databáze splňovala 2. normální formu. Tabulka `software` obsahuje seznam unikátních programů na všech univerzitních počítačích. Tabulka `installed_software` pak funguje jako vazební tabulka pro realizaci M:N vazby `pc-software`. Pomocí cizích klíčů `id_software` a `pc_nazev` tedy určuje, jaký software je nainstalován na jakém počítači.

Poslední nutnou úpravou je rozšíření databáze o možnost uchovávat procesy běžící na univerzitních počítačích. Vzhledem k faktu, že potřebujeme znát i vlastníky procesů, bude vhodné procesy navázat na konkrétní `session`. V rámci `session` se totiž nachází i informace o právě přihlášeném uživateli, který je vlastníkem spuštěných procesů. Celé řešení vidíme na obr. 6. Tabulka `running_processes` slouží jako vazební tabulka mezi `session` a `software`. `Session` obsahuje časové razítko, z databáze tedy můžeme zjistit, kdy byl každý proces spuštěn. V ideálním případě by měl být každý proces přiřazen k softwaru, který ho spouští (pomocí cizího klíče `id_software`). Protože v praxi je však obtížné najít software patřící k danému procesu, je atribut `id_software` nepovinný. O spojování procesů a softwaru bude pojednáno v sekci 5.3.

4.1.5 Optimalizace frekvence obnovování informací

Výše navržená změna databáze (umožnit přihlášené uživatele uchovávat v `session`, tabulce s historií, namísto stavově v tabulce `pc`) s sebou přináší otázku, jak lze spolehlivě určit uživatele, který je právě přihlášen. Dokud byl uživatel uveden v tabulce `pc`, která se aktualizovala při každém proskenování sítě, měli jsme jistotu, že v atributu `prihlas_uzivatel` nalezneme buď přihlášeného uživatele, nebo `NULL` (v tom případě není na počítači přihlášen žádný uživatel). Při navrhnuté změně je situace jiná - při skenování sítě se do tabulky `session` k danému PC vloží jeden záznam za každého uživatele aktuálně přihlášeného k PC. Problémem ovšem je, že pro zjištění aktuálně přihlášených uživatelů nestačí vybrat z tabulky `session` nejnovější záznamy pro daný počítač. Nejnovější záznamy totiž mohou pocházet např. z času před třiceti minutami, kdy uživatel sice ještě přihlášen byl, ale následně se odhlásil a v posledních třiceti minutách byl odhlášený. Pro odhlášené uživatele se z logiky věci v databázi žádné záznamy neuchovávají. Otázkou tedy je, jak určit aktuálně přihlášeného uživatele? Nabízí se dvě možnosti. Obě budou porovnány na scénáři nastíněném na obr. 7.



Obr. 7: Scénář přihlašování uživatelů na jednom PC

První možností je využít znalost délky intervalu, v jakém se aktualizují informace o přihlášených uživatelích. Jak je uvedeno v kapitole 2.1, informace se aktualizují každé čtyři minuty. Pokud bychom tedy z tabulky `session` vybrali záznam s nejnovějším časovým razítkem (atribut `timestamp`) pro daný počítač, a tento záznam byl zároveň z posledních čtyř minut, měli bychom získat aktuálně přihlášeného uživatele. Toto tvrzení platí v časech označených na obr. 7 jako `dotaz 1` a `dotaz 2`. Bohužel časový interval mezi aktualizacemi konkrétního počítače se někdy může prodloužit. Počítače se totiž v rámci dvou dávek nezpracovávají vždy ve stejném pořadí, může se tedy stát, že v jedné várce bude konkrétní PC zpracován mezi prvními, zatímco ve várce druhé bude zpracován jako poslední. Interval pro daný počítač se tedy prodlouží - tato situace na obrázku 7 nastává mezi událostmi `refresh 2` a `refresh 3`, kdy se právě `refresh 3` opozdí. Na `dotaz 3` tedy získáme nesprávnou odpověď. `User 2` se totiž jeví jako odhlášený, protože pro poslední čtyři minuty neexistuje v tabulce `sesssion` žádný záznam (kvůli zpoždění refreshe). Pokud by `refresh 3` proběhl beze zpoždění (v čase t = 480), dostali bychom správný výsledek - na PC je přihlášen `user 2`. Vzhledem k faktu, že se interval aktualizace jednoho PC může prodloužit, není možné se na fixní délku intervalu a tím pádem ani na tuto možnost spolehnout.

Proto přistoupíme k druhé možnosti, ve které se nepracuje s nespolehlivým údajem délky intervalu. Jak jsme zjistili, samotný atribut časového razítka `timestamp` v kombinaci se znalostí délky intervalu nám stačit nebude. Potřebujeme znát přesný čas, kdy byl poslední refresh proveden. Tato informace nikde uložená není, proto byl při návrhu databáze (obr. 6) do tabulky `pc` přidán atribut `last_refresh`. Při aktualizaci konkrétního PC se hodnota atributu `last_refresh` pro daný PC přepíše na aktuální čas. Stejný čas se zanese do atributu `timestamp` v tabulce `session` pro všechny nově přidané záznamy o přihlášených uživatelích. Tím tedy zjistíme čas, kdy byl počítač naposledy aktualizován a zda k tomuto času existuje nějaký přihlášený uživatel. Při dotazu na přihlášeného uživatele pak z tabulky `session` bereme jen zá-

znamy, u nichž se časové razítko shoduje s časem poslední aktualizace dotazovaného počítače. `Dotaz 2` i `dotaz` 3 nyní vrátí správný výsledek - na počítači je přihlášen `user 2`.

Ani jedna možnost bohužel neřeší situaci, kdy se uživatel přihlásí a odhlásí mezi dvěma aktualizacemi, které následují bezprostředně po sobě. Na obr. 7 tuto situaci zachycuje interval mezi událostmi `user 1 - login` a `user 1 - logout`. Z důvodů uvedených v kapitole 2.1 víme, že systém Morpheus bohužel nemá tak vysokou úroveň rozlišení, aby tuto situaci dokázal pokrýt. Museli bychom zkrátit interval mezi aktualizacemi, a takové množství požadavků by server při současné konfiguraci nestíhal vyřizovat. Zkrácení intervalu by navíc zatěžovalo univerzitní síť, což je v rozporu s nefunkčními požadavky uvedenými v kapitole 2.4.2. O zkrácení intervalu bychom mohli uvažovat po zajištění adekvátních zdrojů (šířka pásma sítě, výkon serveru Morpheus...).

4.1.6 Návrh API

Než přistoupíme k samotnému návrhu API, seznámíme se s dvěma hlavními možnostmi tvorby API - REST a SOAP. U obou možností zmíníme jejich výhody i nevýhody.

SOAP (Simple Object Acces Protokol) je standardizovaný protokol sloužící k výměně zpráv mezi aplikacemi. Specifikace obsahuje XML obálku, do které jsou zabalené přenášené informace, a soubor pravidel pro překlad datových typů specifických pro danou aplikaci na reprezentaci XML (Snell, Tidwell a Kulchenko, 2002). Protokol byl vytvořen v roce 1998 ve spolupráci s firmou Microsoft. Vzhledem k tomu, že byl vytvořen velkou softwarovou společností, zaměřuje se tento protokol na řešení potřeb podnikového trhu. Každá operace poskytovaná webovou službou prostřednictvím protokolu SOAP je explicitně definována, včetně přesné XML struktury požadavku a odpovědi. Podobně striktně jsou definovány vstupní parametry dané operace, které musí být požadovaného datového typu. Všechna tato pravidla jsou zakódována ve WSDL (Web Service Description Language). Poskytnout ke svému API i WSDL sice není povinné, ale jeho zveřejnění programátorům výrazně usnadňuje komunikaci s webovou službou. Pokud se totiž vývojář rozhodne využít SOAP webovou službu, vše, co potřebuje znát, je WSDL (Patni, 2017).

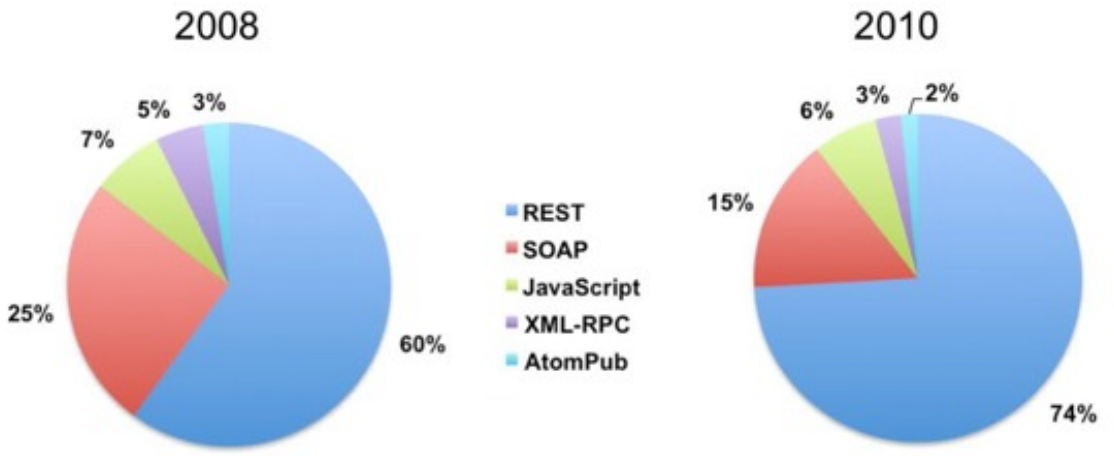
Výhodou SOAP je, že může fungovat na libovolném komunikačním protokolu - např. HTTP, SMTP nebo JMS. V praxi se však nejvíce využívá protokol HTTP, rozšířený s nástupem internetu. Další výhodou je, že součástí protokolu je i zabezpečení a autorizace. Jako poslední výhodu můžeme zmínit fakt, že SOAP může být plně popsán pomocí WSDL.

Striktní specifikace ovšem přináší i nevýhody. Jednak je implementace SOAP pro vývojáře poměrně náročná, jednak je při komunikaci pomocí SOAP nutné přenášet poměrně velké množství metadat (XML hlavičky, specifikace datových typů apod.). Byť se na první pohled může zdát WSDL dobrým prostředkem, díky kterému je API samopopisné, přináší také nevýhodu. Vzhledem k tomu, že WSDL

určuje přesnou podobu komunikace mezi poskytovatelem služby a jeho zákazníky, každá změna v API (a tím pádem i ve WSDL) nutně ovlivní i zákazníky (Patni, 2017). Ti musí při každé změně WSDL přizpůsobit své klientské aplikace novému WSDL. Tento fakt klade na vývojáře poměrně velké nároky. Další nevýhodou je nízká rychlost zpracování XML.

Druhým přístupem pro tvorbu API je REST (REpresentational State Transfer). REST lze označit za architektonický styl. Na rozdíl od SOAP to není standardizovaný protokol vymezující striktní pravidla komunikace. Je to spíše sada doporučení (Patni, 2017). REST využívá existující a široce rozšířené technologie a nevytváří žádné nové standardy. Pro komunikaci využívá protokol HTTP a jeho metody GET, POST, PUT a DELETE, s jejichž pomocí manipuluje s daty. Zatímco SOAP je zaměřený především na funkce, REST se zaměřuje na data.

Výhodou RESTu je jeho poměrně snadná implementace a údržba. Dále zcela jasně odděluje implementaci klienta a serveru. Další výhodou je, že oproti SOAP, který pro formát zpráv využívá výhradně XML, může REST zprávy strukturovat do více různých formátů. Nejobvyklejším z nich je JSON, lze ale využít i XML, YAML nebo jiný strojově čitelný formát. Oproti SOAP je REST ve webových službách využívané čím dál častěji (viz srovnání na obr. 8).



Obr. 8: Distribuce protokolů a stylů API (DuVander, 2010)

Nevýhodou REST je fakt, že funguje pouze s HTTP protokolem. Také implicitně neobsahuje zabezpečení a autorizaci. Na rozdíl od SOAP musíme zabezpečení řešit sami.

Nyní, když známe hlavní rozdíly a výhody jednotlivých přístupů, přistoupíme k samotnému návrhu API serveru Morpheus. Jak bylo zmíněno v kapitole 2.1, Morpheus poskytuje veřejnou část modulů pomocí protokolu SOAP. Toto rozhraní ve velké míře využívá zejména webový klient Morphea. V týmu vyvíjející systém Morheus je naplánováno časem předělat existující rozhraní na REST, aby byly informace zveřejněné pomocí API snáze dostupné ostatním programátorům v univer-

zitním prostředí (jak bylo zmíněno výše, REST API je pro implementaci obvykle snazší než SOAP). Nicméně do té doby musí být funkcionalita všech aplikací využívající API serveru Morpheus zachována. To se týká hlavně rozšiřovaného modulu Refresh, který bude i nadále s okolím komunikovat pomocí protokolu SOAP.

Co se týče nově vytvářeného modulu MRBS, situace se oproti modulu Refresh liší. Tento modul je nový a tím pádem zatím neexistuje aplikace, která by byla na jím poskytovaném API závislá. Z tohoto ohledu si můžeme dovolit zvolit libovolný ze dvou výše zmíněných přístupů (REST nebo SOAP). Nicméně musíme vzít v úvahu způsob, jakým bude API modulu do budoucna využíváno. Jak víme z funkčních požadavků, bude využito v systému SmartPEF. Dále bude využito webovým klientem Morphea a je možné, že do budoucna budou API využívat i jiné aplikace. Použití protokolu SOAP by znamenalo implementaci SOAP serveru na straně Morphea a specializovaných SOAP klientů na straně SmartPEF, webového klienta a všech ostatních aplikací využívající toto API. Jak bylo zmíněno výše, implementace SOAP je poměrně náročná. Z tohoto důvodu bude vhodnější zvolit REST API. Druhým důvodem je výše zmíněný fakt, že se bude současné API Morphea časem také předělávat na REST. Je tedy kontraproduktivní vytvářet SOAP rozhraní, které se bude vzápětí předělávat na REST. V poslední řadě můžeme zmínit, že použití REST API klade menší nároky na datový přenos (byť v našem měřítku je rozdíl zanedbatelný).

Komunikace mezi systémy prostřednictvím API je schematicky znázorněna na obr. 5.

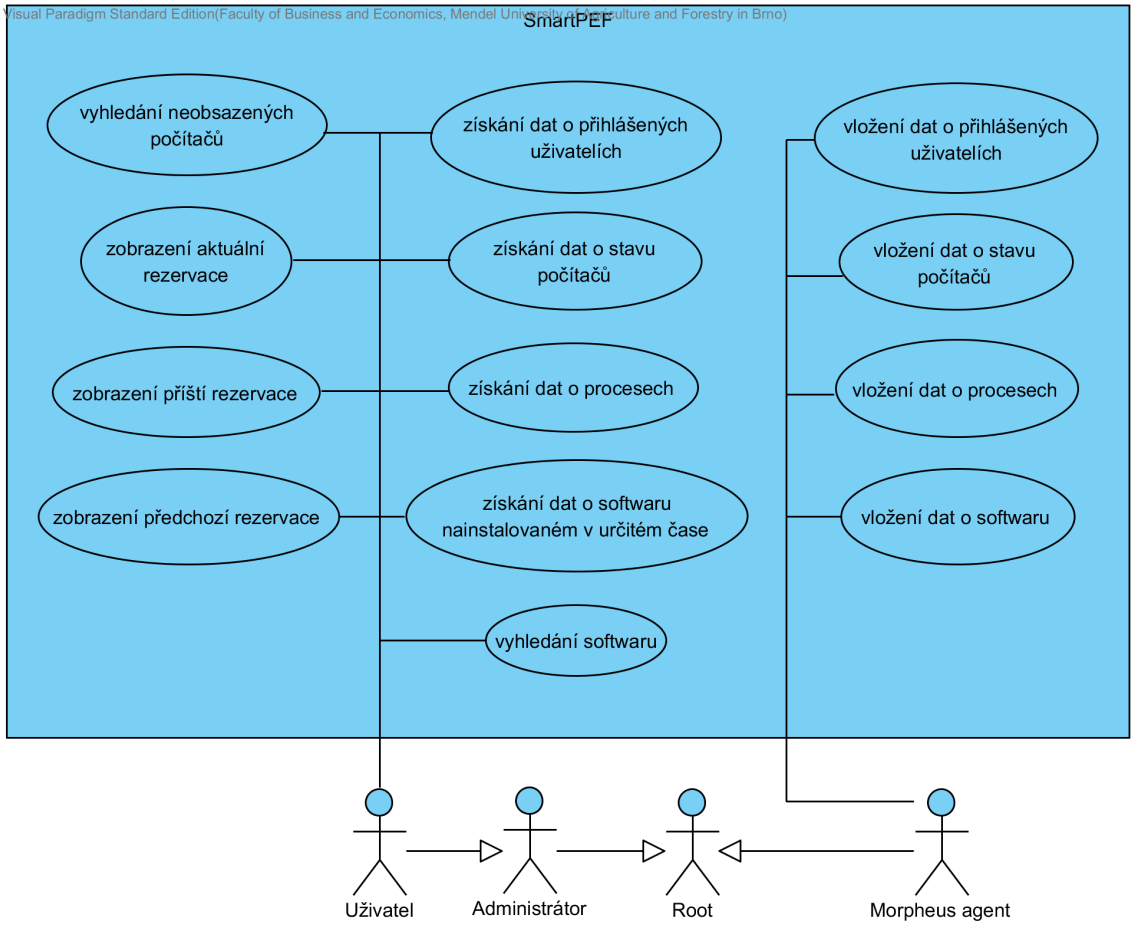
Modul MRBS bude poskytovat rozhraní ke čtyřem klíčovým funkcím, zmíněným v kapitole 4.1.1 - `getActualReservation`, `getPreviousReservation`, `getNextReservation` a `isPCFree`. Odpovědi budou poskytovány ve formátu JSON. Jak dále zmiňuje (Richardson, Amundsen a Ruby, 2017), je vhodné využívat existujících HTTP návratových kódů (např. 200 OK, 400 bad request, 500 internal server error).

4.2 Systém SmartPEF - modul Morpheus

Abychom mohli přistoupit k návrhu modulu Morpheus systému SmartPEF, musíme vědět, jak bude modul uživateli využíván. Pomocí analýzy funkčních požadavků a konzultací se zadavatelem byly sestaveny případy užití modulu, které jsou zachycené na obr. 9.

S modulem budou pracovat uživatelé pod čtyřmi různými rolemi. Jak bylo zmíněno v kapitole 2.4, je potřeba, aby byl přístup k datům, zejména těm citlivým, podmíněný autorizací. Citlivá data by měla být navíc před samotnou prezentací anonymizována. Každá z následujících rolí má proto přístup k rozdílným datům a rozdílným operacím nad daty. Na obr. 9 jsou tyto role znázorněny jako aktoři.

- **Uživatel** - tato role je definovaná pro přístup k veřejné části dat. Skrze tuto roli budou k datům přistupovat například aplikace pracující s databází nainstalo-



Obr. 9: Diagram případů užití modulu Morpheus systému SmartPEF

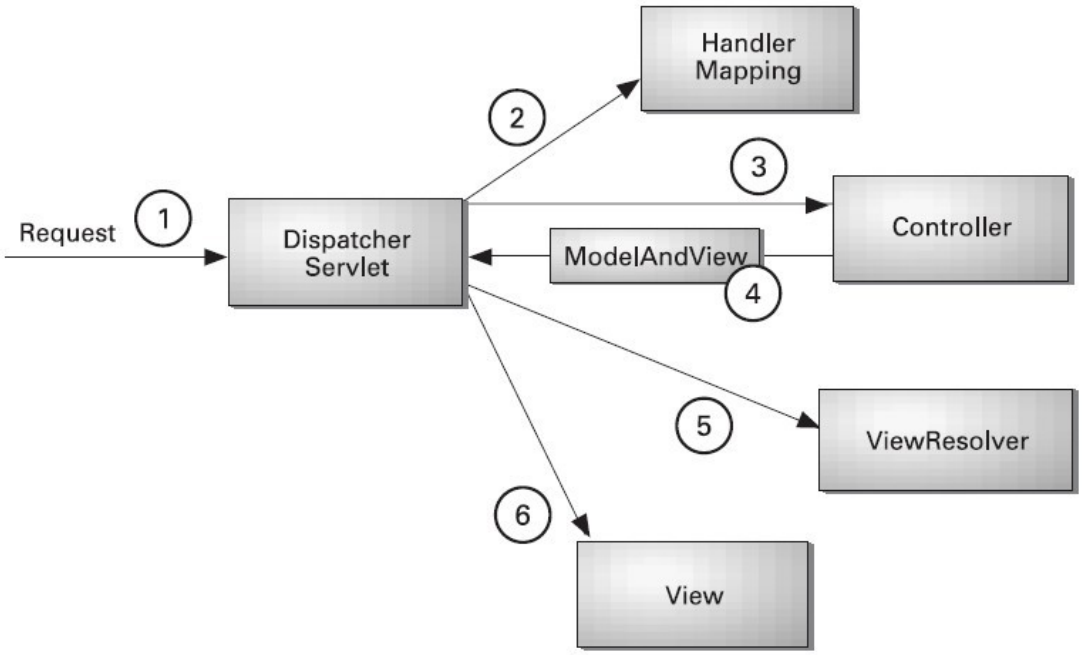
vaného softwaru. Všechna data ovšem před zveřejněním projdou anonymizační procedurou a uživatel tak neuvidí žádné údaje, které by se daly považovat za osobní či citlivé.

- **Administrátor** - má přístup pro čtení všech uložených dat. Na rozdíl od role Uživatel uvidí všechna data, včetně osobních údajů.
- **Root** - tato role má přístup ke všem datům, jak pro čtení, tak pro zápis. Stejně jako administrátor vidí všecha data neanonymizovaná.
- **Morpheus agent** - agent, který se stará o cyklické načítání dat do SmartPEF přes API. Má práva jen na zápis dat. Jeho funkce bude dále rozebrána v kapitole 4.3.3.

Konkrétní implementaci autorizace se věnuje kapitola 5.5. Řešení přístupu k jednotlivým zdrojům dat přes API rozebírá kapitola 4.2.3.

4.2.1 Architektura

Jak bylo zmíněno v kapitole 2.3, vývojáři všech modulů v rámci systému SmartPEF se snaží dodržovat stejnou architekturu napříč moduly. Jedná se o architekturu MVC, konkrétně realizovanou pomocí MVC frameworku Spring[8]. MVC (Model-View-Controller) architektura se skládá ze tří částí. Model zapouzdřuje aplikační data. View zodpovídá za vykreslování dat poskytnutých modelem. Většinou generuje HTML výstup, který uživatel může pomocí prohlížeče zobrazit. Controller zpracovává požadavky uživatelů, na jejichž základě získá data z příslušného modelu a tato data pomocí View vykreslí. Podíváme-li se na konkrétní implementaci architektury MVC frameworkem Spring (obr. 10), postup vyřízení HTTP požadavku vypadá následovně (Java Spring Framework, 2016).



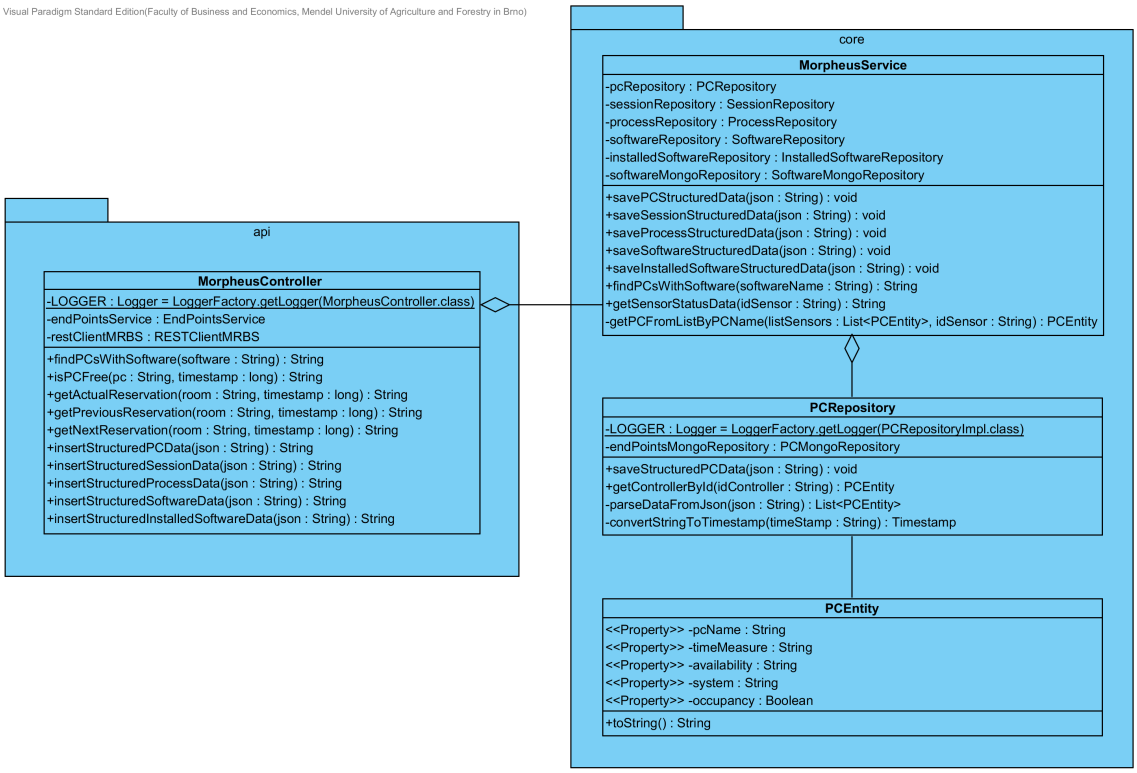
Obr. 10: Postup vyřízení HTTP požadavku MVC frameworkem Spring (Java Spring Framework, 2016)

Po obdržení HTTP požadavku (na obr. 10 označeno číslem 1) se `DispatcherServlet` na základě komponenty `HandlerMapping` rozhodne, jaký Controller bude využit pro vyřízení požadavku (2). Zvolený Controller na základě HTTP požadavku zavolá příslušnou obslužnou metodu. Metoda získá data z modelu a vrátí do `DispatcherServlet` název View, které se má postarat o vykreslení odpovědi (3 a 4). `DispatcherServlet` využije komponentu `ViewResolver` k tomu, aby zvolil vhodné View pro zadaný požadavek (5). Jakmile je vybráno správné View (6), `DispatcherServlet` předá data z modelu do View a to následně může být vykresleno prohlížečem.

[8]Spring je open-source framework určený pro vývoj J2EE aplikací.

Vzhledem k tomu, že využití dat ze SmartPEF závisí už na jednotlivých aplikacích pracujících s poskytnutými daty, nebude potřeba generovat HTML šablony pro prohlížeče. Namísto toho Controller poskytne HTTP odpověď s daty ve formátu JSON. Tato data si už každá jednotlivá externí aplikace zpracuje sama (např. je sama může použít ve své HTML šabloně nebo uložit do své databáze).

Architekturu si popíšeme na výřezu diagramu tříd (obr. 11). Aplikace se sestává ze dvou hlavních částí - `core` a `api`. `Api` zde zastává úlohu Controlleru (třída `MorpheusController` na diagramu 11), zpracovává totiž příchozí HTTP požadavky. `Api` využívá služeb, které přes rozhraní poskytuje `core` (konkrétně metody třídy `MorpheusService`). `Core` zde slouží jako Model a pracuje s databází (konkrétně s NoSQL databází MongoDB, jak bude rozebráno v následující kapitole). K databázi může `core` přistupovat pomocí libovolné třídy, která implementuje rozhraní `MongoRepository`, což je rozhraní pro MongoDB umožňující základní databázové CRUD operace (create, read, update, delete). Pokud z `api` přijde HTTP požadavek typu POST na uložení dat do databáze, instance třídy `MorpheusService` obdrží od `api` JSON řetězec představující dokumenty, které mají být vloženy do databáze. Pomocí nástroje Jackson se JSON řetězec namapuje na Java objekt, jehož struktura je popsaná ve třídě `PCEntity`, a přes `PCRepository` se vloží do databáze.



Obr. 11: Výřez z diagramu tříd modulu Morpheus systému SmartPEF

Může se zdát, že pro jednoduchou operaci vložení dat do databáze je návrh zbytečně složitý. Nicméně rozčlenění aplikace do několika vrstev umožňuje snazší

správu kódu. Pokud bychom se rozhodli vyměnit MongoDB za jiný databázový systém, do části `api` se vůbec nemusí zasahovat.

Návrhu části `api` se bude dále zabývat kapitola 4.2.3. Návrhu databáze bude věnována následující kapitola.

4.2.2 Návrh databáze

Po konzultaci se zadavatelem a s týmem zabývajícím se vývojem projektu SmartPEF byla pro ukládání dat vybrána databáze MongoDB.

MongoDB je multiplatformní dokumentová databáze. Řadí se mezi NoSQL[9] databáze a je vydávána jako open-source. Data ukládá jako dokumenty v binárním formátu BSON (Binary JSON[10]). Dokumenty s podobnou strukturou jsou obvykle organizovány do kolekcí. Kolekce by se daly přirovnat k tabulce v relační databázi. Dokumenty jsou podobné řádkům, atributy dokumentů zase sloupcům (MongoDB, Inc., 2017).

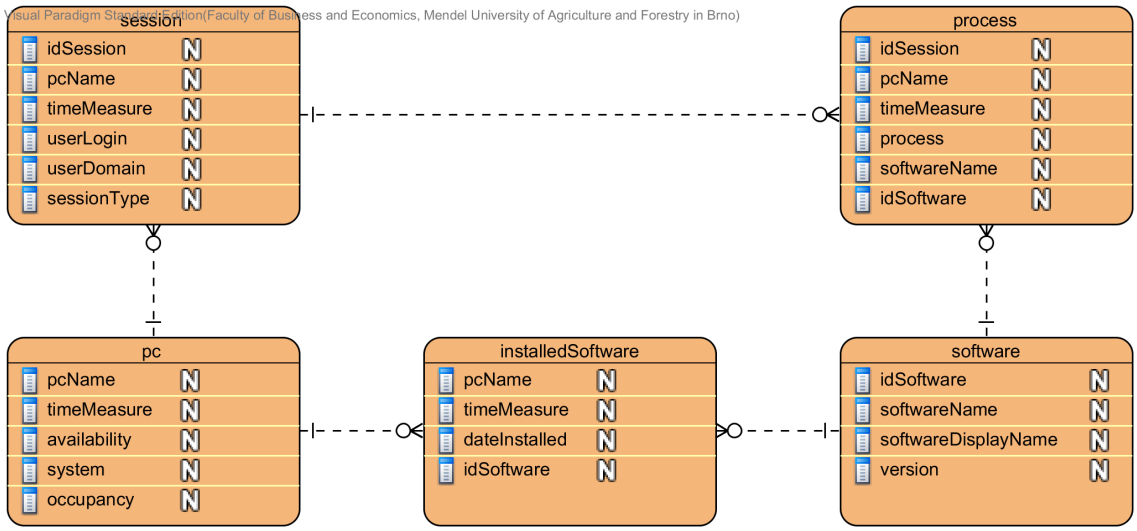
MongoDB kombinuje přístupy známé z relačních databází s NoSQL technologiemi. Z relačních databází si bere například možnost vytváření indexů nebo zajištění silné konzistence dat. Z NoSQL technologií zase dynamické databázové schéma založené na dokumentech. Na rozdíl od relačních databází, kde mají data svou pevnou strukturu, dokumenty umožňují mnohem snazší škálovatelnost aplikací využívajících MongoDB.

Ačkoliv MongoDB nevynucuje žádné pevné schéma, je vhodné vědět, s jakými daty bude modul Morpheus systému SmartPEF pracovat a na tato data připravit aplikační rozhraní komunukující s databází. Na úrovni aplikace tak můžeme ošetřit, aby se data do databáze dostávala v námi požadovaném formátu.

Požadovanou strukturu dat a vazby mezi daty zachycuje obr. 12. Je potřeba zmínit, že ačkoliv jsou na diagramu znázorněny vazby mezi entitami, MongoDB nemá, na rozdíl od relačních databází, žádný mechanismus, který by zajistil a vynutil referenční integritu. Stejně tomu je i se strukturou dat. Schéma není pevně definované. Do kterékoliv kolekce (`session`, `process`, `pc`, `installedSoftware` nebo `software`) je proto možné nahrát dokument v libovolném formátu. Můžeme například vynechat některé atributy či přidat atributy nové. Samotná databáze nám v tom nezabrání. Pokud tedy chceme korigovat strukturu ukládaných dat, musíme ji definovat už na straně aplikace.

[9]NoSQL (Not only SQL) je soubor konceptů, který umožňuje rychlé a efektivní zpracování dat se zaměřením na výkon, spolehlivost a agilitu. Jak zmiňuje (McCreary a Kelly, 2014), často se tento termín používá jako způsob, jak odlišit databázový systém od tradičních relačních databázových systémů (RDBMS).

[10]JSON (JavaScript Object Notation) je formát pro výměnu dat založený na jazyce JavaScript. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. JSON je založen na dvou strukturách - kolekce párů název-hodnota a seřazený seznam hodnot. V různých jazycích bývají struktury implementovány rozdílně. Například seřazený seznam hodnot může být realizován jako pole, vektor, seznam nebo posloupnost (Ecma International, 2013).



Obr. 12: Návrh databáze systému SmartPEF

Data, kterými se plní kolekce v databázi SmartPEFu, se berou ze systému Morpheus a jsou detailně popsána v kapitole 4.1.4. Za zmínku stojí některá specifika MongoDB. MongoDB poskytuje několik možností, jak realizovat vazbu 1:N. Rozebereme si zde tři - vnořený dokument, reference na potomka a reference na rodiče (Zola, 2014).

Tři různá řešení si uvedeme na příkladu entit `session-process` (viz obr. 12), tedy kdy v rámci jedné `session` (strana „1“ vazby 1:N) je uloženo více procesů (strana „N“ vazby 1:N). V případě vnořeného dokumentu by dokument v kolekci `session` vypadal následovně(viz atribut `processes`):

```
{
 idSession: 1542,
 pcName: 'rejpal02',
 timeMeasure: '2017-05-02 15:32:422',
 userLogin: 'xkral',
 userDomain: 'AD',
 sessionType: 'local',
 processes: [
    { process: 'firefox', idSoftware: 152648 },
    { process: 'OneDrive', idSoftware: 587512 }
 ]
}
```

Výhodou vnořeného dokumentu je fakt, že máme data v rámci jednoho dokumentu a nemusíme provádět žádné joiny na úrovni aplikace. Na druhou stranu, pokud s vnořenými entitami potřebujeme pracovat jako se samostatnými objekty, a nikoliv jen v rámci kontextu jejich rodiče, tento přístup se nám nevyplatí. Vno-

ření dokumentů se využívá v případech, kdy je kardinalita vazby 1:N nízká a kdy nepotřebujeme přistupovat k vnořeným dokumentům jako k samostatným entitám (Zola, 2014).

Druhou možností je využít referenci na potomka. Dokument potomka, procesu, by vypadal následovně:

```
{
 idProcess: 145,
 process: 'firefox',
 idSoftware: 152648
}
```

Dokument session s referencemi na tři procesy (atribut `processes`) by pak místo celých dokumentů s jednotlivými procesy obsahoval pouze reference na tyto procesy.

```
{
 idSession: 1542,
 pcName: 'rejpal02',
 timeMeasure: '2017-05-02 15:32:422',
 userLogin: 'xkral',
 userDomain: 'AD',
 sessionType: 'local',
 processes: [
    145,
    148,
    254
 ]
}
```

Tento přístup má opačnou sadu výhod a nevýhod než vnořování dokumentů. K potomkům lze snadno přistoupit jako k samostatným entitám. Na druhou stranu, pokud potřebujeme získat detaily o procesu, je nutné provést ještě druhý dotaz (join na úrovni aplikace). Další výhodou může být fakt, že jednotlivé procesy mohou být odkazovány z několika různých session. Z 1:N vazby se tedy stane vazba M:N, bez nutnosti existence jakékoliv vazební tabulky.

Třetí možností, jak řešit vazbu 1:N, je využít referenci na rodiče. V našem příkladu by dokument `session` vypadal následovně:

```
{
 idSession: 1542,
 pcName: 'rejpal02',
 timeMeasure: '2017-05-02 15:32:422',
 userLogin: 'xkral',
 userDomain: 'AD',
 sessionType: 'local'
}
```

V dokumentu proces by pak byla uvedena reference na rodiče (atribut `idSession`):

```
{
 idProcess: 145,
 idSession: 1542,
 process: 'firefox',
 idSoftware: 152648
}
```

Tento přístup se využívá v případech, kdy je potomků velké množství (v rámci MongoDB můžeme mluvit o milionech). Maximální velikost dokumentu je pro MongoDB 16 MB (Zola, 2014). Pokud by dokument `session` uchovával reference na miliony procesů, mohl by být tento limit překročen.

Pro kvalifikované rozhodnutí, které z výše zmíněných přístupů bude vhodné využít, potřebujeme vědět, jaká bude kardinalita vazeb mezi entitami. Kardinalitu vazeb zjistíme z analýzy objemu dat nahrávaných do databáze, která je nastíněna v kapitole 4.3.1. Dále potřebujeme vědět, jak budou data využívána. To vyplývá z funkčních požadavků v kapitole 2.4, kde je vytyčeno, jaké dotazy by měla nasbíraná data umožnit zodpovědět.

Podíváme-li se na požadované dotazy v kapitole 2.4, zjistíme, že ke všem entitám bude v určitých případech nutné přistupovat jako k samostatným objektům. Řešení vazby 1:N pomocí vnořených dokumentů je tedy nevyhovující. Proto budeme většinou využívat referenci na rodiče, v některých případech i referenci na potomka. Příklad reference na rodiče vidíme na obr. 12 v kolekci `process` - atribut `idSession`. Vzhledem k tomu, že řešení vazby 1:N pomocí referencí vyžaduje použití joinů na úrovni aplikace, bude v databázi potřeba vhodně nastavit indexy[11]. Indexy je vhodné nastavit na atributech, které jsou často prohledávány. Správné nastavení indexů může přispět ke zvýšení rychlosti provádění dotazů.

Dalším opatřením, kterým zvýšíme rychlost provádění dotazů na databázi, je denormalizace. V relačních databázích většinou dbáme na to, aby byla databáze normalizovaná a předcházelo se tak možné redundanci a ztrátě konzistence dat. V MongoDB jsou ale případy, kdy je výhodné databázi denormalizovat. Můžeme

[11]Databázový index je datová struktura, která slouží ke zrychlení vyhledávacích a dotazovacích operací v databázi.

tak zajistit rychlejší přístup k datům. Uveďme si příklad opět na dvojici `session-process`. Proces v našem případě obsahuje referenci na rodiče (atribut `idSession`).

```
{
 idProcess: 145,
 idSession: 1542,
 process: 'firefox',
 idSoftware: 152648
}
```

Pokud tedy chceme zjistit čas, ve který byl proces zaznamenán, musíme tento údaj (`timeMeasure`) vyhledat v dokumentu rodiče - v `session`.

```
{
 idSession: 1542,
 pcName: 'rejpal02',
 timeMeasure: '2017-05-02 15:32:422',
 userLogin: 'xkral',
 userDomain: 'AD',
 sessionType: 'local'
}
```

To vyžaduje join na úrovni aplikace. Namísto toho můžeme dát atribut `timeMeasure` z dokumentu `session` i do dokumentu `process` (denormalizovat ho). Dokument `process` pak bude vypadat následovně.

```
{
 idProcess: 145,
 idSession: 1542,
 process: 'firefox',
 idSoftware: 152648,
 timeMeasure: '2017-05-02 15:32:422'
}
```

Při dotázání se na čas zaznamenání procesu (`timeMeasure`) pak nepotřebujeme provádět žádný join na úrovni aplikace - tento údaj máme přímo v dokumentu `process`. Join bychom museli provést až v případě, kdy bychom potřebovali další, doplňující informace o `session`. Denormalizace má smysl pouze v případě, kdy jsou operace `read` nad danými atributy prováděny mnohem častěji než operace `update` (Zola, 2014). Denormalizace často čtených atributů sice zvýší rychlost čtení, ale komplikuje operace `update`. Pokud totiž potřebujeme aktualizovat atribut `timeMeasure` v takto denormalizované databázi, musíme provést `updaty` dva - jeden v kolekci `session`, druhý v kolekci `process`. V momentě mezi dvěma `updaty` se navíc bude databáze nacházet v nekonzistentním stavu.

Na základě analýzy funkčních požadavků a dotazů, které budou nad daty prováděny, byly proto určeny atributy, které budou čtené častěji než ostatní, a tyto byly denormalizovány. Na obr. 12 si uveďme jako příklad kolekci `process` - její atributy `pcName`, `timeMeasure` jsou převzané z kolekce `session`, podobně jako atribut `softwareName` je převzaný z kolekce `software`.

4.2.3 Návrh API

Při návrhu API modulu Morpheus systému SmartPEF budeme vycházet z diagramu případů užití (obr. 9). Pokud víme, jaká data budou uživatelé ze systému využívat, můžeme navrhnout API, které bude uživatelům skutečně sloužit.

V rámci kapitoly 4.1.6 byly popsány dva přístupy řešení API - SOAP a REST, včetně jejich výhod a úskalí. Na rozdíl od systému Morpheus, SmartPEF je koncipován jako otevřený systém, jehož data budou využívána různorodými aplikacemi, například pro zvýšení informovanosti studentů a zaměstnanců fakulty. Vzhledem k širokému spektru vývojářů, kteří budou využívat API systému SmartPEF, bude vhodné pro systém vybudovat REST API. Jak totiž bylo uvedeno výše, implementovat aplikaci využívající REST API (poskytnuté třetí stranou) je obvykle méně náročné než implementovat komunikaci mezi systémy pomocí protokolu SOAP.

Tabulka 1: Zdroje poskytované přes REST API modulu Morpheus systému SmartPEF

Metoda	**URL**	**Výsledek operace**
GET	mrbs/actualReservation	Vrátí aktuální rezervaci učebny v zadaném čase
GET	mrbs/freePC	Vrátí, zda je počítač volný v zadaném čase
GET	mrbs/nextReservation	Vrátí nejbližší rezervaci učebny po zadaném čase
GET	mrbs/previousReservation	Vrátí nejbližší rezervaci učebny před zadaným časem
POST	installedSoftware	Vloží do databáze data o nainstalovaném softwaru
POST	pc	Vloží do databáze data o stavu počítačů
POST	process	Vloží do databáze data o běžících procesech
POST	session	Vloží do databáze data o uživatelské aktivitě
POST	software	Vloží do databáze data o softwaru
GET	installedSoftware	Vrátí data o softwaru nainstalovaném v určitém čase
GET	pc	Vrátí data o stavu počítačů z určeného časového intervalu
GET	process	Vrátí data o procesech z určeného časového intervalu
GET	session	Vrátí data o uživ. aktivitě z určeného časového intervalu
GET	software/find	Vrátí počítače, na nichž je nainstalován hledaný software

Na základě diagramu případu užití (obr. 9) byly v první řadě definované zdroje, které bude REST API nabízet. Každý zdroj je dostupný pod specifickou URL. Následně byly definovány operace, které lze nad zdroji provádět. Všechny zdroje a operace, které nad zdroji lze provádět, nalezneme v tabulce 1. Uveďme si příklad zdroje `session` - pokud na URL `/session` zašleme HTTP GET požadavek s patřičnými parametry, REST API nám vrátí informace o uživatelích. Pokud ale na to stejné

URL zašleme HTTP POST požadavek, data o uživatelích poslaná v těle požadavku se vloží do databáze.

Postup řešení implementace REST API modulu Morpheus systému SmartPEF bude rozebrán v kapitole 5.4.

4.3 Komunikace mezi systémy

V kapitole 2 byl popsán současný stav systémů Morpheus, MRBS a SmartPEF. Bylo zjištěno, že tyto systémy spolu v současné době nijak nekomunikují. Jak vyplynulo z funkčních požadavků, je potřeba navrhnout vhodný způsob komunikace prostřednictvím API. Pro kvalitní návrh potřebujeme znát povahu dat, která si systémy mezi sebou budou vyměňovat. Z předchozích kapitol máme jasnou představu o tom, jaká data se nachází v systémech Morpheus a MRBS a jaká data je žádoucí uchovávat v systému SmartPEF. Nyní bychom měli analyzovat, jaký bude objem sbíraných a přenášených dat.

4.3.1 Analýza objemu dat

Jak bylo řečeno, do SmartPEF se budou posílat data z Morphea, aby zde byla archivována pro analýzu a další aplikace. Je třeba rozhodnout, jaká data má smysl na Morpheovi nechávat. Podíváme-li se na způsob plnění jednotlivých tabulek nové databáze Morphea, zjistíme, že objem dat bude výrazně narůstat pouze v tabulkách `session` a `running_processes`. Výpočet nárůstu záznamů v tabulce `session` je uveden v rovnicích 1, 2 a 3.

$$P_s = c \times u \times s \tag{1}$$

kde P_s je velikost přírůstku tabulky `session` [počet záznamů / den],
c počet počítačů ve správě Ústavu informatiky,
u průměrná doba využití počítače během dne v hodinách,
s počet spuštění skriptu na jednom počítači během jedné hodiny.

Dosadíme-li za proměnné konkrétní naměřené a odhadnuté hodnoty, získáme

$$P_s = 300 \times 4 \times 15 \tag{2}$$

a tedy

$$P_s = 18000 \tag{3}$$

Za jeden den tedy do tabulky `session` přibude 18 000 záznamů. Vezmeme-li v potaz fakt, že se tabulka `running_processes` plní zároveň se `session` a že v naprosté většině případů je k počítači přihlášen právě jeden uživatel (myšleno v době, kdy je k PC vůbec někdo přihlášen), stačí P_s vynásobit průměrným počtem procesů na jednoho uživatele a dostaneme P_{rp}, velikost přírustku tabulky `running_processes` [počet záznamů / den]. Podle experimentálního měření je průměrný počet procesů na jednoho uživatele 9, získáme tedy $P_{rp} = 162\ 000$.

Přírustky jsou uváděny v počtech záznamů za den, protože velikost v bytech záleží na použitém databázovém systému a na datových typech atributů.

4.3.2 Způsob výměny dat

Z předchozí kapitoly známe objem dat, která budou přenášena mezi systémy. Nyní je potřeba navrhnout, jak bude proces přenosu dat iniciován. V následujících odstavcích porovnáme dvě možnosti.

První možností je stav, kdy je proces přenosu dat iniciován jakoukoliv změnou v databázi systému Morpheus. Pokud se tedy změní stav libovolného počítače (např. je počítač vypnut nebo se na něj přihlásí nový uživatel) a Morpheus tuto změnu detekuje, tato informace se okamžitě přenese i do systému SmartPEF. Realizace tohoto přístupu by byla možná například pomocí triggerů na straně databáze systému Morpheus. Trigger je funkce, která se spustí vždy, když je v databázi vykonán určitý druh operace nad tabulkou, pohledem nebo schématem. Jak uvádí (Kaur a Shaik, 2016), mezi operace, které v PostgreSQL mohou vyvolat spuštění triggeru, patří INSERT, UPDATE a DELETE (DML[12] příkazy), CREATE, ALTER a DROP (DDL[13] příkazy) a SERVERERROR, LOGON, LOGOFF, STARTUP a SHUTDOWN (databázové operace). Trigger může být spuštěn buď před vykonáním nebo po vykonání operace. Trigger samotný nemůže zajistit nahrání dat na SmartPEF, ale tento nedostatek je možné odstranit spuštěním externího programu z triggeru[14], jak zmiňuje (Dar a kol., 2013). Výhodou je, že může jít o program napsaný v libovolném jazyce, nejsme tedy omezeni programovým vybavením PostgreSQL (databázový systém využívaný Morpheem).

Možností druhou je nahrávat data do systému SmartPEF cyklicky v pravidelných časových intervalech, po větších dávkách.

Nyní porovnáme obě možnosti. První možnost zajišťuje maximální možnou míru aktuálnosti dat v systému SmartPEF. Aktuálnost dat ve SmartPEF při zvolení druhého způsobu záleží na tom, jak často se budou data do SmartPEF nahrávat. Vzhledem k tomu, že SmartPEF slouží hlavně jako analytická platforma, nemusí nutně obsahovat naprosto aktuální data. Z tohoto ohledu je tedy i druhá možnost přijatelná. Rozhodující proto bude jiný faktor - výkon. Musíme si uvědomit, že trigger bude spuštěn při každé změně v databázi. Jak víme z kapitoly 4.1.1, informace o každém počítači se aktualizují co čtyři minuty. Aktualizace každého počítače může

[12]DML (Data Manipulation Language) je skupina syntaktických prvků podobná programovacímu jazyku, která se v databázi používá pro čtení, vkládání, mazání a editaci dat

[13]DDL (Data Definition Language) je skupina syntaktických prvků podobná programovacímu jazyku. Využívá se pro definování datových struktur, zejména databázových schémat.

[14]Pomocí tzv. neomezených jazyků (rozšíření PostgreSQL, např. PL/Pythonu) můžeme přistupovat nejen k databázi, ale i k externím službám, vytvářet a mazat soubory na databázovém serveru či posílat e-maily. Tyto jazyky jsou však označovány jako nedůvěryhodné, protože dovolují provádět operace, které omezené jazyky nepovolují. Fakt, že je jazyk nedůvěryhodný, značí sufix 'u' (untrusted) přidaný na konec názvu daného jazyku.

vyvolat až desítky databázových operací[15]. Počítačů je na fakultě přes tři sta, což není zanedbatelné číslo. Použití triggerů má tedy negativní dopad na výkon samotné databáze. Krom toho se jejich použití podepíše na výkonu samotného serveru Morpheus a také na zátěži univerzitní sítě, která má být dle nefunkčních požadavků pokud možno minimalizována. Pokud bychom totiž při každé aktualizaci databáze posílali data do SmartPEF, objem dat posílaných po síti v tisících malých požadavcích výrazně vzroste oproti stavu, kdy jednou za čas pošleme větší dávku dat. Pro každý přenos dat se musí vytvořit HTTP požadavek odeslaný na REST API systému SmartPEF. Na každý HTTP požadavek dostane Morpheus HTTP odpověď. Každý HTTP požadavek i odpověď obsahují hlavičky, které navyšují objem komunikace. Pokud bychom místo tisíce jednotlivých malých HTTP požadavků poslali jeden objemný, můžeme navíc velikost snížit i pomocí komprimace.

Z výše zmíněných důvodů bude proto vhodnější zvolit možnost druhou - cyklické nahrávání dat na SmartPEF v pravidelných intervalech. Realizace nahrávání dat do SmartPEF bude popsána v následující kapitole.

4.3.3 Agent pro nahrávání dat do SmartPEF

Prvkem propojujícím systémy Morpheus a SmartPEF je tzv. SmartPEF agent. Jak je vidět ze schématu na obr. 5, pomocí protokolu SOAP načte data z Morphea a následně je nahraje na SmartPEF přes REST API. Agent data do SmartPEF nahrává v pravidelných intervalech. Intervaly pro různé typy dat se liší, jak bude uvedeno dále.

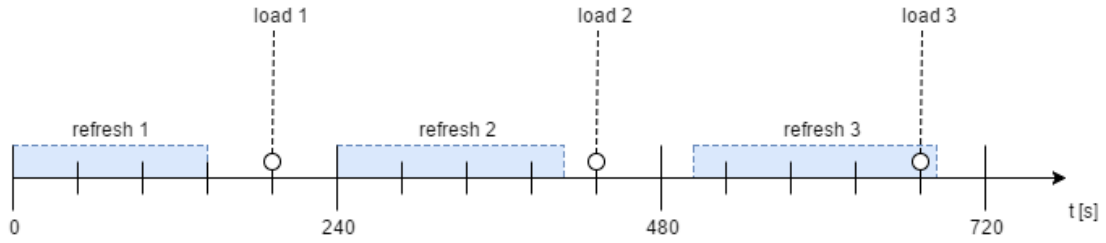
Stav počítačů

Prvním typem přenášených dat jsou data o stavu počítačů, tedy jaký systém je nabootovaný, jestli je počítač zapnutý a zda je na počítači někdo lokálně přihlášený (v tomto momentu nás nezajímá konkrétní uživatel ani uživatelé, kteří jsou přihlášení vzdáleně). Tato data se čerpají především z databázové tabulky `pc` systému Morpheus (viz obr. 6), která udržuje pouze aktuální stav počítačů. Proto je potřeba tato data nahrávat na SmartPEF ve stejném intervalu, v jakém Morpheus aktualizuje informace o stavu počítačů (jinak bychom přišli o data z některých časových intervalů). Délka intervalu je nastavena na čtyři minuty, jak víme z kapitoly 2.1. Musíme zajistit synchronizovanost obou procesů (aktualizace stavu počítačů pomocí modulu `Refresh` - dále v textu bude označeno jako proces **refresh** - a nahrání dat na SmartPEF pomocí agenta - dále v textu označeno jako **load**).

Ukažme si nyní, proč je synchronizace důležitá. Na obr. 13 je znázorněno řešení bez synchronizace. Refresh i load se provádí v pravidelných intervalech co čtyři minuty, load vždy mezi dvěma refreshi. Tento model funguje do chvíle, kdy refresh doběhne předtím, než začne load (na obrázku případy `refresh 1` a `refresh 2`). Jakmile se refresh opozdí, jak je tomu v případě `refresh 3` na obr. 13, zvýší se riziko

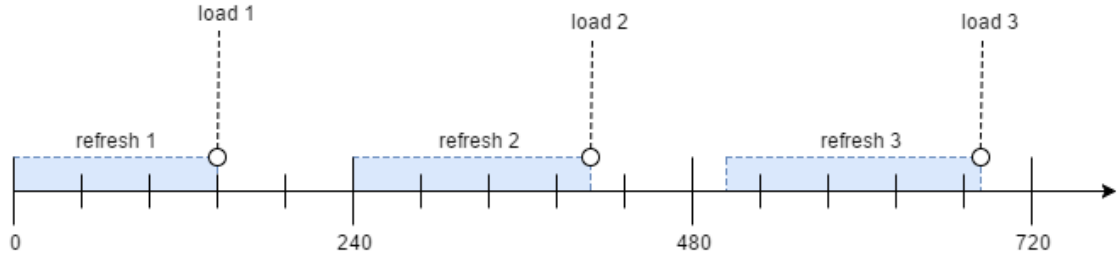
[15]UPDATE tabulky pc, INSERT do tabulek session a runningProcesses (až desítky spuštěných procesů).

ztráty části dat. Konrétně můžeme přijít o data z počítačů, které refresh nestihl zpracovat před loadem. Tato data totiž můžou být dalším refreshem přepsána.



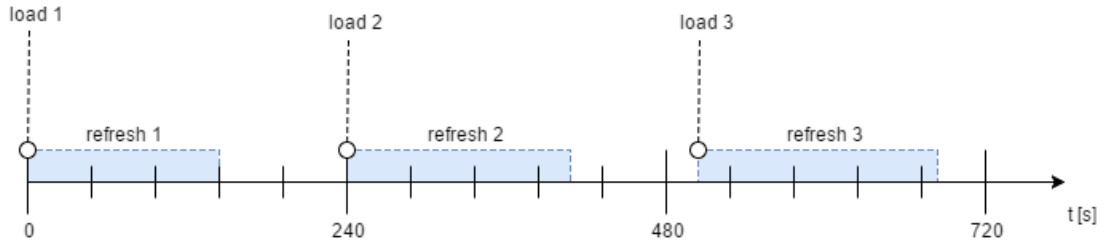
Obr. 13: Časový průběh procesů refresh a load: řešení bez synchronizace

Řešením by mohlo být ošetřit, aby po dobu, co proces refresh zapisuje do databáze, nemohl proces load z databáze číst. A četl by tedy vždy jen v intervalu mezi dvěma refreshi. Neberme nyní v potaz technické řešení, které by šlo realizovat např. pomocí databázových transakcí, explicitního zamykání tabulky nebo pomocí proměnných typu semafor (Downey, 2016). I pokud bychom implementovali toto ošetření, zůstává problém se synchronizací. Při scénáři na obr. 13 `load 3` nezíská přístup k databázi, protože do databáze právě zapisuje `refresh 3`. Musí si tedy najít jiný vhodný čas pro přístup. Vzhledem k tomu, že o sobě procesy load a refresh nejsou informovány, proces `load 3` nemůže odhadnout, kdy bude databáze znovu volná k využití. Proto je možné, že přijdeme o data z `refreshe 3`. Než totiž najdeme vhodný náhradní čas pro opakování procesu `load 3`, může se stát, že se začne zpracovávat další refresh a databázi znovu uzamče.



Obr. 14: Časový průběh procesů refresh a load: řešení se synchronizací, load po refreshi

Z výše uvedených důvodů bude tedy vhodnější procesy load a refresh synchronizovat. Nabízí se dvě možnosti. Provádět load bezprostředně před refreshem nebo bezprostředně po něm. První možnost je znázorněna na obr. 14, druhá na obr. 15. Zde narážíme na problém nastíněný v kapitole 2.1. Rozeberme si, jak funguje proces refresh. Jednou za čtyři minuty CRON pošle přes SSH příkaz na všechny počítače. Tyto příkazy zpracovává bash na serveru Morpheus, který jejich výsledky nahrává do databáze Morphea. Bash si však posloupnoust vykonávání příkazů řídí sám a CRON již nezjistí, v jaký moment se zpracoval poslední příkaz. Proto musíme využít řešení



Obr. 15: Časový průběh procesů refresh a load: řešení se synchronizací, load před refreshem

navrhnuté na obr. 15, tedy spouštět load vždy bezprostředně přes refreshem. Jedině takto zajistíme, že do SmartPEF nahrajeme vždy všechna data z předchozího refreshe.

Uživatelé a procesy

Druhým typem dat nahrávaných na SmartPEF jsou data o uživatelích a spuštěných procesech. Tato data se na SmartPEF budou nahrávat jednou denně, a to v době, kdy Morpheus není využívaný - tedy v noci. Vliv na zátěž serveru má proces nahrávání minimální, hlavní argument pro spouštění v nočních hodinách je proto jiný. Vždy po úspěšném dokončení loadu uživatelů a procesů se totiž promažou tabulky `session` a `running_processes`. V momentu, kdy se promažou, budou v Morpheovi chybět aktuální data např. o právě přihlášených uživatelích. Tyto informace potřebuje mimo jiné webový klient Morphea, který přes den využívají zaměstnanci fakulty.

Jednou za den se tedy na SmartPEF nahrávají data o uživatelích a procesech - konkrétně jaký uživatel byl v daný čas přihlášen, k jakému počítači, pod jakou doménou, a jestli byl přihlášen lokálně nebo vzdáleně. K tomu se přidají procesy, které byly v daném čase spuštěné pod jeho uživatelským účtem. V ideálním případě se procesy namapují na konkrétní nainstalovaný software. Postup mapování procesů na software bude rozebrán v kapitole 5.3.

Nainstalovaný software

Posledním typem nahrávaných dat jsou data o softwaru. Tato se čerpají z tabulek `software` a `installed_software`. Vzhledem k povaze dat stačí, aby byla na SmartPEF nahrávána jednou týdně. Nainstalovaný software se totiž v rámci fakulty příliš nemění a pokud ano, tak spíše v horizontu semestrů. Jak bylo zmíněno v kapitole 4.1, o promazávání tabulek se stará ten stejný skript, který získává data ze stanic. V tabulce `installed_software` je tedy vždy aktuální seznam nainstalovaného softwaru, nejedná se o historickou tabulku. Tabulka `software` se nepromazává vůbec, jde o číselník, do kterého se záznamy pouze přidávají.

5 Implementace

Tato kapitola se věnuje konkrétním krokům programování nejdůležitějších částí řešení navrhnutého v kapitole 4. Čtenář bude seznámen s problémy, které bylo nutno při implementaci řešit.

5.1 Identifikace procesů

Jak vyplynulo z analýzy funkčních požadavků a následného návrhu, je potřeba sbírat informace o procesech spuštěných na univerzitních počítačích pod aktivními uživatelskými účty. Operační systém Microsoft Windows poskytuje několik možností, jak se dopracovat k seznamu procesů. Tři z nich budou dále uvedeny a na základě následujících kritérií bude vybrána nejvhodnější varianta.

Je potřeba sledovat pouze procesy patřící pod konkrétní uživatelský účet a spuštěné přímo uživatelem, nikoliv procesy systémové (např. aktualizace systému nebo různé systémové služby). Systémové procesy nejsou z hlediska analýzy zajímavé. Další procesy, které nejsou pro analýzu důležité, by měly jít filtrovat pomocí blacklistu. Nutné je zjistit i vlastníka procesu, aby šly procesy spárovat s uživateli přihlášenými na daném počítači. V poslední řadě, jak vyplývá z nefunkčních požadavků v kapitole 2.4.2, je třeba, aby bylo možné příkaz pro zjištění běžících procesů spustit vzdáleně ze serveru Morphea a to bez nutnosti instalace dodatečného softwaru na všechny počítače na fakultě.

5.1.1 Tasklist

První možností, jak získat seznam procesů, je použít příkaz `tasklist`, který lze spustit pomocí příkazové řádky (program `cmd.exe`). Tento příkaz zobrazí všechny spuštěné procesy a služby. Příkaz může být vykonán jak lokálně, tak na vzdáleném stroji. Vrací textový výstup ve formátu `table`, `list` nebo `csv`.

5.1.2 Wmic

Druhou možností je použít program `wmic`, což je rozšíření WMI[16] umožňující používat služby WMI přes rozhraní příkazové řádky. Je to poměrně silný nástroj, jehož příkaz `wmic process` vrací množství detailů o spuštěných procesech (např. priorita procesu, čas vytvoření procesu nebo počet vláken). Výstup je opět textový a dá se převést do mnoha formátů, například `csv`, `xml` nebo `table`. K dispozici jsou také styly definované v `.xsl` souborech, pomocí kterých lze textový výstup převést na `html` (např. na `html` tabulku).

[16] Windows Management Instrumentation poskytuje skriptovacím jazykům rozhraní k operačnímu systému Microsoft Windows a umožňuje získávat data o řízení z lokálních i vzdálených počítačů. WMI je implementací technologií Web-Based Enterprise Management (WBEM) firmy Microsoft.

5.1.3 Get-Process

Možností třetí je využít PowerShell[17] a jeho cmdlet[18] `Get-Process`. Pracuje obdobně jako příkaz `tasklist` s tím rozdílem, že v rámci PowerShellu lze následně s výstupem cmdletu pracovat jako s objektem.

5.1.4 Výběr vhodné metody

Všechny tři příkazy je možné spustit vzdáleně ze serveru, rovněž jsou všechny tři okamžitě spustitelné na všech stanicích bez nutnosti instalace specializovaného softwaru. Všechny příkazy také poskytují potřebné informace o spuštěných procesech. Rozhodující byla tedy snadnost zpracování výstupu. Zde uspěl powershellový cmdlet, s jehož výstupem se pracuje elegantněji než s textovými výstupy prvních dvou metod. Odpadá nutnost složitě parsovat text, z kolekce objektů lze totiž snadno vybrat vhodné atributy, pomocí PID[19] spárovat procesy s vlastníky a takto předzpracovaná data poslat na server Morphea. Na server je tedy možné posílat jen potřebné informace, aby se zbytečně nezatěžovala univerzitní síť. Na základě posouzení výše zmíněných kritérií jsem tedy vybral třetí možnost - powershellový cmdlet `Get-Process`.

5.2 Identifikace nainstalovaného softwaru

Kromě spuštěných procesů je dle funkčních požadavků potřeba sbírat informace i o softwaru nainstalovaném na stanicích. Rozebereme si dvě možnosti, které nabízí nástroje OS Microsoft Windows. Rozhodující zde budou tři faktory. Kompletnost, tedy zda ve výsledku nebudou chybět data o některém softwaru. Dále rychlost provedení příkazu. A v poslední řadě fakt, že vykonání příkazu nesmí mít výrazný vliv na výkon dotazovaného počítače. Stejně jako v minulé kapitole platí, že příkaz musí jít spustit vzdáleně ze serveru, bez nutnosti instalace dodatečného softwaru na stanice.

5.2.1 Třída Win32Product

Win32Product je třída WMI reprezentující produkty, které byly nainstalovány pomocí nástroje Windows Installer. Obvykle se dá k produktu přiřadit instalační balíček jednoho programu. Třída má mnoho vlastností, které se hodí pro účely Morphea i SmartPEF - např. `Name`, `InstallDate` nebo `Version`. Avšak jak vyplývá z dokumentace Microsoftu (Microsoft, 2017), třída není optimalizovaná pro dotazování. Dotazy typu „`select * from Win32Product where (name like 'Mozilla%')`“

[17]Powershell je produkt firmy Microsoft. Jedná se o shell příkazového řádku a skriptovací jazyk založený na frameworku .NET (Kumaravel a kol., 2008).

[18]Cmdlet (čteno [kə'ma:ndlet]) je specializovaný příkaz, který se používá v prostředí Windows PowerShell. Cmdlet vykoná akci a většinou vrátí Microsoft .NET Framework objekt, který lze převzít dalším příkazem v rouře.

[19]Process identifier, jednoznačný identifikátor procesu.

vyžadují, aby WMI využila MSI[20] pro získání seznamu nainstalovaných produktů. Tento seznam je následně sekvenčně procházen a parsován, aby mohla být provedena `where` klauzule. Dotaz na třídu Win32Product také spustí kontrolu konzistence nainstalovaných balíčků s případným ověřením a opravením instalace.

5.2.2 Využití registrů

Druhou možností je využít registrů, konkrétně registru `SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`, ze kterého čerpá i seznam „Aktuálně nainstalované programy“ v Ovládacích panelech OS Microsoft Windows. V tomto registru se nachází každý produkt, který je kompatibilní s Windows a má program k odinstalování. Bude nutné zahrnout ještě druhý registr, a to `SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall`, který obsahuje 32bitové aplikace nainstalované na 64bitovém operačním systému. Z těchto registrů se dají vyčíst požadované hodnoty o konkrétních programech - např. atributy `DisplayName`, `InstallLocation`, `InstallDate`, `Version` nebo `UninstallString`. S registry je nutno pracovat opatrně, nesprávný zásah do registrů může způsobit jejich poškození a fatální důsledky pro činnost operačního systému.

5.2.3 Výběr vhodné metody

Co se týče kompletnosti výsledku, je spolehlivější využít registry. Obsahují totiž i programy, které nebyly nainstalovány pomocí Windows Installeru. Z testu rychlosti vychází registry také lépe, vzhledem k faktu, že třída Win32Product není optimalizovaná pro dotazování. Pro využití registrů hraje také třetí faktor - vliv na výkon stanice. Jak bylo zmíněno výše, dotazování na třídu Win32Product spouští kontrolu konzistence nainstalovaných balíčků a tudíž může mít vliv na systémové zdroje počítače. Na základě posouzení těchto kritérií jsem se rozhodl využít registry. Ani jedna metoda však nezachytí úplně všechen dostupný software - například programy, které můžeme spustit bez předchozí instalace.

5.3 Matching software-proces

V této fázi máme implementované dva powershellové skripty, které se přes ssh spouští na počítačích Ústavu informatiky a které získávají informace o spuštěných procesech, přihlášených uživatelích a nainstalovaném softwaru. Aby šlo odpovědět na otázky položené v kapitole 2.4, jako „Který software patří mezi nejčastěji využívané?“ nebo „Jak často byl na počítači X spouštěn program Y?“, musíme propojit procesy se softwarem. Tedy najít, který proces je spouštěn kterým softwarem.

[20] Microsoft Windows Installer, instalační služba firmy Microsoft pro instalaci a správu instalačních balíčků ve formátu MSI.

Porovnáme-li názvy procesů a názvy programů, zjistíme, že názvy procesů jsou často zkratkovité (z dvojic název procesu – název programu uveďme například `zps` – `Zoner Photo Studio 15`, `DTLite` – `DAEMON Tools Lite`, `skype` – `Skype 7.30`, `chrome` – `Google Chrome`). Proto byla vyloučena možnost spojování na základě regulárních výrazů nebo fuzzy matchingu[21]. Podíváme-li se na atributy, které lze vyčíst z registrů, najdeme mezi nimi dvojici `InstallLocation` a `UninstallString`, ze které jde ve většině případů zjistit umístění nainstalovaného programu. Proto byla implementována následující metoda spojování procesů a softwaru.

Algoritmus se pokusí zjistit umístění programu. Následně rekurzivně prohledá tuto složku včetně jejích podsložek a vrátí seznam všech spustitelných souborů s příponou `.exe`. Vzhledem k tomu, že procesy jsou z naprosté většiny pojmenované podle souboru, kterým byly spuštěny, lze namapovat `.exe` soubor přímo na název procesu. Tak zjitíme, ve které složce se nachází `.exe` soubor zodpovědný za spuštění procesu a tím pádem i jaký software k procesu můžeme přiřadit.

Řešení není dokonalé, mohou totiž nastat následující problémy. Jeden proces může být spuštěn různými programy a my nemáme možnost zjistit, jaký program je reálně jeho původcem. Toto se týká zejména případů, kdy je na počítači nainstalováno více verzí jednoho programu - např. proces `zps` mohl být spuštěn programem `Zoner Photo Studio 15`, stejně jako programem `Zoner Photo Studio 16`. V takovém případě se v databázi Morphea (viz obr. 6) atribut `id_software`, patřící danému procesu, namapuje na hodnotu `ambiguous` - „víceznačný“. Dále může nastat případ, kdy není nalezen žádný program, který by k danému procesu patřil. Například tehdy, není-li v registrech uveden atribut `InstallLocation`. Pokud nastane tato situace, atributu `id_software` je přiřazena hodnota `undefined` - „nedefinovaný“. Tyto nedostatky jsou sice aplikačně ošetřeny, ale i tak je na ně potřeba myslet při další práci s těmito daty.

5.4 SmartPEF REST API

Jak bylo zmíněno v kapitole 4.2.1, systém SmartPEF využívá Java MVC framework Spring. Spring 3.0 přináší silné nástroje pro tvorbu RESTful webových služeb. Framework využívá přístup založený na `HttpMessageConverter` a systému anotací. Webový klient a server spolu obvykle komunikují pomocí HTTP požadavků a odpovědí, tedy pomocí obyčejného textu. Nicméně metody Controlleru frameworku Spring vrací objekty třídy `String`. Tyto objekty je třeba serializovat na obyčejný text, o což se stará právě `HttpMessageConverter`. Kromě obyčejného textu můžeme Java objekty převést například do formátu JSON pomocí knihovny `Jackson`.

Abychom mohli vytvořit RESTful webovou službu, je v prvé řadě potřeba definovat třídy představující reprezentaci jednotlivých zdrojů. Pro každý zdroj definu-

[21] Spojování výrazů, které si jsou vzájemně podobné. Fuzzy matching se využívá například při automatizované kontrole překlepů. V našem případě by seznam programů sloužil jako referenční databáze, ve které by se vyhledával název programu podobný názvu procesu.

jeme třídu s atributy, konstruktorem a gettery pro přístup k atributům. Např. pro zdroj `pc` vytvoříme třídu `PcResource`:

```
public class PcResource {

    private String pcName;
    private String room;

    public PcResource(String pcName, String room) {
        this.pcName = pcName;
        this.room = room;
    }

    public String getPcName() {
        return pcName;
    }

    public String getRoom() {
        return room;
    }
}
```

Pokud chceme ke zdroji přistoupit pomocí HTTP požadavku, je potřeba vytvořit Controller. U třídy Controlleru uvedeme anotaci `@RestController`. V tradiční architektuře MVC Controller generuje HTML šablonu, která je následně vykreslena prohlížečem. Avšak anotací `@RestController` zajistíme, že namísto HTML šablony každá metoda Controlleru vrátí objekt, který bude automaticky převeden na JSON a takto zaslán v těle HTTP odpovědi. Instance třídy `PcResource` převedená na JSON pak může vypadat následovně:

```
{
    "pcName": "rejpal02",
    "room": "Q04"
}
```

Jakmile máme definované zdroje a Controller, můžeme pomocí anotací `@RequestMapping` u jednotlivých metod Controlleru zajistit mapování HTTP požadavků na konkrétní metody.

```
@RequestMapping(value = "/software/find",
    method = RequestMethod.GET)
@ResponseBody
public PcResource findPCsWithSoftware(
    @RequestParam("software") String software) {
        return endPointsService.findPCsWithSoftware(software); }
```

V případu uvedené metody anotace `@RequestMapping` zajistí, že se HTTP GET požadavek na adresu `/software/find` namapuje na metodu `findPCsWithSoftware`. Tato metoda vrátí instanci třídy `PcResource`, která je frameworkem automaticky převedena na JSON (framework by mohl instanci převést i na XML nebo obyčejný text, JSON byl frameworkem automaticky zvolen díky použití knihovny `Jackson`). Anotace `@RequestParam` dále zajistí namapování parametru `software` v URL HTTP požadavku[22] na parametr metody `findPCsWithSoftware`.

5.5 Autentizace a autorizace

Jak bylo zmíněno v předchozích kapitolách, ke všem systémům (SmartPEF, Morpheus, MRBS) je možné přistupovat pomocí veřejného API. V této kapitole se proto budeme zabývat především zabezpečení API.

Morpheus je webová aplikace a pro přenos SOAP zpráv využívá protokol HTTP. Stejným protokolem se svým okolím komunikuje i SmartPEF. Proto se v následujících řádcích budeme zabývat zajištěním autentizace a autorizace API právě pomocí HTTP. Rozebereme zde tři základní způsoby autentizace - pomocí HTTP Basic, OAuth 1.0 a OAuth 2.0.

5.5.1 HTTP Basic

Pokud uživatel na svůj HTTP požadavek dostane HTTP odpověď v následujícím tvaru, server vyžaduje autentizaci pomocí HTTP Basic.

```
401 Unauthorized HTTP/1.1
WWW-Authenticate: Basic realm="My API"
```

HTTP kód 401 sděluje uživateli, že se musí autorizovat. Hlavička `WWW-Authenticate` vysvětluje, jaký typ autorizace server akceptuje. V tomto případě server po uživateli vyžaduje využít autentizaci HTTP Basic. Jak uvádí (Richardson, Amundsen a Ruby, 2017), HTTP Basic autentizace je popsána v RFC 2617 a jedná se o jednoduchou autentizaci pomocí uživatelského jména a hesla. Žádný standard pro způsob, jakým uživatel toto uživatelského jméno a heslo získá, neexistuje. Většinou je obdržení přihlašovacích údajů realizováno pomocí registrace uživatele na webových stránkách. Nebo, jako v případě Morphea, jsou uživatelé zapsáni administrátorem napevno v konfiguračním souboru. Jakmile uživatel získá přihlašovací údaje, může je na API poslat v hlavičce `Authorization` HTTP požadavku.

```
POST /morpheus/dataendpoint/pc HTTP/1.1
Host: smartpef.mendelu.cz
Authorization: Basic dXNlcjpwYXNzd29yZA==
```

[22]URL požadavku, kde uživatel potřebuje najít software Chrome, by vypadalo následovně: /software/find?software=Chrome.

Pokud autentizace proběhne správně, uživatel získá ze serveru odpověď (v tomto případě se pokoušel nahrát na SmartPEF data o počítačích). Tento způsob autentizace má dva zásadní problémy. Zaprvé, není bezpečný. Řetězec `YWxpY2U6cGFzc3dvcmQ=` možná vypadá zašifrovaně, ve skutečnosti jde ale pouze o řetezec `user:password`[23] zakódovaný pomocí reverzibilní transformace Base64. Ze zakódovaného řetězce tedy lze jednoduchou transformací získat zpět původní uživatelské jméno a heslo. Kdokoliv, kdo odposlouchává komunikaci uživatele se serverem, se proto může heslo dozvědět.

Tento nedostatek lze odstranit využitím zabezpečeného protokolu HTTPS[24], který HTTP požadavky a odpovědi šifruje pomocí SSL vrstvy. Po síti se tak již nebudou posílat nezašifrované přihlašovací údaje.

Druhým problémem, dle (Richardson, Amundsen a Ruby, 2017), je fakt, že uživatel obecně nemůže věřit svým klientům, kteří zprostředkovávají komunikaci s API. Pokud tak například uživatel používá pět různých klientských aplikací pro přístup k API Twitteru, musí všem těmto aplikacím předat své přihlašovací údaje. Pokud se jedna z nich projeví jako nedůvěryhodná (např. začne posílat SPAM na uživatelův twitterový účet), musí si uživatel změnit heslo a změněné heslo sdělit zbylým čtyřem důvěryhodným klientským aplikacím. Pokud by měl uživatel pro každou klientskou aplikaci jinou sadu přihlašovacích údajů, tento problém by odpadnul. Proto si představíme druhý způsob autentizace, pomocí protokolu OAuth.

5.5.2 OAuth 1.0 a OAuth 2.0

Autentizaci pomocí OAuth 1.0 si uvedeme na příkladu z prostředí SmartPEF. Mějme hypotetickou klientskou webovou aplikaci SoftwareFinder.cz, která uživatelům umožnuje najít, na kterých počítačích na univerzitě je nainstalovaný požadovaný software. Dále máme server SmartPEF (smartpef.cz), který poskytuje REST API. Toto REST API je využíváno i aplikací SoftwareFinder.cz. Uživatel chce pomocí aplikace najít počítače, kde je nainstalovaný software AutoCad. Autentizace pak probíhá v následujících krocích:

1. SoftwareFinder.cz požádá SmartPEF o dočasné přístupové údaje.
2. SoftwareFinder.cz uživatele přesměruje na adresu smartpef.cz (pomocí HTTP redirect). Na této stránce se uživatel přihlásí pod svým účtem, který má registrovaný na smartpef.cz (tedy nikoliv pod účtem SoftwareFinder.cz).
3. Jakmile se uživatel přihlásí, smartpef.cz se uživatele zeptá, zda si přeje, aby aplikace SoftwareFinder.cz získala přístup k jeho uživatelskému účtu na smartpef.cz. Uživatel se rozhodne (např. na základě důvěryhodnosti klientské aplikace SoftwareFinder.cz).

[23] Jedná se pouze o smyšlené přihlašovací údaje.

[24] Hypertext Transfer Protocol Secure využívá protokol HTTP spolu s protokolem SSL nebo TLS, které zajišťují zabezpečenou komunikaci.

4. a) Pokud uživatel v kroku 3 aplikaci SoftwareFinder.cz přístup nepovolí, klientská aplikace žádný přístup nezíská.

   b) Pokud uživatel v kroku 3 přístup povolí, klient od smartpef.cz získá reálný přístupový token, který vymění za dočasné přístupové údaje získané v kroku 1. Aplikace SoftwareFinder.cz tak získá přístup k API smartpef.cz. Tento přístupový token je pak přikládán ke všem HTTP požadavkům na API smarpef.cz. Přístupový token může mít buď časově omezenou nebo neomezenou platnost.

Výhodou protokolu OAuth je fakt, že uživatel se autorizuje oproti smartpef.cz, nikoli oproti potenciálně nedůvěryhodnému klientovi. Pokud se tak jeden z klientů projeví jako nedůvěryhodný, poskytovatel API může tohoto klienta zablokovat (Patni, 2017). Navíc uživatel nezadává heslo přímo klientovi, ale pouze důvěryhodné aplikaci poskytující API (v našem případě smartpef.cz).

Výše zmíněný protokol OAuth 1.0 má jednu nevýhodu. Celá komunikace probíhá v rámci uživatelova internetového prohlížeče. Pro webové aplikace funguje tento přístup dobře, ale při využití desktopových nebo mobilních aplikací to může být problém (desktopová aplikace musí otevírat nové okno prohlížeče, aby získala uživatelovo svolení). OAuth 2.0 tento problém řeší pomocí definování čtyř různých způsobů, jak získat OAuth přístupový token. Tyto způsoby jsou definovány ve specifikaci RFC 6749.

5.5.3 Výběr vhodné metody

Předpokladem konceptu SmartPEF je, že jeho API bude využíváno množstvím[25] různorodých klientských aplikací, zejména webových a mobilních. Právě vzhledem k mobilním aplikacím bude vhodnější využít OAuth 2.0 místo starší verze OAuth 1.0. Autentizace pomocí HTTP Basic není vhodná, protože je možné, že vývoj všech klientských aplikací nebude čistě pod kontrolou Ústavu Informatiky a jejich kvalita a důvěryhodnost tak nebude kontrolována. SmartPEF tedy bude mít k dispozici svůj vlastní OAuth 2.0 autorizační server, vůči němuž se budou autorizovat všechny klientské aplikace. Tento server je zajištěn týmem vývojářů SmartPEF.

Situace systému Morpheus je jiná. API systému Morpheus nebude nabízeno programátorské veřejnosti v takové míře jako API SmartPEF. Bude využíváno především interně - pro komunikaci mezi webovým klientem Morphea a jeho serverem, mezi serverem a agentem, atp. Vývoj klientů využívající API je pod kontrolou a tudíž by neměli vznikat nedůvěryhodní klienti. Z tohoto důvodu si můžeme dovolit využít autentizaci pomocí HTTP Basic, ovšem s podmínkou, že komunikace bude probíhat zabezpečeně pod protokolem HTTPS.

[25]Toto množství zůstává zatím nespecifikováno. Konkrétní aplikace nemá smysl vytvářet dříve, než bude zajištěn sběr relevantních dat z prostředí fakulty.

5.5.4 Autorizace na úrovni metod

Jak bylo popsáno v kapitole 4.2, k REST API systému SmartPEF budou přistupovat uživatelé pod čtyřmi různými rolemi - user, agent, admin a root. Spring framework přináší možnost řešit autorizaci na úrovni jednotlivých metod pomocí anotací. Tyto anotace jsou celkem čtyři - `@PreAuthorize`, `@PreFilter`, `@PostAuthorize` and `@PostFilter`. Stěžejní je anotace `@PreAuthorize`, která rozhoduje, zda daná metoda může být pod aktuální rolí zavolána nebo ne.

```
@PreAuthorize("hasRole('AGENT') OR hasRole('ADMIN')")
public String insertProcessData(@RequestBody String json) {
    ...
}
```

Takto anotovaná metoda může být zavolána pouze uživatelem s rolí agent nebo admin. Výhodou je, že v anotaci můžeme využít i argument metody a na jeho základě provést autorizaci požadavku. Takto můžeme omezit přístup k některým zdrojům na základě argumentu metody. Příkladem může být metoda `getPC`, která vrací historické informace o zadaném počítači.

```
@PreAuthorize("hasPermission(#pc, 'ADMIN')")
public String getPC(@RequestParam("pc") String pc){
   ...
}
```

Pomocí anotace můžeme omezit přístup na úrovni jednotlivých počítačů (tedy rozlišit přístup pomocí argumentu metody).

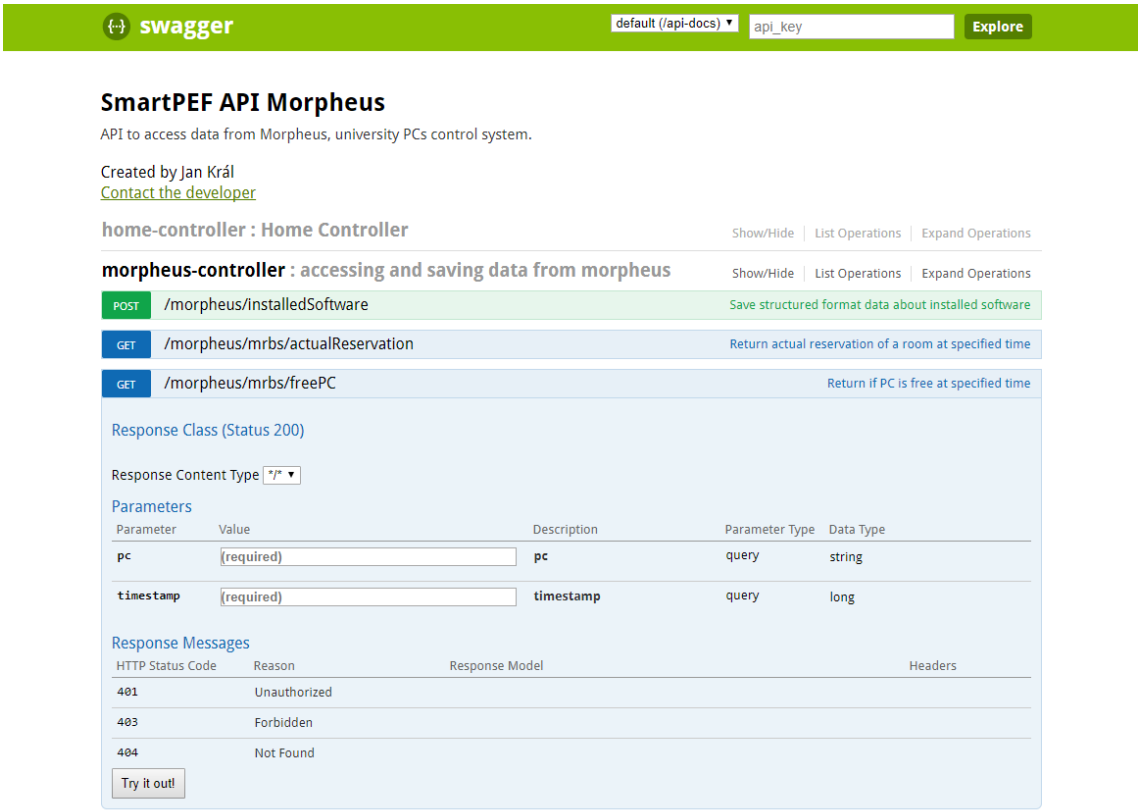
5.6 Tvorba dokumentace pomocí Swagger UI

Jak bylo zmíněno v kapitole 2.4, jedním z předpokladů snadné použitelnosti API je kvalitní dokumentace. REST API modulu Morpheus systému SmartPEF je zdokumentováno pomocí nástroje Swagger UI. Swagger UI umožňuje vizualizovat zdroje nabízené skrze API a umožňuje interakci uživatele s API (ať už je uživatelem vývojář daného API nebo koncový konzument API). Dokumentace se generuje automaticky na základě Swagger specifikace, bez nutnosti implementovat dodatečnou logiku (SmartBear Software, 2017). Swagger pracuje se systémem anotací, podobně jako Spring framework. Uveďme si příklad na metodě `isPCFree`.

```
@ApiOperation(value = "Return if PC is free at specified time")
@RequestMapping(value = "/mrbs/freePC",
    method = RequestMethod.GET
)
@ResponseBody
public String isPCFree(
```

```
        @RequestParam("pc") String pc,
        @RequestParam("timestamp") long timestamp) {
            ...
    }
```

Anotací `@ApiOperation` uvedenou u metody lze popsat danou operaci nad zdrojem `/mrbs/freePC`. Swagger pro tvorbu dokumentace využívá i anotace samotného frameworku Spring - například anotaci `@RequestMapping`. Ukázka UI vygenerovaného automaticky na základě anotací je zachycena na obr. 16. Za povšimnutí stojí zdroj `/morpheus/mrbs/freePC` uvedený v příkladu výše. Přes Swagger UI je možné vyzkoušet si práci s API napřímo - zadáním parametrů `pc` a `timestamp` a stisknutím tlačítka `Try it out` se zavolá obslužná metoda `isPCFree` namapovaná na `/morpheus/mrbs/freePC` a uživateli se zobrazí příslušný výsledek.



Obr. 16: Dokumentace API pomocí Swagger UI

Výhodou je, že celá dokumentace Swagger UI je generována dynamicky na základě kódu. Změní-li se tedy implementace API, změna se projeví i v dokumentaci.

6 Zhodnocení a závěr

Cílem této práce byla integrace nových datových zdrojů do systému SmartPEF, nově vznikajícího systému, který by měl sloužit studentům a zaměstnancům Provozně ekonomické fakulty Mendelovy Univerzity v Brně. Novými datovými zdroji jsou především systém MRBS pro správu rezervací učeben a systém Morpheus pro vzdálenou správu univerzitních počítačů.

V první části práce byla provedena analýza současného stavu. Byly analyzovány systémy, jejichž data budou využita v rámci SmartPEF - systém MRBS a systém Morpheus. Následně byl současný stav porovnán s požadavky na novou funkcionalitu. Na základě analýzy a funkčních požadavků byl proveden návrh rozšíření stávajících systémů a také návrh způsobu komunikace mezi systémy prostřednictvím API.

Jedním z výsledků práce je rozšíření systému Morpheus o dva moduly - modul zjišťující informace o uživatelské aktivitě na vzdálených počítačích (přihlášení uživatelé, běžící procesy, ...) a modul pro komunikaci s rezervačním systémem MRBS. Dále byl implementován agent pro automatizované nahrávání velkého objemu dat ze systému Morpheus do systému SmartPEF, kde se tato data ukládají historicky.

Dalším výsledkem práce je nově vytvořený modul Morpheus systému SmartPEF, který skrze své API poskytuje data z provozu počítačů na fakultě. Tato data jsou používána pro analýzu využívanosti nainstalovaného softwaru a vytíženosti počítačů v rámci dne, týdne či semestru. Informace získané z této analýzy mohou fakultě ušetřit náklady na licence k softwaru, který není využíván. Data mohou být skrze API také poskytnuta jiným aplikacím v rámci fakulty.

Navržené a implementované řešení poskytuje prostor pro další rozšiřování a vývoj. Může jím být například řešení otázky, jak optimálně (co nejrychleji a s co nejvyšší přesností) nalézt software patřící k danému procesu. Případně, jak bylo v práci zmíněno, vyměnit robustní SOAP API systému Morpheus za odlehčenější REST API. Dalším návrhem na rozšíření systému Morpheus je automatická detekce změny IP adres vzdálených počítačů. Pokud se totiž nyní změní IP adresa některého počítače, Morpheus o této skutečnosti zůstává neinformován.

V úvodu definovaný cíl práce byl naplněn, ovšem jak budou sbíraná data dále využita, záleží již na konkrétních aplikacích, které budou vyvíjeny v následujících letech. Jedním z možných využití může být aplikace umožňující studentům a zaměstnancům vyhledat, na kterých počítačích je nainstalován požadovaný software. Dalším příkladem využití dat může být aplikace, která umožní provést rezervaci na konkrétní počítač, nikoliv na celou učebnu. Možností využití nově sbíraných dat je však bezesporu více.

7 Literatura

Blum, R. *Linux command line and shell scripting bible*. Indianapolis: John Wiley & Sons, Inc., 2015. ISBN 978-1-84969-544-2.

Dar, U., Krosing, H., Mlodgenski, J., Roybal, K. *PostgreSQL Server Programming Second Edition*. Birmingham: Packt Publishing, 2013. ISBN 978-1-78398-058-1.

Downey, A. B. © 2016. *The Little Book of Semaphores* [online]. [cit. 2017-05-19]. Dostupné z: `http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf`.

DuVander, A. © 2010. New Job Requirement: Experience Building RESTful APIs. *ProgrammableWeb* [online]. [cit. 2017-05-19]. Dostupné z: `https://www.programmableweb.com/news/new-job-requirement-experience-building-restful-apis/2010/06/09`.

Ecma International. *Standard ECMA-404: The JSON Data Interchange Format*. In: Standard ECMA-404: The JSON Data Interchange Format [online]. 2013 [cit. 2017-05-19]. Dostupné z: `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`.

Java Spring Framework © 2016. Spring MVC Framework Overview [online]. [cit. 2017-05-19]. Dostupné z: `http://javaspringframework.com/spring-mvc.html`.

Kaur, M., Shaik, B. *PostgreSQL Development Essentials*. Birmingham: Packt Publishing, 2016. ISBN 978-1-78398-900-3.

Kumaravel, A., White, J., Li, M. N., Happell, S., Xie, G., Vutukuri, C. K. *Windows PowerShell Programming: Snap-ins, Cmdlets, Hosts, and Providers*. Indianapolis: Wiley Publishing, Inc., 2008. ISBN 978-0-470-17393-0.

McCreary, D., Kelly, A. *Making Sense of NoSQL*. Shelter Island: Manning Publications, 2014. ISBN 9781617291074.

Microsoft. © 2017. Win32Product class. *Microsoft Developer Network* [online]. [cit. 2017-05-19]. Dostupné z: `https://msdn.microsoft.com/en-us/library/aa394378(v=vs.85).aspx`.

MongoDB, Inc. © 2017. MongoDB Architecture. *MongoDB* [online]. [cit. 2017-05-19]. Dostupné z: `https://www.mongodb.com/mongodb-architecture`.

MRBS. 2017. MBRS: introduction [online]. [cit. 2017-05-19]. Dostupné z: `http://mrbs.sourceforge.net`.

Nayak, A. *MongoDB Cookbook*. Birmingham: Packt Publishing, 2013. ISBN 978-1-78216-194-3.

Orenstein, D. Application Programming Interface. *Computerworld* [online]. [cit. 2017-05-12]. Dostupné z: http://www.computerworld.com/article/2593623/app-development/application-programming-interface.html.

Patni, S. *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*. Santa Clara, California: Apress, 2017. ISBN 978-1-4842-2664-3.

Richardson, L., Amundsen, M. a Ruby, S. *RESTful Web APIs*. Sebastopol: O'Reilly Media, 2013. ISBN 978-1-4493-5806-8.

SmartBear Software. © 2017. Swagger UI. *Swagger* [online]. [cit. 2017-05-19]. Dostupné z: http://swagger.io/swagger-ui/.

Snell, J., Tidwell, D. a Kulchenko, P. *Programming Web services with SOAP*. Sebastopol, CA: O'Reilly & Associates, 2002. ISBN 0-596-00095-2.

SSH Communications Security. © 2017. SSH Home. *ssh.com* [online]. [cit. 2017-05-14]. Dostupné z: https://www.ssh.com.

TechTarget. © 1999 - 2017. What is agentless?. *WhatIs.com* [online]. [cit. 2017-05-19]. Dostupné z: http://whatis.techtarget.com/definition/agentless.

The IEEE and The Open Group. © 2001-2016. The Open Group Base Specifications Issue 7. *IEEE Std 1003.1-2008, 2016 Edition* [online]. [cit. 2017-05-19]. Dostupné z: http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xbd_chap04.html.

The PHP Group. © 2001 - 2017. Traits - Manual. *PHP: Hypertext Preprocessor* [online]. [cit. 2017-05-19]. Dostupné z: http://php.net/manual/en/language.oop5.traits.php.

Vyhnalík, J. *Systém pro hromadnou správu počítačů*. Bakalářská práce. Brno: PEF Mendelova univerzita v Brně, 2016.

Zola, W. © 2014. 6 Rules of Thumb for MongoDB Schema Design: Part 1. *MongoDB* [online]. [cit. 2017-05-19]. Dostupné z: https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1.

Přílohy

A Elektronická příloha

Přiložené CD obsahuje diplomovou práci v PDF a zdrojové kódy jednotlivých systémů. Dále obsahuje obrázky uvedené v práci, v jejich původním rozlišení.