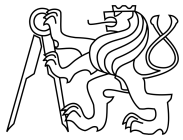
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Modulární webová aplikace pro správu serverů
Student:	Viktorie Novotná
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Web a multimédia
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Obsahem práce je návrh a implementace webové aplikace včetně samostatných modulů zajišťujících komunikaci s externími aplikacemi.

1. Navrhněte a naprogramujte základní webovou aplikaci, do které se budou připojovat následující moduly:
   Účetní - načítání finančních údajů o zákazníkovi (software Flexibee),
   Monitoring - načítání stavu virtuálních serverů zákazníků (software Icinga),
   Hypervizor - zajištění klonování, vypínání a zapínání virtuálních serverů (software libvirt).
2. Moduly naprogramujte, otestujte a zakomponujte do webové aplikace.
3. Implementujte uživatelsky přívětivé rozhraní.
4. Aplikaci jako celek otestujte, zdokumentujte a popište implementaci ve zvolené firmě.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 18. ledna 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Modulární webová aplikace pro správu serverů

Viktorie Novotná

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Petra Pavlíčková, Ph.D.

7. ledna 2018

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 7. ledna 2018 .....

Odkaz na tuto práci

Novotná, Viktorie. *Modulární webová aplikace pro správu serverů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce řeší problém evidence zákazníků a serverů v hostingové společnosti, přesněji roztříštěnost již existující evidence do mnoha nespolupracujících systémů. Tento problém práce řeší vytvořením nového informačního systému, který zobrazuje veškeré informace na jednom místě a jehož pomocí je možné i vytvářet nové serverové instance a jejich evidenci. Firmě a jejím zaměstnancům tento systém přináší zjednodušení běžných úkolů a přesnější evidenci. Pro externího čtenáře může být zajímavá samotná implementace. Systém například nepoužívá žádné vlastní úložiště a veškerá data asynchronně získává přes HTTP REST API.

Klíčová slova informační systém, integrace služeb, hosting, REST API, Ansible, PHP

Abstract

This work solves problem of evidence of customers and servers in a hosting company. More precisely it solves fragmentation of such evidence across many non-cooperating systems. This problem is solved by creation of new information system which shows all information in one place. It is even capable of creating new server instances along with proper evidence. For the company

and its employees the system brings simplification of routine operations and more precise evidence. For an external reader, implementation itself may be interesting. For example, the system does not use any storage, and gets all the data asynchronously over the HTTP REST API.

Keywords information system, services integration, hosting, REST API, Ansible, PHP

Obsah

Seznam obrázků

Úvod

Pracuji v malé firmě zabývající se převážně hostingem cizích i vlastních aplikací. Právě kvůli velikosti firmy bojujeme o každého zákazníka. V tomto boji je jedním z nejdůležitějších aspektů rychlost doručení ukázkové verze požadovaného programu. V nejlepším případě pak programu naplněného daty daného zákazníka. Ačkoliv se nejedná o příliš složitý úkol, jeho provedení vyžaduje spolupráci několika lidí. Obchodní zástupce přijme požadavek, předá jej oprávněnému správci serverů a ten po vytvoření vhodného virtuální stroje předá přihlašovací údaje programátorovi, který provede napojení dat. Celý tento proces může v nejhorších případech trvat i dny a po takové době je možné, že potencionální zákazník již používá konkurenční produkt.

Dalším problematickým místem je situace, kdy zákazník přestane za naše služby platit, ale jsou mu i nadále poskytovány. V současné době musí administrátor z účetního software zjistit, kteří zákazníci neplatí, najít server daného zákazníka a ten vypnout. Vzhledem k ručnímu vytváření takových strojů, nemusí být v konečném důsledku ani jasné, který server neplatící zákazník používá.

Samotná práce analyzuje skutečné potřeby společnosti, možnosti řešení objevených potřeb a představuje řešení v podobě nového informačního systému.

Kapitola 1

Cíl práce

Cílem práce je:

- analýza současných procesů ve společnosti týkajících se správy serverů,
- návrh aplikace umožňující objevené procesy zefektivnit,
- vytvoření takové aplikace,
- její nasazení v prostředí společnosti.

V konečném důsledku jde tedy o zvýšení efektivity práce za současného zlepšení evidence. Ušetřený čas poté umožní zaměstnancům soustředit se například na další rozvoj firmy.

Kapitola 2

Rešeršní část

V této kapitole se zabývám teoretickým popisem technologií souvisejících s problematikou této práce. V první části se věnuji definicím pojmů a průzkumu trhu se software pro poskytovatele hostingu. Druhá část se pak věnuje webovým aplikacím a jejich specifikům.

2.1 Technologie IT infrastruktury

2.1.1 Virtualizace

Jádrem virtualizace je virtuální stroj[1], což je dokonale izolovaný kontejner s vlastním operačním systémem a aplikacemi. Protože každý virtuální stroj je nezávislý, může jich běžet na jenom počítači hned několik[2]. Software zajišťující izolaci jednotlivých virtuálních strojů se nazývá hypervizor. Hypervizor se také stará o přidělování výpočetního výkonu jednotlivým virtuálním strojům podle potřeby [1].

2.1.2 Automatizace a orchestrace

Orchestrace znamená automatizaci určitých IT procesů[2]. Proces v tomto kontextu může být například vytvoření nového virtuálního stroje s určitou aplikací. Takový proces se skládá z mnoha úkolů jako instalace OS, databáze, webového serveru, které se neustále opakují. To sebou nese vyšší náklady a větší prostor pro chyby. Orchestrační nástroje[3] dokáží podle určité šablony postupně jednotlivé úkoly provádět bez lidského zásahu.

[1] Také nazývaný virtuální počítač či virtuální server.
[2] V závislosti na hardware teoreticky neomezeně, běžně jde o desítky až stovky.
[3] například Vagrant, Puppet, Ansible

2.1.3 Cloud Computing

Zjednodušeně cloud computing spočívá v poskytování výpočetní síly nebo přímo aplikací vzdáleně pomocí internetu [3]. Existuje velké množství variant, které se liší především tím, zda je nabízen celý virtuální stroj nebo jen určitá aplikace.

Základním stavebním kamenem pro poskytovale cloud computingu pak je virtualizace a orchestrace. Tyto technologie umožňují poskytovatelům efektivní využití jejich fyzických serverů a efektivní správu tisíců virtuálních strojů.

2.2 Existující řešení pro poskytovatele hostingu

Pro poskytovatele hostingových služeb existuje množství systémů. Tyto systémy obvykle spravují buď procesy související s objednávkami a platbami za hosting nebo naopak spravují pouze nastavení serverů a popřípadě hypervizoru.

2.2.1 WHMCS

WHMCS je software pro kompletní správu hostingové firmy. Obsahuje systém pro automatickou tvorbu faktur pro zákazníky a taktéž modul zákaznické podpory s ticket systémem[4]. Ve variantě pro 1000 klientů stojí licence přibližně 500 Kč měsíčně, což je ve firemním prostředí zanedbatelná částka.

Systém však neobsahuje modul pro vytváření nových virtuálních strojů a jejich správu. Absence této funkce poněkud kazí dojem ze slibované plné automatizace hostingové společnosti. Dalším aspektem systému je, že modul pro faktury se nezdá příliš flexibilní a jeho použití v českém právním prostředí by tedy mohlo být problematické.

2.2.2 Freeside

Freeside[4] je open-source systém pro správu hostingové firmy. Se software WHMCS sdílí prakticky všechny vlastnosti. Obsahuje však velmi zastarale působící uživatelské rozhraní. Na druhou stranu je zadarmo.

2.2.3 cPanel a WHM

WHM a cPanel jsou dva spolupracující systémy pro konfiguraci serverů. Obsahují webové rozhraní a průvodce pro velké množství běžných úloh, ať již se jedná o nastavení mail serveru, záloh nebo třeba webového serveru.

Pro poskytovatele hostingu však příliš funkcí nenabízí[5]. Systém je určený spíše pro koncové zákazníky, kteří si mohou svůj virtuální server nakonfigu-

[4] http://freeside.biz/freeside/

rovat v příjemné webovém prostředí. Odradit však může jeho cena, která je v přepočtu 400 Kč za měsíc a jednu IP.

2.2.4 VMmanager od ISP System

VMmanager je software pro správu virtualizace. Jeho možnosti pokrývají velké množství úkonů spojených se správou hypervizoru[6]. Software umí vytvářet nové instance virtuálních strojů, provádět zálohy a sledovat stav jednotlivých virtuálních strojů. Velmi se mi líbí například evidence použitých IP adres.

Kromě poslední zmíněné funkce a webového rozhraní však systém nepřináší nic navíc oproti open-source alternativám jako například virt-manager[5].

2.3 Webové aplikace

Webová aplikace je počítačový program založený na architektuře klient-server, jehož uživatelské rozhraní se obvykle zobrazuje ve webovém prohlížeči[7] a používá jazyk HTML, popřípadě CSS a JavaScript. Existuje mnoho způsobů implementací webových aplikací, nejdůležitější typy[8] představuji na následujících řádcích.

2.3.1 Aplikace generující HTML na straně serveru

V současné době nejpoužívanější způsob tvorby aplikací. Veškerý obsah webové stránky je naráz vygenerován na serveru a odeslán klientovi. Tento způsob má své nesporné výhody. Takovéto stránky se obrazí na téměr jakémkoliv klientském zařízení. Také vývoj těchto aplikací je snadný[8]. Nespornou nevýhodou takové aplikace je však její náročnost na přenosové kapacity. Při každém novém požadavku klienta se znovu přenáší celá stránka.

2.3.2 Aplikace používající komponenty

Jedná se o rozšíření předchozí architektury. V tomto případě stránka obsahuje komponenty, které se načítají pomocí technologie AJAX. Při změně části stránky se pak může načítat pouze změněná část. Tím se zmenší množství přenesených dat a zátěž serveru. Nevýhodou je však náročnější vývoj, kdy vývojář musí navíc ovládat technologie skriptování na straně klienta.

2.3.3 Aplikace generující HTML na straně klienta

Tento druh aplikací používá serverovou část převážně jen jako úložiště dat. Generování HTML z dat se pak provádí na straně klienta. Jedná se o moderní způsob tvorby webových aplikací a ze všech typů přenáší nejmenší množství

[5] https://virt-manager.org/

dat. Nevýhodou takových aplikací pak je posunutí logiky aplikace do prohlížeče klienta, kde je spouštěný kód ve zdrojové podobě viditelný a modifikovatelný. To může představovat bezpečnostní riziko.

2.4 Specifika vývoje webových aplikací

Vývoj webových aplikací je v určitých ohledem specifický oproti vývoji aplikací pro běžná PC. V následujích řádcích pak některá tato specifika krátce představím.

2.4.1 Protokol HTTP

Protokol HTTP pocházející z devatdesátých let minulého století je základem dnešního internetu. Zajímavostí tohoto protokolu z pohledu vývoje webových aplikací je to, že je bezestavový. To znamená, že webový server nemá informaci o jakékoliv vazbě mezi jednotlivými požadavky a to i když bezprostředně následují[9]. Tento fakt by mohl učinit HTTP prakticky nepoužitelným například pro implementaci nákupních košíků v elektronických obchodech. Naštěstí existuje technologie HTTP Cookie, která dokáže uložit malou část informace na straně klienta a s každým požadavkem tuto informaci odeslat serveru. Je však třeba dbát na fakt, že s cookies může být na straně klienta nakládáno libovolně a to včetně jejich pozměnění.

2.4.2 Architektura

2.4.2.1 Klient-server

Kvůli protokolu HTTP jsou i všechny webové aplikace založené na architekuře klient-server. Tedy komunikace vždy vychází od klienta v podobě požadavku a server na požadavek reaguje odpovědí[9]. Není možné aby server navázal komunikaci s klientem například pro odeslání notifikací. V současné době existuje několik možností jak toto omezení obejít. Příkladem můžou být technologie jako Server-Sent Events[10], Web Notifications[11] a Push API[12]. Jedná se však o technologie relativně nové a částečně si navzájem konkurující. To způsobuje, že jejich podpora není přítomna ve všech webových prohlížečích[6].

2.4.2.2 Oddělení prezentace a logiky

Oddělení uživatelského rozhraní od logiky aplikace je obecným standardem. Příkladem může technika MVC (Model View Controller), kde [13]:

- Model představuje cílový objekt se svými daty a aplikační logikou. Model nic neví o dalších vrstvách.

[6] Ověření současného stavu nabízí například web https://caniuse.com/.

- Pohled (View) je způsob reprezentace dat.
- Ovladač (Controller) pak zajišťuje načítání dat z Modelu a jejich předání do pohledu.

Ačkoliv se jedná o obecnou techniku, tak ve vývoji webových aplikací má zcela nezastupitelné místo. HTML (a CSS) umí definovat libovolné uživatelské rozhraní a dávají programátorům téměř neomezenou svobodu. To sebou ovšem nese nevýhodu poměrně rozsáhlého zápisu takového rozhraní.

2.4.3 Framework

Framework je kolekce znovupoužitelného software a systém práce s takovým kódem. Framework obvykle programátora nutí k určitému stylu práce[14]. To sice znamená určitá omezení, odměnou však je konzistentnější kód i při spolupráci více lidí. Další výhodou pak můžeme spatřit v rychlejším vývoji aplikace, protože programátoři se již nemusí zabývat detaily implementace. Často také framework pomáhá s bezpečností automatickým aplikováním některých kontrol na uživatelský vstup [15].

2.4.4 Bezpečnost

Prostředí webu a jeho aplikací také přináší některé specifické hrozby:

- Veškerá data jsou uložená na jednom[7] serveru, jeho kompromitování je lákavější než útok na jednotlivé uživatele.
- Komunikace klienta se serverem probíhá přes internet obvykle velkou vzdálenost. Hrozí zde riziko odposlechu komunikace.
- Samostatnou kapitolu pak tvoří bezpečné ukládání hesel.

Při vývoji webových aplikací si musí vývojář tyto hrozby uvědomovat a klást bezpečnost na první místo. Ačkoliv různé knihovny a frameworky mohou s bezpečností pomoci, musí vývojář při jejich výběru být opatrný a být si vědom, které hrozby může daná knihovna minimalizovat a které nikoliv.

Problém s možností odposlechu nebo dokonce pozměnění dat po cestě pak řeší šifrovaný protokol HTTPS, který se od běžného HTTP liší právě použitím šifrování.

Co se ukládání hesel na serveru webové aplikace týče, tak ta nesmějí být za žádnou cenu uložena v podobě umožňující jejich přečtení. K tomuto účelu se používají hašovací funkce[16]. Ty z libovolného vstupu vytvoří výstup konstantní délky a tato transformace je jednosměrná[8]. Pro zvýšení bezpečnosti se k heslu doporučuje ještě vygenerovat náhodný řetězec (tzv. sůl) a k heslu ho přidat.

[7] nebo jednotkách

[8] Existují ale seznamy (krátkých) hesel a jejich hashů, tzv Rainbow Tables.

Kapitola 3

Analýza a návrh

V této kapitole popisuji, jakým způsobem jsem se rozhodla splnit dané zadání a jeho případné rozšíření o požadavky firmy. Nejdříve však krátce představím společnost pro kterou práce vznikla a ve společnosti používané programy.

3.1 Prostředí společnosti

Práce vznikla pro potřeby společnosti Arit s.r.o. Jedná se o menší IT společnost zabývající se primárně správou serverů, stavbou sítí a prodejem hardware a software. Z pohledu software hraje nejvýznamnější roli účetní program Flexibee. Součástí firmy je také programátorské oddělení ve kterém vznikají nadstavby nad prodávaným software (primárně Flexibee) nebo samostatné programy, vždy však s nějakou vazbou na Flexibee. Tyto nadstavby počínají programy provádějícími náročnější datové transformace přes eshopovou platformu a končí u samostatně fungujícího pokladního systému. Většina těchto programů je pak také provozována přímo na serverech společnosti.

Těchto serverů společnost provozuje řádově stovky a to v různých lokalitách. Je tedy potřeba vést detailní evidenci, tak aby bylo jasné, kde se který server nachází, jaká je úloha daného serveru a pro kterého zákazníka je server provozován. Tento seznam byl historicky veden jako jeden dokument tabulkového procesoru na sdíleném disku. Toto řešení však postupem času přestalo stačit a byl zaveden program iTop, kterému se věnuji v následující kapitole.

Obecným problémem dalších systémů bývá jejich přidaná komplexnost. V případě programu iTop se místo otevření tabulky na sdíleném disku, musí technik vytvářející nový server přihlásit a vyplnit veškeré údaje, kterých je více než v původní tabulce. Může se tak snadno stát, že osoba vytvářející nový server zapíše jen některé údaje, popřípadě nevyplní údaje žádné. Nejedná se však pouze o program iTop. Nový server je třeba zaregistrovat do monitorovacího software. V případě nutnosti také zajistit doménové jméno a vytvořit směrovací pravidlo na routeru. V neposlední řadě je nutné informo-

vat zákazníka, že server pro něj právě vznikl. Jedná se tedy o nezanedbatelné administrativní zatížení.

Z nedokonalé evidence pak vzniká celá řada problémů, například v situacích, kdy je potřeba provést odstávku některého systému nebo v případě stížností zákazníka na nefunkčnost. V takové situaci není možné se zákazníka ptát, co za služby si u firmy pořídil, jelikož to působí velmi neprofesionálně. Navíc už samotný telefonát zákazníka může značit problém na naší straně; díky monitoringu bychom měli o problémech vědět ideálně dříve než zákazník.

3.2 V současnosti využívané programy

V této podkapitole popisuji ve firmě aktuálně používané systémy. Každý systém také krátce představím a zhodnotím z hlediska využitelnosti pro splnění zadání. Společnou a velmi nepříjemnou vlastností vyjmenovaných systémů (kromě programu FlexiServis) je jejich nulová provázanost.

3.2.1 Flexibee

Jedná se o účetní systém pro malé a střední podniky, který mezi ostatními vyniká možností provozu na všech PC platformách, množstvím funkcí a neposlední řadě velmi kvalitním REST API[17]. Kromě základního účetnictví však program zvládá i například skladové hospodářství, výrobu, mzdy zaměstnanců, evidenci smluv atp. Ve své podstatě je tedy možné o Flexibee mluvit jako o jednoduchém ERP systému.

Společnost Arit je distributorem programu Flexibee a tedy má rozsáhlé zkušenosti s jeho využitím v různých nasazeních. Za další důležitou výhodu programu Flexibee lze také považovat rozsáhlou dokumentaci a to jak pro běžné uživatele, tak hlavně pro programátory pracující s Flexibee API.

Nevýhodu programu Flexibee, zvláště s ohledem na využití jako informačního systému, vidím v jeho neměnitelné databázové struktuře. Není žádným způsobem možné rozšířit data ukládaná do programu o další políčka. Tento fakt je možné částečně překonat použitím nepotřebných políček, popřípadě uložením dat ve formě přílohy [9]. Obě tato řešení však sdílejí stejný problém, že takto uložená mohou editovat i běžní uživatelé. Toto komplikuje využití Flexibee jako úložiště dat, protože nadstavbové aplikace musí počítat s tím, že data mohla být kdykoliv poškozena. Ukládání dat jako příloh pak navíc trpí nízkým výkonem, který je dán způsobem uložení v databázi.

[9] Příloha v programu Flexibee je speciální druh evidence, do které je možné ukládat libovolné soubory.

3.2.2 iTop

Program iTop je webový open-core[10] portál pro evidenci hardware, software a IT procesů. V naší firmě nahradil již dříve zmíněnou tabulku se seznamem serverů. Z celé škály možností systému se momentálně používá hlavně databáze virtuálních serverů, jejich vazby na hostitelské servery a také vazby mezi serverem a zákazníkem.

Možnosti programu iTop však jdou mnohem dále. Za zmínku určitě stojí evidence výpadků, evidence stížností (ticket) nebo například evidence SLA smluv. Tyto vlastnosti již však nejsou z pohledu této práce úplně relevantní. Jedná se však o zajímavou oblast dalšího rozvoje. Za důležité ovšem považuji, že tyto dodatečné vlastnosti nemohou plně fungovat bez dokonalé evidence.

3.2.3 Icinga

K monitorování běhu serverů firma momentálně využívá dva systémy. Přesněji řečeno probíhá migrace ze systému Nagios na systém Icinga. Oba systémy jsou určené k automatické kontrole jednotlivých funkcí monitorovaných počítačů. Ať už se jedná o spuštění systému, zaplnění disku nebo stav webového serveru.

Není bez zajímavosti, že systém Icinga vznikl původně jako fork systému Nagios. Současné verze těchto systémů jsou však již značně odlišné a přechod od jednoho programu k druhému není bez komplikací.

Systém Icinga se skládá z mnoha jednotlivých komponent [18]. Za nejdůležitější považuji:

- `Core` obsahuje základní jádro systému, které tvoří REST API, plánovač kontrol, systém rozesílání upozornění.
- `Web` obsahuje prezenční webovou vrstvu. Využívá jazyka PHP a pro samotné rozhraní jazyka Javascript a technologie AJAX. Ačkoliv se přímo nabízí využít pro `Web` komponentu REST API z komponenty `Core`, Icinga tohoto nevyužívá a `Web` komponenta používá API odlišné.
- `IDODB` komponenta zajišťuje ukládání do různých relačních SQL databází.
- Další nezbytnou součástí jsou samotné monitorovací skripty, které však Icinga neobsahuje a místo toho využívá skripty z projektu Nagios.

3.2.4 FlexiServis

FlexiServis je ve firmě vzniklý systém pro správu výkazů práce. Zároveň je přes něj možné práci zadávat ve formě servisních požadavků vázaných na zákazníky. Jedná se tedy ve své podstatě o ticket systém. Jeho výhodou je, že

[10] Program s otevřeným zdrojovým kódem, pro který vlastník vydává placené doplňky

pracuje pouze nad daty z Flexibee, tedy nemá žádné vlastní úložiště. Tento fakt výrazně usnadňuje následnou fakturaci vykonané práce, protože nejsou potřeba žádné synchronizace. Data se do účetnictví ukládají již ve vhodném formátu.

Drobnou nevýhodou programu je, že nemá žádné API, přes které by bylo možné data číst. Na druhou stranu, je možné provádět stejná čtení jako FlexiServis přímo z Flexibee.

3.2.5 Ansible

Ansible je mocný nástroj primárně určený pro automatizaci nasazení software (deployment). Jeho použití je poměrně snadné, ale zkušení uživatelé dokáží pomocí Ansible zvládnout i velmi komplexní úlohy[19]. Mezi další jeho výhody patří fakt, že na cílových strojích nepotřebuje žádnou podporu, pouze jazyk Python, který je však na většině Linuxových strojů již nainstalovaný.

Ve firmě Ansible pomalu zavádíme do praxe a já osobně v něm vidím velký potenciál v usnadnění stále se opakujících činností. Vzhledem k implementaci Ansible jako programu pouze příkazovou řádku je jeho bezpečná integrace do dalších systémů složitější. Existuje sice interní API, které je možné volat z jazyka Python, tvůrci Ansible však použití tohoto API nedoporučují, protože jeho změna může nastat kdykoliv[20]. Tento problém řeší nadstavby jako například Semaphore nebo Ansible Tower, která obsahuje rozsáhlé REST API.

3.3 Možnosti řešení

V této kapitole popisuji jednotlivé možnosti řešení. Postupně probírám jednotlivé možnosti řešení a analyzuji jejich výhody a nevýhody. Beru v potaz celkem tři metody: naprogramovat potřebné moduly do některého ze stávajících systémů, naprogramovat velké množství malých synchronizačních programů nebo naprogramovat zcela samostatný systém. V úvahu neberu programy z kapitoly 2.2. Primárně z důvodu, že společnost již vlastní a má nasazené obdobné programy. Zavádění dalších programů proto považuji za zbytečné a navíc mimo rozsah této práce.

3.3.1 Integrace do současných systémů

Jako první možnost jsem zvážila integrování potřebných funkcí a modulů plynoucích ze zadání do některého ze stávajících systémů.

3.3.1.1 Flexibee

Program Flexibee umožňuje několik způsobů integrace do jiných systémů:[21]

- Odkazování do Flexibee, tedy vytvoření odkazu na určitou evidenci do webového rozhraní programu Flexibee.

- Přímé čtení REST API a s tím související podporu nahrávání a zíkávání dat ve formátu Extensible markup language (XML).
- Integrace Graphical user interface (GUI) jedná se o poměrně zajímavou funkcionalitu, avšak není téměr dokumentovaná.

Všechny předchozí způsoby však umožňují pouze integraci systému Flexibee do jiných programů a ne obrácený postup. Tedy integraci nových funkcionalit do Flexibee.

Integraci nových vlastností do Flexibee pak určitým způsobem umožňuje nástoj `uživatelské tlačítko`[22]. Tento nástroj umožňuje do rozhraní systému Flexibee přidat tlačítko, které v integrovaném nebo externím webovém prohlížeči otevře předem zvolenou adresu. Bohužel i integroaný prohlížeč otevírá zcela nové okno programu a tak pocit z integrace není nejlepší a vždy musí existovat externí systém, na který bude odkazováno.

3.3.1.2 Icinga Web

Webová část systému Icinga podporuje rozšíření pomocí vlastních modulů. Její rozšíření je tedy snadné. Dokonce existuje i hotový modul[11] modul pro import dat ze systému iTop do systému Icinga. Modul pro komunikaci s hypervizorem by byl nepochybně také možný. Vzhledem ke skutečnosti, že by nebylo nutné psát zcela nový program, se mohou výsledky takovéto práce dostavit velmi brzy.

Tento způsob řešení však skrývá několik problémů. Celý informační systém by byl závislý na stabilitě a podpoře od třetí strany. Také další rozvoj systému by tímto utrpěl. Jednak by se ztížila údržba, protože by bylo nutné kontrolovat kompatibilitu modulů s novými vydáními software Icinga. Taktéž se počítá s dalším rozvojem plánovaného informačního systému o další moduly, které nemají s monitorigem a servery nic společného a jejich integrace do systému na monitoring by nedávala smysl. Navíc zde pak vzniká riziko plynoucí z omezení API pro moduly systému Icinga.

3.3.1.3 iTop

I tento program umožňuje integraci uživatelských rozšíření. Z principu však trpí stejným omezením jako předchozí návrh.

Kvůli výše zmíněným omezením jsem se tedy rozhodla cestu integrace do cizích systémů nevyužít.

3.3.2 Křížová synchronizace

Druhou možností je vytvoření velkého množství malých aplikací spojující každé dva systémy. S tímto druhem řešení mám již zkušenosti. Jedná se ob-

[11] https://github.com/Super-Visions/icingaweb2-module-itop

vykle o poměrně jednoduché aplikace překládající volání API na jiný formát. Tento způsob řešení je sympatický tím, že je jednoduchý a každá aplikace dělá pouze jeden druh činnosti, což je žádoucí [KISS]. Problém však představuje absence centrálního bodu, který poskytuje všechny dostupné informace. Hlavně z důvodu, že každý systém spravuje jinou část informace a synchronizovaly by se pouze duplicitní části dat.

Další nevýhodou je komplikovaná správa a dozor nad funkčností jednotlivých programů. V neposlední řadě se jedná o neudržitelný systém z pohledu budoucího vývoje. S každým novým systémem by rostl počet potřebných služeb.

Tento způsob řešení jsem tedy také zavrhla.

3.3.3 Samostatný informační systém

Tvorba vlastního informačního systému má taktéž své nevýhody:

- časová náročnost: Je třeba naprogramovat nejenom komunikaci s okolním světem, ale i samotný základ aplikace. Tedy řešit vlastní uživatelské rozhraní, autentifikaci a provázání modulů mezi sebou. Vzniká tak větší množství kódu, který již někdo jiný jinde napsal a otestoval.
- Vzniká n+1 systém, což není optimální [23].
- Oproti distribuovaným systémům, znamená chyba v centralizovaném programu kompletní výpadek.

Tyto nevýhody je však možné výrazně zmírnit. S časovou náročností a potencionálním duplikováním kódu pomůže vhodný framework. Větší dopady chyb v programu lze ošetřit vhodnou architekturou, kontrolováním chyb a důkladným testováním.

Tento přístup má ale i své nesporné výhody:

- úplná volnost ve využití technologií a způsobů práce,
- nezávislost na třetích stranách,
- z předchozího plynoucí nižší náklady na údržbu a jednodušší budoucí rozvoj.

Mé konečné rozhodnutí tedy padlo na vytvoření nového samostatného informačního systému, který bude sám modulární, tak aby maximalizoval znovuvyužitelnost kódu a zjednodušoval budoucí rozvoj.

3.4 Uživatelské příběhy - případy užití

V této podkapitole stručně popisuji úkony, se kterými informační systém pomůže a také pojednám o uživatelích, kteří budou systém používat.

3.4.1 Vytvoření serveru pro zákazníka

Proces vytvoření serveru pro zákazníka obvykle začíná přijetím žádosti od zákazníka obchodním oddělením, které od zákazníka získá potřebné údaje. Především pak, jaký software si na novém serveru přeje provozovat. Pokud se jedná o nového zákazníka obchodní oddělení o zákazníkovi vytvoří záznam v systému Flexibee a předá požadavek na vytvoření serveru kompetentní osobě. Tento postup informační systém nepoužívá, protože zatím není implementováno zakládání nových zákazníků mimo účetní program Flexibee.

Informační systém použije až technik, který vyplní formulář pro vytvoření nového serveru. Důležitou součástí tohoto formuláře jsou údaje o síťovém připojení a hypervizoru, na kterém nový server vznikne. Po vytvoření pak technik dle požadavku může spustit instalaci dodatečného software. Tento krok předpokládá vytvoření a zaregistrování těchto skriptů v software spravujícím Ansible. Tedy například Semaphore nebo Ansible Tower.

3.4.2 Průběžné sledování stavu

Každý server se po vytvoření zaregistruje do monitoringu Icinga, který pak sleduje jeho zdravotní stav. V případě problémů Icinga rozesílá emailem upozornění. Informační systém technikům pomáhá v tom, že přímo na úvodní stránce zobrazuje seznam řešených a neřešených problémů. Počítá se vystavením tohoto seznamu na velkém monitoru v kanceláři techniků.

3.4.3 Zjištění stavu serveru při stížnosti

I nejlepší monitoring může někdy selhat a zákazník se na firmu obrátí se stížností. V tuto chvíli je potřeba co nejrychleji najít všechny údaje o zákazníkovi. Informační systém svým uživatelům proto poskytuje vyhledávání jak v zákaznících firmy, tak případně v seznamu serverů. Po nalezení zákazníka je pak na jedné stránce vidět výpis jeho serverů, jejich stavu a platební morálka. V případě, že informaci od zákazníka přebírá obchodní oddělení, může na základě informací ze systému technikovi předat mnohem detailnější informace. V případě že informaci přebírá technik, může ten zákazníkovi sdělit o problému více informací například i s odhadem znovuzprovoznění. Ale například při značné neuhrazené sumě může technik nejprve kontaktovat obchodní oddělení, kvůli určení priority řešení takového problému.

3.4.4 Spuštění skriptu na serveru

Čas od času je na serverech nutné spustit některé Ansible skripty. Typickým skriptem může být aktualizace operačního systému nebo nasazení nové aplikace. V tomto případě stačí technikovi zvolit požadovaný skript, v dalším kroku server a nakonec vyplnit skriptem požadované parametry. Toto předpokládá evidování daného Ansible skriptu v systému pro správu Ansible.

3.5 Funkční požadavky

3.5.1 Vytvoření serveru

Tento požadavek stál u zrodu celého informačního systému. Systém musí umět vytvořit nový virtuální server a zaregistrovat jej do všech požadovaných služeb. Účastníkem tohoto procesu je servisní technik a postup vypadá takto:

1. Technik se přihlásí svými údaji do systému a zvolí v menu *Nový Server*.
2. Systém zobrazí formulář pro vyplnění potřebných údajů o serveru, jedná se hlavně o zákazníka, IP adresu a hypervizor.
3. Při vyplňování formuláře technikem systém některé údaje předvyplní.
4. Technik stistkne tlačítko vytvořit.
5. Systém data roztřídí a uloží přes příslušná API.
6. Systém spustí skript na tvorbu serveru.
7. Systém přesměruje technika na výstup skriptu.

Možné výjimky:

- **Nedostupný datový zdroj:** Uživatel je o problému informován a není mu umožněno vytvořit server.
- **Selhání ukládání:** Uživatel je informován a server není uložen.

Předpokladem scénáře je pak vedení zákazníka v učetním software, záznam o hypervizoru v dokumentaci (iTop) a správná konfigurace Ansible Tower.

3.5.2 Zobrazení seznamu serverů

Systém musí umět zobrazit seznam spravovaných serverů.

3.5.3 Zobrazení detailu serveru

Systém musí umět zobrazit podrobnosti o vybraném serveru. Tedy zobrazit všechna dostupná data včetně vazby na zákazníka a hostitelský fyzický server.

3.5.4 Zobrazení informací o zákazníkovi

Systém musí umět zobrazit seznam zákazníků, sumu faktur po splatnosti daného zákazníka a vazbu mezi zákazníkem a serverem.

3.5.5 Zobrazení stavu serverů

Systém musí umět zobrazit servery, u kterých se objeví technické problémy, tedy data z monitoringu. Dále musí umět u daného problému uložit tzv. `acknowledgement`. Tedy zprávu o tom, že se problém řeší. Účastníkem tohoto procesu je servisní technik a postup vypadá takto:

1. Technik se přihlásí svými údaji do systému.
2. Systém zobrazí seznam řešených a seznam neřešených problémů.
3. Technik kliknutím přejde do detailního výpisu neřešeného problému.
4. Systém zobrazí údaje z monitoringu a zobrazí formulář pro zadání poznámky.
5. Technik formulář uloží.
6. Systém odešle data do monitoringu, problém se označí jako v řešení.

Pokud je problém již ve stavu *řešený*, systém místo formuláře zobrazí poznámku, kterou byl problém označen. V případě nedostupnosti datového zdroje s monitoringem je uživatel informován a není mu umožně scénář provést.

3.6 Souhrn využitých technologií - Nefunkční požadavky

V této podkapitole se věnuji důležitým technologiím, které jsem v práci použila. Při výběru jsem zohledňovala mnoho faktorů a to jak vyloženě technických, tak i týkajících se dlouhodobé udržitelnosti, možností dalšího rozvoje a v neposlední řadě jsem dbala na to, aby práce zapadla do ekosystému firmy.

3.6.1 Typ aplikace a programovací jazyk

Zvolila jsem klasickou webovou aplikaci s menším využitím skriptování na straně klienta. Tedy webovou aplikaci druhého, tak jak je popsána v podkapitole 2.3.2.

Jako hlavní programovací jazyk jsem zvolila PHP, které je i přes svoji špatnou pověst nejrozšířenějším jazykem na webu [24]. Tedy existuje potenciálně velké množství lidí, kteří mohou aplikaci dále rozvíjet. Dále se jedná i o hlavní a téměř jediný jazyk firmy. Navíc poslední verze PHP (řada 7.x) přidávají možnost vynucení datového typu parametru u funkcí a metod. Další možnosti silnější typového kontroly pak přidává direktiva `strict_types`, která pro parametry funkce vypne automatickou konverzi jejich datových typů [25].

3.6.2 Framework Nette

Pro usnadnění vývoje a údržby jsem dále použila framework Nette a to v zatím nevydané verzi 3, která plně využívá nových vlastností PHP 7.1. Další výhodou tohoto frameworku je jeho rozšířenost a tedy množství dokumentace a návodů. Různé komponenty Nette frameworku jsou taktéž hojně využívány v ostatním software firmy.

3.6.3 Bootstrap

Pro návrh uživatelského rozhraní byl použit CSS framework Bootstrap. Mezi jeho hlavní přednosti patří filozofie `mobile-first`, tedy vytvoření uživatelského rozhraní nejdříve pro mobilní zařízení a až pozdější optimalizace pro zařízení s větší obrazovkou. Nutno poznamenat, že tyto úpravy nemusejí být vždy nezbytné a tak vzniká univerzální rozhraní pro všechny druhy zařízení naráz.

Další nespornou výhodu vidím v jednotnosti uživatelských rozhraní navržených pomocí frameworku Bootstrap. Uživatelé tak při seznamování s novou aplikací mohou využít své předchozí znalosti z jiného rozhraní napsaného touto technologií, prvky uživatelského rozhraní budou totiž stejné nebo alespoň velmi podobné.

3.7 Architektura

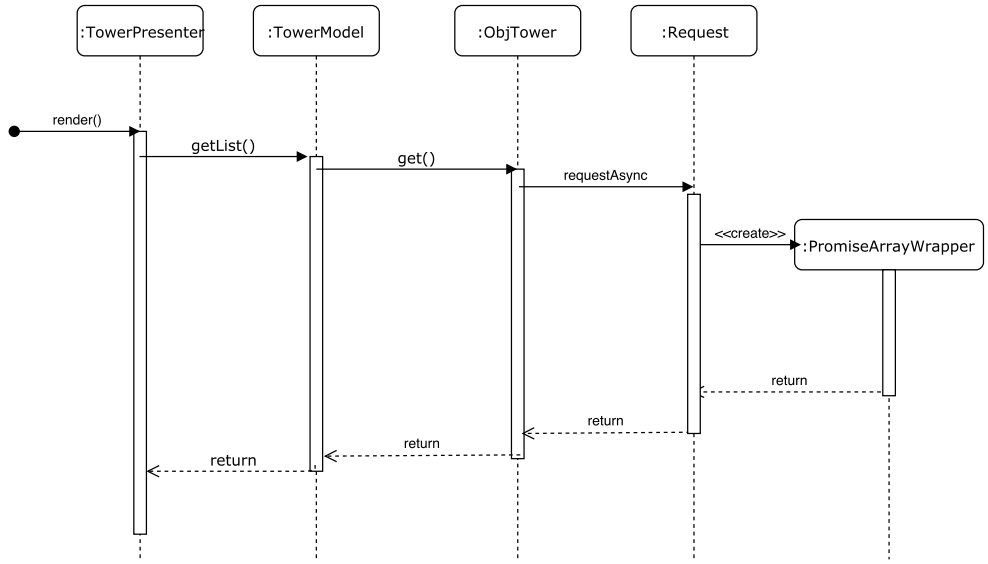
Po oddělení dat a uživatelského rozhraní používám Nette frameworkem doporučenou strukturu MVC (viz také kapitola 2.4.2.2): „Model-View-Controller je softwarová architektura, která vznikla z potřeby oddělit u aplikací s grafickým rozhraním kód obsluhy (controller) od kódu aplikační logiky (model) a od kódu zobrazujícího data (view).“[26]

3.7.1 Jmenné konvence

Pro snazší orientaci ve zdrojovém kódu jsou dodržovány konvence v pojmenování a zařazení do jmenných prostorů. Při jejich stanovování jsem se vzala v potaz standardy navrhované skupinou PHP-FIG[27], standardy definované Nette frameworkem[28] a také názory hlavního vývojáře frameworku Nette Daviga Grudla[29]. V některých případech uvedené standardy uvádějí protichůdná doporučení, často se však jedná spíše o nepodstatné odchylky jako například používání mezer místo tabulátorů [30].

Výsledné konvence shrnuje následující seznam:

- Jména presenterů a komponent končí na `Presenter` respektive `Control` a jsou umístěny ve jmeném prostoru `Presenter` respektive `Control`.



Obrázek 3.1: Schéma datového toku při požadavku na API Ansible Tower

- Jména modelových tříd končí na `Model` a jsou umístěny ve jmeném prostoru `Model`.
- Modely v datových modulech pak navíc začínají na jméno systému, který využívají a evidenci, se kterou pracují například `FlexibeeCustomerModel`.
- V modulech se také nacházejí třídy začínající na `Obj`, které mají za úkol práci s instancemi třídy `Request`.

V modulech je také nutné dodržovat standard PSR-4, který určuje vztah mezi jmenným prostorem a adresářovou strukturou a to z důvodu, že o načítání modulů se stará program Composer. V hlavním modulu automatické načítání tříd (tzv. autoloading) zajišťuje Nette Robot Loader, který umí pracovat s libovolnou adresářovou strukturou.

3.7.2 Datové toky

Veškeré načítání dat z externích systémů se také řídí jednotnými pravidly. Každý požadavek postupně projde přes čtyři třídy informačního systému:

1. Některý `Presenter`, ze kterého framework vyvolá metodu [12].
2. Třída se jménem končícím na `Model`. Obvyklá jména metod jsou například `getList` nebo `getDetail`.

[12] Obvykle metodu `action` nebo `handle` v závislosti na typu požadavku[26].

3. Třída začínající na `Obj` s metodami například `get` a `post`. Tato třída má za úkol vytvořit URL adresu pro získání požadovaných dat.

4. Třída `Request`, která vytváří samotný požadavek a monitoruje jeho průběh.

Jedno konkrétní načítání dat z REST API je možné vidět na obrázku 3.1.

3.8 Bezpečnost

Informační systém má přístup i pro zápis do mnoha systémů důležitých pro běh firmy a to včetně přístupu k fyzickým serverům v roli hostitele. Jeho zabezpečení musí být tedy nejvyšší možné.

3.8.1 Ochrana proti útokům

S minimalizováním řady bezpečnostních hrozeb pomáhá samotný Nette framework [15]. Dále informační systém nevyužívá žádnou vlastní databázi, čímž odpadají útoky typu SQL injection. Komunikace s externími úložišti je vždy přes šifrovaný protokol. K přihlašování uživatelů byl využit server LDAP a tedy odpadá možnost úniku hesel přímo ze systému. Momentálně je server s nasazeným informačním systémem pouze ve vnitřní síti společnosti. Tato opatření významně snižují prostor k vnějšímu útoku. Nesnižují však neúmyslné škody způsobené programátorskou či uživatelskou chybou.

3.8.2 Ochrana proti chybám

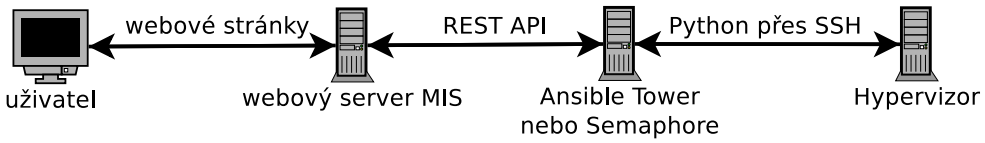
Zabránit uživatelským chybám není již tak technický úkol, jako návrh bezpečného systému. Za hlavní způsob obrany považuji vytvoření jednoduchého a přímočarého uživatelského rozhraní v případě nutnosti doplněného o nápovědu. Dále je také nutné pečlivě kontrolovat vstupy na validitu vyplněných dat. I tak triviální chyba jako překlep v IP adrese může mít nepříjemné následky.

Co se týče programátorských chyb, tak mezi hlavní obrany řadím jednotkové testy a taky fakt, že čím jednodušší a kratší kód je, tím méně může obsahovat chyb.

3.8.3 Oddělení privilegovaných částí aplikace

Tento princip se používá pro zvláště choulostivé části aplikací. Tedy části vyžadující nějaká zvláštní oprávnění oproti zbytku aplikace.

První myšlenka tedy byla vytvořit minimální aplikaci pro webový server, která by vystavovala minimální REST API. To by ale znamenalo na hypervizor nainstalovat webový server nebo použít ve firmě zcela neznámý jazyk a



Obrázek 3.2: Komunikace informačního systému (MIS) s hypervizorem

server napsat v něm. Tyto aplikace by běžely neustále a tedy by byly neustále otevřené k napadení, nepovažuji tedy toto řešení za příliš bezpečné.

Druhá myšlenka byla využít program Ansible, který již ve firmě úspěšně používáme. Nespornou výhodu vidím v tom, že Ansible nepotřebuje na cílovém stroji běh žádného démona kromě programu ssh, na kterém v podstatě vyžaduje využití certifikátů k přihlášení, což bezpečnost dále zvyšuje. Již popsanou nevýhodou je v základu absence jakéhokoli podporovaného rozhraní pro programatické spouštění úlohy. Toto rozhraní přidává program Ansible Tower, který však byl v době návrhu (léto 2017) placený a pro stovky serverů by šlo nejspíše o stovky tisíc ročně. Vybrala jsem tedy program Semaphore [13], který sliboval otevřenou alternativu k Ansible Tower zdarma.

V průběhu implementace informačního systému došlo 7. září k otevření zdrojových kódů Ansible Tower pod jménem AWX [31], čímž odpadla finanční překážka. Kvůli nedokonalostem programu Semaphore popsaných v kapitole 4.2.1 a ustávajícímu vývoji, jsem informační systém přepsala pro použití REST API programu Ansible Tower. Výsledné schéma provádění příkazů je vidět na obrázku 3.2.

3.9 Datové zdroje

3.9.1 Flexibee

V účetním software již jsou zavedeni zákazníci firmy a kompletní fakturace firmy. Naopak chybí evidence virtuálních serverů. Informační systém tedy využil data v evidenci zákazníků (adresář) a data o fakturaci. Zároveň začal využívat evidenci `majetek`, do které ukládá data o virtuálních serverech. Pro účely této evidence bylo potřeba založit speciální dokladovou řadu, tak aby se nepletla s majetkem, který se podle zákona musí evidovat. Virtuální servery pro svoji nulovou hodnotu musí být vedeny mimo běžné účetní procesy.

Zachycení vazby mezi položkou adresáře (zákazníkem) a položkou majetku (virtuálním serverem) se pak řeší uživatelskou vazbou. Uživatelské vazby jsou nástrojem Flexibee, který umožňuje provázat libovolné dvě položky pojmenovanou vazbou [32].

[13] https://github.com/ansible-semaphore/semaphore

3.9.2 iTop

Software iTop je určen pro kompletní evidenci IT vybavení. Jeho silnou stránkou je hlavně evidence vazeb mezi vybavením. Za předpokladu správného vyplnění iTop dokáže říct, že například odpojení napájení ovlivní nějaký fyzický sever a na něm běžící stroje virtuální a rovnou vypsat kterých zákazníků se tento výpadek dotkne.

Informační systém spravuje evidenci virtuálních strojů, jejich vazby na stroje fyzické a vazby na zákazníky.

3.9.3 Icinga

Icinga je software pro monitoring běhu IT infrastruktury, v užším smyslu běhu serverů. Pro každý server je možné definovat neomezený počet sledovaných parametrů. Každý parametr pak je pak měřen jednou službou, která má nastavené prahové hodnoty, po jejichž překročení zahlásí problém. Příkladem takové služby může být například měření zaplnění disku. Službami nahlášené problémy je pak možné označit, že jsme je vzali na vědomí tzv. `acknowledgement` a začali řešit.

Informační systém na úvodní stránce načítá seznam řešených a neřešených problémů a umožňuje problém označit jako v řešení. V detailu každého serveru pak systém zobrazuje všechny měřící služby daného serveru a jejich výsledky.

Pro jednodušší správu konfigurace byl využit doplněk Icinga Director, který dovoluje změny konfigurace přes vlastní REST API. Dále obsahuje tzv. `Self Service API`. Toto API umožňuje, aby se monitorovaný systém sám zaregistroval jediným HTTP požadavkem. Po přijetí požadavku Icinga Director nastaví monitoring předem daných parametrů monitorovaného objektu.

3.9.4 Semaphore

Semaphore je grafická webová nadstavba nad program Ansible. Jedná se o jednodušší alternativu k software Ansibe Tower, jehož hlavní výhoda byla cena, protože je open-source a zdarma.

Informační systém spravuje seznam serverů v kontextu Ansible nazývaný inventář. Do této evidence systém při tvorbě nového serveru i zapisuje. Dále systém čte evidenci `job templates`. Jedná se o předpis spuštění úkolů definovaných jako playbook a případné získání potřebných proměnných od uživatele. Systém rovněž dokáže vybraný `job template` spustit na vybraném serveru a případně získat od uživatele dodatečné proměnné pomocí běžného webového formuláře.

Informační systém taktéž dokáže zobrazit průběh spuštění vybrané `job template` s případnou možností úlohu přerušit nebo spustit znovu.

Kapitola 4

Implementace

V této kapitole popisuji samotnou implementaci informačního systému. Nejedná se však o podrobný popis všech tříd, ale spíše o popis komunikace s ostatními systémy vypíchnutí zajímavých částí implementace.

4.1 Obecné myšlenky

4.1.1 Žádné vlastní úložiště dat

Vzhledem k množství propojovaných systémů jsem se rozhodla, že samotný informační systém nebude mít žádné vlastní úložiště kromě konfigurace. Díky tomuto rysu zcela odpadá řada problémů se synchronizací mezi systémy a konzistencí dat. Bylo však nutné zvolit primární externí systém přes který bude vedena základní evidence. Za tento primární systém jsem zvolila účetní program Flexibee. Jednak je jeho API poměrně rychlé a dobře popsané, ale hlavně se pro evidenci zákazníků a serverů, tedy majetku, nejlépe hodí. Dalším jeho výrazným rysem je podpora externích identifikátorů. Jedná se o možnost ke každému záznamu přiřadit libovolné množství dalších unikátních identifikátorů. Této vlastnosti jsem využila a ke každém záznamu o virtuálním serveru jsem uložila i identifikátory ostatních evidencí. Výsledný záznam o virtuálním serveru pak ve Flexibee z pohledu PHP vypadá takto:

```
[
    "external-ids" => [
        "ext:icinga/viky-testing",
        "ext:itop/664",
        "ext:Ansible/viky-testing"
    ],
    "id" => "1",
    "kod" => "M00001"
]
```

4.1.2 Jednotný přístup k REST API

Kvůli snazšímu vývoji a možnosti integrace do ostatních programů firmy jsem jako samostatný projekt vyčlenila skupinu tříd pro komunikaci s REST API. Všechna používaná API poskytují data ve formátu JSON. Tento formát je v jazyce PHP snadno převeditelný na asociativní pole, které využívají všechny programy firmy a i moje práce. Na pozadí pak využívám obsáhlou knihovnu Guzzlehttp obalující práci s programem Curl.

Za zajímavou část v tomto ohledu považuji využití metody Promises (slibů). V kontextu komunikace přes Curl se jedná o metodu, kdy na požadavek na stažení dat, funkce místo dat vrátí pouze slib, že data dodá. Tímto způsobem je možné dosáhnout souběžného stahování dat i v jazyce PHP, který jinak práci s vlákny nepodporuje, přesněji ji podporuje pouze v režimu příkazového řádku [33]. Interně se využívají funkce z rodiny `curl_multi_*`.

Využití Promises pak přináší zrychlení při práci s více zdroji naráz. Již při dvou souběžných požadavcích se běžně stává, že celkový čas strávený stahováním všech dat je delší než celkový čas vykreslení stránky. Toho je dosaženo právě souběžným stahováním. S rostoucím počtem souběžných požadavků se rozdíl mezi sériovým a paralelním stahováním logicky zvětšuje.

Problém s využitím Promises poskytnutých knihovnou Guzzlehttp je nutnost přepsání všech programů, které pracují s poli. Tento problém jsem vyřešila třídou `PromiseArrayWrapper`, která implementuje rozhraní `ArrayAccess` a `IteratorAggregate`. Třída se tedy chová stejně jako pole s výjimkou úprav přes referenci. Data jsou stahována na pozadí a případné blokování, kvůli stahování dat, nastává až při prvním přístupu. V ten okamžik jsou však data již částečně nebo zcela stažena. Je tedy možné nasadit Promises i v kódu, který s jejich využitím nepočítá a to bez jeho změny.

Toto řešení však skrývá i některé záludnosti ilustrované na následujících příkladech:

```php
<?php
public function actionDefault(int $id) {
    $events = $this->tower->getTaskOutput($id);
    foreach ($events as $event) { ... }
    $this->task = $this->tower->getTask($id);
}
?>
```

```php
<?php
public function actionDefault(int $id) {
    $this->task = $this->tower->getTask($id);
    $events = $this->tower->getTaskOutput($id);
    foreach ($events as $event) { ... }
}
?>
```

V obou případech bude výsledek funkce stejný. První implementace funkce však bude pomalejší, protože program nejdříve dostane slib, poté musí být slib kvůli přístupu k datům ihned splněn a až nakonec požádá o další data (tedy slib). Druhá implementace nejdříve zažádá o data, která potřebuje a dostane dva sliby. Na pozadí se začne s paralelním stahováním dat. Jeden ze slibů bude muset být opět okamžitě splněn jako v prvním případě, ale při čekání na splnění tohoto slibu se stále v pozadí pracuje na splnění i druhého slibu. Za podmínek nulové režie by byl druhý způsob dvakrát rychlejší.

V reálném prostředí jsem na stránce používající tuto funkci naměřila nástrojem ApacheBench zlepšení z průměrných 950 ms na průměrných 690 ms. Celková doba strávená plněním slibů pak, podle interního nástroje, který jsem pro tento účel napsala, je okolo 700 ms. Správné použití Promises tedy umožní načíst stránku rychleji, než sériové řešení vůbec získá potřebná data k tomu aby stránku začalo vykreslovat.

Kvůli různým implementacím vzdálených API je pak součástí knihovny i abstraktní třída `PromiseDecorator`, která kvůli návrhovému vzoru Dekorátor usnadňuje flexibilní přidávání dalších vlastností pro Promise[34]. Například třída `PromiseDecoratorFlexibee` přidává k běžné Promise ještě „rozbalení polí“. Každá odpověď od Flexibee API je obsažena v poli o jenom prvku s klíčem `winstrom`, jehož psaní je otravné.

4.2 Semaphore a Ansible Tower

Jediný potřebný software, který firma zatím nepoužívala byla webová nadstavba nad Ansible. Nadstavba nad Ansible je nutná kvůli bezpečnosti a pohodlí práce. V neposlední řadě takováto nadstavba umožní technikům spouštět Ansible skripty i bez použití informačního systému. Vzhledem k (již minulému) zpoplatnění software Ansible Tower padla původně volba na využití software Semaphore.

4.2.1 Semaphore

Software Semaphore nabízí jednoduchou webovou nadstavbu pro spuštění a správu Ansible s open-source (MIT) licencí. Semaphore nabízí komplexní správu uživatelů, inventáře (tedy seznamu serverů) a Ansible skriptů. Před spuštěním každého skriptu umožňuje zvolit i případné dodatečné parametry. Semaphore taktéž poskytuje REST API pro všechny operace dostupné z uživatelského rozhraní. Osobně Semaphore považuji za velmi příjemný a jednoduchý systém.

Bohužel některá omezení činí jeho využití ve vznikajícím informačním systému složité. Velmi nepříjemným omezením je nemožnost omezit vykonání Ansible skriptu pouze na určitých cílech (parametr `-limit`). Tato možnost je přístupná pouze při definici skriptu, před každým spouštěním, které bylo nutné omezit pouze na určité cíle, bylo nutné upravit definici skriptu.

Další nedokonalost poté vidím v nemožnosti definovat pro uživatele spouštějícího skript dotazník zjišťující potřebné proměnné. Při spouštění Ansible z příkazová řádky je možné použít direktivu `vars_prompt`, tak však v neinteraktivním prostředí nefunguje. Semaphore tak po uživateli spouštějícím skript požaduje znalost parametrů spouštěného skriptu. Tento požadavek je však v prostředí firmy poměrně těžko splnitelný.

Stejná omezení platí i pro Semaphore REST API. Při přístupu přes API je problém nemožnosti definovat dotazník na proměnné ještě palčivější. Informační systém totiž vůbec nemá přístup k definicím jednotlivých skriptů a nemůže tedy ani žádným způsobem požadované proměnné načíst.

Avšak jak již bylo uvedeno v podkapitole 3.8.3 byl software poskytující webovou nadstavbu v průběhu implementace vyměněn. Výsledným software pro správu Ansible se tedy stal Ansible Tower.

4.2.2 Ansible Tower

Software Semaphore byla navržen jako otevřený klon Ansible Tower. Vlastnosti popsané v předchozí kapitole tedy platí i pro Ansible Tower s výjimkou omezení. Ansible Tower umožňuje například definovat dotazníky před spuštěním skriptu. Tyto dotazníky pro každou otázku umí definovat její text, popis, datový typ odpovědi a případnou výchozí hodnotu odpovědi. Dále je možné se před spuštěním zeptat na tagy, parametr `-limit` atp.

Další velkou výhodu Ansible Tower vidím v implementaci uživatelského rozhraní jako client-side javascriptové aplikace. Toto uživatelské rozhraní nepoužívá žádné speciální API, ale data načítá přímo z veřejného REST API.

Vzhledem k možnostem Asible Tower jsem se rozhodla do informačního sytému integrovat univerzální spouštěč Ansible skriptů s podporou dotazníků a výběru tagů.

4.3 Ansible skripty

Vzhledem ke skutečnostem popsaným v podkapitole 3.8.3 informační systém využívá program Ansible ke spouštění skriptů na spravovaných serverech. V tomto ohledu se jedná jak o hypervizor tak virtuální servery. V podkapitolách dále popisuji jednotlivé skripty (playbooky)

4.3.1 Vytvoření virtuálního serveru

Tento playbook využívá nástrojů libvirt [14] instalaci nových virtuálních serverů. Skript se skládá ze získání potřebných proměnných a spuštění příkazu `virt-install`. Jeho důležitou součástí je také tzv. `kickstart` soubor. Jedná se o soubor ve speciálním formátu, který definuje odpovědi na všechny otázky

[14] https://libvirt.org/

v průběhu instalace. Tím pádem probíhá celý proces instalace bez jakékoli interakce s uživatelem, což je nutnou podmínkou toho, aby bylo možné instalaci provádět pomocí `Ansible`.

Firma využívá výhradně Operační systém (OS) CentOS 7 a tedy tento skript se zabývá pouze instalací tohoto OS. Jeho rozšíření na další typy na Linuxu založených operačních systémů by však mělo být velmi jednoduché, protože tento druh instalace nepodporují pouze systémy z rodiny RedHat[15], ale například také Ubuntu [16].

4.3.2 Vypnutí a zapnutí serveru

Nástroj Ansible obsahuje základní podporu pro správu virtuálních strojů spravovaných pomocí libvirt [17]. Chybí například výše zmíněná instalace. Naopak nechybí možnosti pro vypínání a zapínání a proto je tento skript využívá, díky tomu se celý vejde na 27 řádek.

Pro určení operace, která se má provést využívám Ansible tagů. Tagy se v Ansible využívají pro označení částí skriptů. Při spuštění skriptu je pak možné jednotlivé tagy uvést a skript spustí pouze stejně označené úlohy [35].

4.3.3 Registrace do monitoringu

Tento skript se stará o instalaci klienta monitorovacího systému Icinga. Jedná se o poměrně komplikovaný proces, kdy je nutné přidat repozitář, zaregistrovat klientskou instalaci na hlavním serveru, správně vystavit certifikáty pro komunikaci a vytvořit konfigurační soubory. Skript také využívá již zmíněné `Self Service API` (viz podkapitola 3.9.3).

Vzhledem ke složitosti implementace považuji za úspěch, že skript je v souladu s pravidly tvorby playbooků idempotentní, tedy nezáleží na počtu spuštění tohoto skriptu. Výsledek bude vždy stejný.

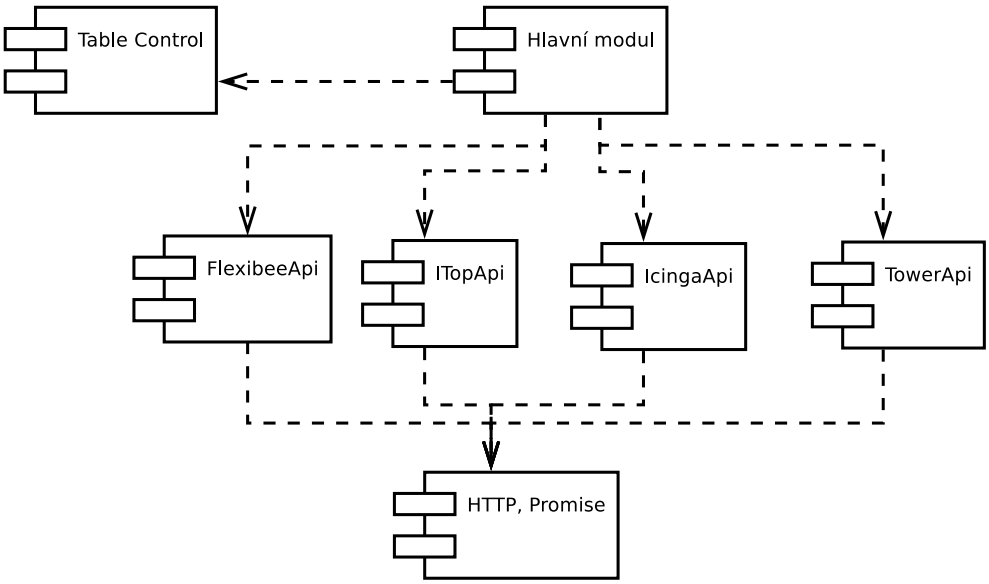
4.3.4 Další skripty

Informační systém je napsaný tak, aby dokázal spouštět jakékoliv Ansible skripty. Mezi již používané patří například skript pro instalaci či aktualizaci firemního eshopu.

4.4 Modularita

Aplikaci jsem rozdělila do tří základních druhů modulů popsaných v následujících sekcích. Do prvního druhu spadá jen jeden modul, obsahuje základní aplikaci s jejím grafickým rozhraním a definicí závislostí. Druhý druh modulů

[15] společnost vyvíjející distribuce Red Hat Enterprise Linux, CentOS a Fedora
[16] https://help.ubuntu.com/community/KickstartCompatibility
[17] http://docs.ansible.com/ansible/latest/virt_module.html



Obrázek 4.1: Schéma závislostí modulů

jsou datové vrstvy komunikující s jednotlivýma API. Třetím druhem jsou pak pomocné moduly.

O stažení a načítání modulů se pak stará program Composer, který „dovoluje deklarovat libovolně složité závislosti jednotlivých knihoven a pak je za nás nainstaluje do našeho projektu.“[36] Composer využívá rozsáhlý hlavní repozitář, ale dovoluje definovat i vlastní repozitáře například pro neveřejný kód. Datové moduly navíc ještě využívají možnosti rozšíření DIC poskytnutého frameworkem Nette. Výsledné schéma závislostí modulů je zachycené na obrázku 4.1.

4.4.1 Hlavní modul

Úkolem hlavního modulu je definovat všechny potřebné závislosti, spravovat konfiguraci a také tento modul obsahuje uživatelské rozhraní. Obsažení uživatelského rozhraní přímo v hlavním modulu jsem zvolila primárně z důvodu jednoduchosti. Ačkoliv je možné presentery a komponenty (tj. třídy definující uživatelské rozhraní) oddělit do samostatných modulů, tak presentery často požadují více datových zdrojů. To by vedlo buď k jednomu velkému modulu s velkým množstvím závislostí nebo k situaci, kdy by každý presenter měl vlastní modul a tím by vznikla alespoň desítka dalších modulů.

Dále hlavní modul obsahuje třídu implementující logiku zakládání serverů.

4.4.2 Datové moduly

Datové moduly navenek poskytují rozhraní pro jednoduché získávání dat ze čtených REST API. Uvnitř poté zajišťují správné dotazovaní na zdroje a pomocí Promise modulu zajišťují samotné skládání HTTP požadavku a jeho následné zpracování. Snahou bylo napsat vnější rozhraní všech modulů co nejvíce jednotné.

Datovým modulům musí být z hlavní aplikace předána konfigurace přístupových údajů k daným API. Toho je docíleno technologií rozšíření frameworku Nette. Definici rozšíření v každém modulu zajišťuje třída ze jmenného prostoru `DI` daného modulu. Třída zajistí načtení své předdefinované konfigurace a zaregistruje své třídy do DIC kontejneru.

4.4.2.1 FlexibeeApi

Tento modul zajišťuje komunikaci s účetním software Flexibee. Jak již bylo v práci zmíněno, firma Arit se specializuje na software rozšiřující možnosti software Flexibee. Práce využívá již dříve napsanou třídu `ObjFlexibee`, za jejímž vznikem stojí kolega Martin Klein. Já jsem třídu dále rozšířila o možnost načítání sumací, možnost využití cache[18] a použití technologie Promise.

4.4.2.2 iTopApi

Přidání REST API do programu iTop proběhlo až v nedávných verzích. Možná i to zapříčinilo, že se jedná o API relativně nezvyklé. Všechny zdroje sice mají unikátní URI, tedy jedinečnou cestu k daným datům. Tato cesta se však zapisuje ve formátu JSON přímo do URI. Výsledné URI tedy není příliš čitelné, jak je možné vidět na následující ukázce, která načítá data o jednom serveru:

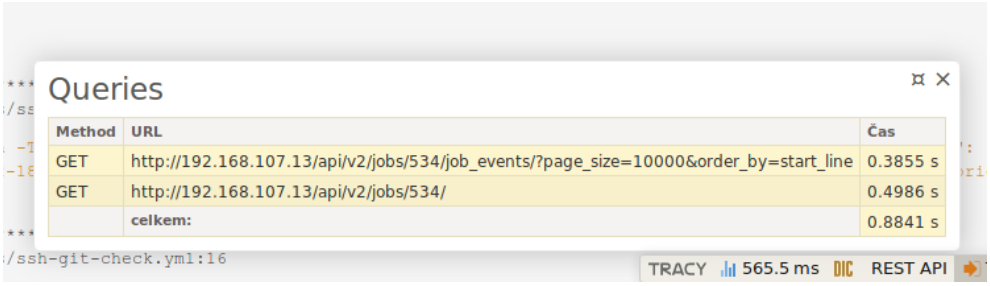
```
rest.php?version=1.3&json_data=
{"key":664,"operation":"core\/get","class":"VirtualMachine"}
```

Další specifikum API programu iTop spočívá ve filtrování dat (klíč `key` v přechozí ukázce). Při číselné hodnotě se `key` považuje za unikátní identifikátor záznamu. Při textové hodnotě se předaný výraz vyhodnocuje jako výraz jazyka OQL. Jazyk OQL je velmi podobný jazyku SQL a dá se říci, že se jedná o jeho podmnožinu[37]. Bohužel jazyk OQL a tedy ani REST API programu iTop nepodporuje například stránkování, což spolu s celkově nízkou rychlostí API působí pomalejší načítání stránek i v samotném informačním systému.

4.4.2.3 IcingaApi

Datový modul pro čtení REST API programu Icinga musí překonat pouze jednu nepříjemnost, která souvisí až s implementací knihovny Guzzlehttp.

[18] V této práci není cache použita.



Obrázek 4.2: Rozšíření na sledování REST API do Tracy

Problém spočívá v předávání HTTP parametrů (dat za znakem ? v URL). Knihovna Guzzlehttp tyto parametry očekává jako asociativní pole, ale program Icinga v některých případech požaduje opakované uvedení stejných parametrů. Asociativní pole však mají klíče unikátní.

Tento problém modul řeší předáváním některých parametrů jako textových řetězců.

4.4.2.4 TowerApi

Vytvoření datového modulu pro čtení REST API programu Ansible Tower bylo nejsnazší ze všech. Jeden z důvodů vidím ve skutečnosti, že JavaScript tvořící uživatelské rozhraní využívá běžné veřejné API a z tohoto důvodu nestojí vývoj REST API na vedlejší koleji. Viditelně velké úsilí vývojáři také věnovali rychlostní optimalizaci zvláště s ohledem na souběžné načítání dat. Ačkoliv jsem neprováděla žádná exaktní měření, tak mi přijde API programu Ansible Tower ze všech nejrychlejší a také nepříjemnější na práci.

4.4.3 HTTP/Promise modul

Dále práce obsahuje Promise modul zajišťující HTTP komunikaci. Jeho hlavní přednosti shrnuje kapitola 4.1.2.

Pouze naznačen však byl způsob monitorování HTTP požadavků. Požadavky jsou monitorovány rozšířením do Tracy, integrovaného ladícího software frameworku Nette. Rozšíření implementují rozhraní `IBarPanel`, které vyžaduje jen definici způsobu zobrazení. Výsledek je možné vidět na obrázku 4.2, na kterém ještě stojí za povšimnutí, že načítání dat souhrnně trvalo téměř 880 ms, ale stránka byla vygenerována již za 560 ms. Tohoto výsledku je dosaženo souběžným načítáním dat pomocí technologie slibů.

4.4.4 TableControl modul

Modul a zároveň jediná třída `TableControl` poskytuje jednoduchý nástroj pro vykreslování dat do tabulky. Stačí v kódu definovat strukturu vykreslovaných polí, tedy například jejich názvy a datové typy a zaregistrovat `callback`, který se bude startat o poskytování dat. Tomuto callbacku jsou pak třídou volitelně předány požadavky uživatele na číslo stránky, řazení a vyhledávání.

Vzhledem k implementaci modulu jako Nette komponenty (viz podkapitola 4.6) je velmi snadné provést integraci do jiných systémů. I díky tomu se tento modul již používá v dalším projektu firmy.

4.5 Testování

Součástí zdrojových kódů informačního systému jsou také testy. V případě testů jednotlivých modulů jde o testy jednotkové, testující samotné metody. V případě hlavní aplikace jsou předmětem testů prvky uživatelského rozhraní a funkčnost presenterů, jedná se tedy spíše o testy integrační.

Pro oba typy testů jsem použila program Nette Tester. Původně jsem testy psala s použitím programu PHPUnit, který jsem ale shledala nedostatečným při testování presenterů, primárně kvůli značnému množství závislostí, které bylo třeba ručně vytvářet. Jinak považuji oba programy za velmi podobné a způsob psaní testů pro oba programy je téměř totožný. Jediný zásadní rozdíl vidím v odlišném zápisu assercí.

4.5.1 Testy datových modulů

„Testování na úrovni modulu (neboli jednotky) znamená výrazně jednodušší zadání i provádění než testování integrace. V zásadě můžeme doporučit, aby každý modul měl svou vlastní testovací jednotku zabudovanou do svého kódu a aby tyto testy byly prováděny automaticky jako součást pravidelného procesu tvorby.“[13]

Na základě předchozího doporučení každý z datových modulů obsahuje vlastní sadu jednotkových testů a konfigurační soubor `.gitlab-ci.yml`, který zajišťuje automatické spouštění testů ve chvíli nahrání (push) zdrojových kódu na verzovací server. Vzhledem k tomu, že se jedná o moduly pouze pro získávání dat, byla nutná jejich správná izolace od externích zdrojů. Další komplikací při testování bylo použití Promises, jejichž obalování HTTP požadavků taktéž není snadné nasimulovat.

Problém izolace se mi podařilo vyřešit knihovnou Mockery[19]. Knihovna vytváří testovací dvojníky, tzv. mock objekty, které je možné využít místo skutečných objektů. Této vlastnosti využívám pro simulování načítání dat z REST API. Místo skutečné implementace z Promise modulu (4.1.2) použiji

[19] https://github.com/mockery/mockery

dvojníka, který vrací předem definovaná data bez závislosti na dostupnosti externího zdroje.

4.5.2 Testy HTTP/Promise modulu

Promise modul se skládá ze dvou částí, které bylo nutné testovat odděleně. Prvním případem je třída RestPanel zobrazující informace o HTTP požadavcích aplikace. Překážkou v testování bylo použití statických metod a privátních statických proměnných, které bylo nutné před každým testem vymazat. K tomuto vymazání jsem použila vlastnosti anonymních funkcí v PHP, které je možné za běhu připojit k libovolnému objektu a měnit i privátní proměnné. Osobně se mi vůbec nelíbí, že je něco takového možné, pro účely testování se však jedná o velmi mocný nástroj.

Druhá část modulu se stará o samotné dotazovaní na REST API. Zde jsem nepoužila testovacích dvojníků, ale využívám vzdálený server `jsontest.com`, který poskytuje jednoduché REST API vhodné pro testování. Nevýhodu takových testů vidím v závislosti na internetovém připojení k testovacímu serveru. Alternativní možností by bylo vytvořit si takovýto REST API server pro testování ručně.

4.5.3 Testy uživatelského rozhraní

V hlavním modulu se testují jednotlivé akce presenterů. Akce jsou pro uživatele odlišenou rozdílnou URL adresou a z pohledu aplikace má každá akce presenteru odpovídající metodu pro zpracování. Automatický test jednotlivě volá tyto funkce a kontroluje jejich výstup.

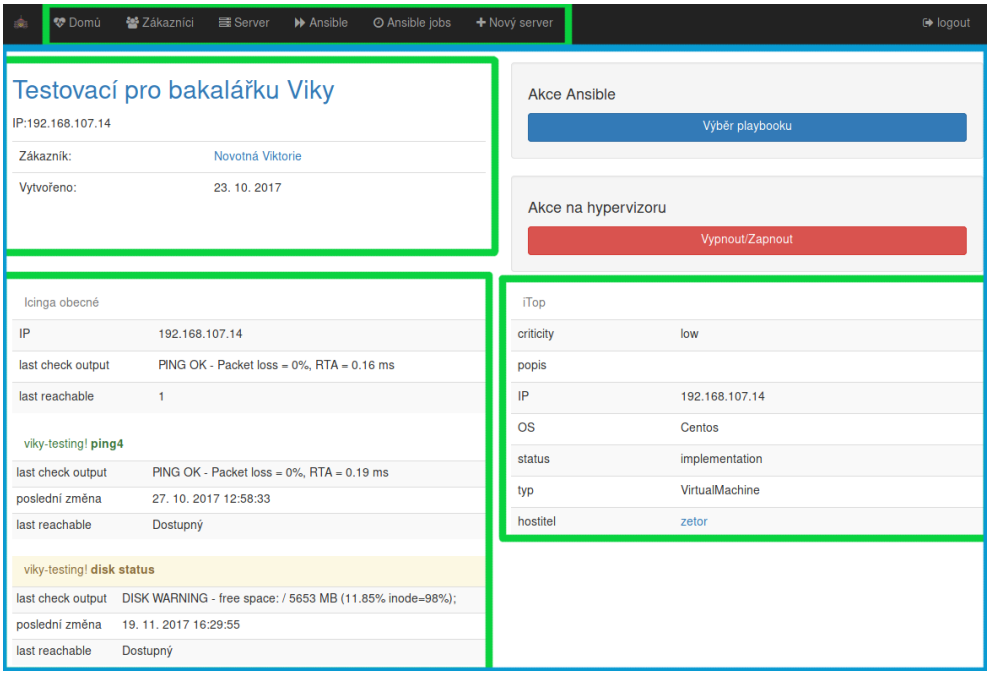
4.6 Nette komponenty

V kontextu Nette „Komponenta představuje vykreslitelný objekt. Jsou to například formuláře, menu, ankety a podobně. V rámci jedné stránky jich může existovat libovolný počet.“[38] Komponenty[20] pomáhají se znovupoužitelností prvků uživatelského rozhraní a informační systém komponent v hojné míře využívá. Téměř všechny prvky uživatelského rozhraní jsou komponentou. Dále popisuji některé zajímavé komponenty.

4.6.1 Detail serveru

Stránka detailu serveru se skládá z komponenty `ServerDetail` (viz obrázek 4.3 modrý rámeček), která obsahuje ještě další tři komponenty pro jednotlivé zdroje dat (viz obrázek 4.3 zelené rámečky). Výzvou v tomto ohledu bylo získání dat v pro uživatele přijatelném čase. Zvláště REST API programu iTop

[20]Třídy komponent se vytváří jako potomci třídy `Control`, v češtině se však ujal název komponenta.



Obrázek 4.3: Znovupoužitelné komponenty na stránce detailu serveru (vyznačené zeleně a modře)

vrací data jednoho serveru i několik vteřin. Řešení souběžného stahování dat pomocí slibů (viz 4.1.2) nebylo dostatečné, protože i pomocí této technologie je nutné před vykreslením stránky počkat na všechny zdroje dat.

Problém jsem vyřešila souběžným načítáním až samotných komponent pomocí technologie AJAX. Předně však bylo nutné takovému chování komponenty naučit. Pro tento úkol jsem se pokusila využít již hotovou knihovnu `AsyncControl` [21] ačkoliv se jedná pouze o několik měsíců starý kód, je tato knihovna určena pouze pro Nette verze 2.2 z jara 2014. Po vyzkoušení však knihovna fungovala bezchybně i pro Nette verze 3.0. Bohužel se mi nepodařilo kontaktovat původní autory ohledně začlenění podpory novějších verzí. Tato knihovna také při souběžném načítání komponent nepodporuje předávání parametrů, věc u komponent jinak zcela běžnou.

Výše popsané problémy a fakt, že je knihovna pod open-source licencí mne vedla k vytvoření forku. V této kopii jsem pak nastavila podporu novějších verzí Nette. A hlavně knihovnu rozšířila o možnost předávání parametrů do komponent.

[21] https://github.com/peckadesign/AsyncControl/

4.6.2 Menu

Komponenta pro vykreslování menu je jednou z komponent, která se již využívá v dalším projektu. Jedná se o poměrně jednoduchou komponentu, která vykreslí dodané pole jako menu. Datová vrstva může být jakákoliv třída implementující požadované rozhraní. Při vykreslování pak komponenta jako jediný parametr bere aktuální presenter a jeho položku v menu zobrazí odlišně. Každá položka v menu pak může mít nastavena přístupová oprávnění pro zobrazení.

4.6.3 Formulář na tvorbu serverů

Formulář pro vytvoření serveru je také implementován jako komponenta. Komponenta obsahuje samotné vykreslování, které se provádí specificky upraveným kódem, tak aby výsledný vzhled ladil s pravidly, která nastavuje framework Bootstrap. Dále se také stará o validaci vložených dat a poskytování nápovědy formou nabízení možných odpovědí. Zpracování dat poté komponenta nechává na nadřazené komponentě popřípadě presenteru.

Kapitola 5

Zhodnocení a doporučení dalšího rozvoje

Vzniklou aplikaci rozhodně nepovažuji za zcela kompletní a vyčerpávající všechny možnosti. V této kapitole se zabývám zhodnocením praktické části a možností dalšího rozvoje, ať již se jedná o nasazení, rozšíření existujících modulů nebo přidání zcela nových.

5.1 Zhodnocení praktické části

Kromě samotné aplikace bylo v prostředí společnosti nasazeno několik nových systémů. Kromě LDAP serveru, která aplikace využívá pouze omezeně, se jedná hlavně o systém Ansible Tower. To zdržovalo vývoj samotné aplikace.

Aplikaci samotnou považuji za vyústění mých přibližně pětiletých zkušeností s vývojem webových aplikací. I tak se v kódu samozřejmě objevují místa lepší a místa horší.

5.1.1 Modularita

K vytvoření modulů jsem vybrala způsob poskytovaný samotným Nette frameworkem [39]. Tento systém se ukázal jako velmi komplexní a jeho použití není zrovna snadné. Navíc přidaná hodnota tohoto řešení je pro tuto aplikaci pouze v kontrole konfigurace.

Mám tedy určité obavy o vývoj dalších modulů. Myslím si, že ne každému programátorovi se bude chtít psát moduly tímto způsobem. Tato obava je o to větší, že společně s touto prací jsem ve firmě navrhla ještě jinou modulární aplikaci. V té sice nejsou moduly snadno přenositelné do jiné aplikace, ale jejich vývoj nepotřebuje žádné dodatečné znalosti.

5.1.2 Promises

Naopak za podařené považuji návrh a využití technologie slibů (Promises), zvláště pak jejich maskování jako běžné pole. Celkově však považuji vícevláknové získávání dat v jazyce, který více vláken nepodporuje, za hezký trik, který je navíc možné okamžitě použít v dalších aplikacích.

5.2 Produkční nasazení

Aplikace je momentálně nasazena pouze v testovacím provozu, kdy se ověřuje její funkčnost v praxi. Vzhledem k tomu, že se s tímto informačním systémem zavádí do produkce i software Ansible Tower a LDAP, bylo rozhodnuto, že se nejdříve řádně nasadí a otestují tyto služby a až pak jejich nadstavby.

Osobně vidím problém současného nastavení v neexistenci uživatelských rolí. Uživatelé se rozlišují pouze na dvě skupiny. Na skupinu uživatelů kteří přístup do systému mají a na skupinu bez přístupu. Jak framework Nette tak například komponenta na vykreslování menu již obsahují veškerou nutnou přípravu na použití rolí. Jejich přidání do systému by tedy mělo být spíše otázkou minut než hodin. Nutnou podmínkou je však specifikace, která role má mít přístup k čemu. Toto rozhodnutí se neodvažuji sama učinit.

5.3 Další moduly

5.3.1 Ticket systém

V naší firmě se momentálně používají dva systémy pro správu úkolů a zadávání práce. Vývojářské oddělení tomuto účelu používá systém Gitlab, zbytek firmy poté na míru vytvořený FlexiServis (viz také podkapitola 3.2.4).

Gitlab je webovým správcem repozitářů verzovacího systému Git. Kromě samotné správy repozitařů Gitlab poskytuje ještě další řadu nástrojů, mezi hlavní řadím kontinuální integraci[22], která může zajišťovat automatické testování a nasazování kódu a ticket systém. Oproti některým externím ticket systémům má tento integrovaný výhodu přímého provázání na zdrojový kód a jeho verze (tzv. commit). To umožňuje zobrazit přímo chybou ovlivněný kód a jeho opravu. Dále je také možné jednotlivé hlášení uzavírat přímo v tzv. commit message, tedy bez návštěvy webového rozhraní programu Gitlabu, přímo v příkazové řádce počítače vývojáře. Nevýhodou tohoto ticket systému je však absence přímé vazby na zákazníka.

FlexiServis oproti integrovanému ticket systému v programu Gitlab obsahuje přímou vazbu na zákazníka, vazbu na výkaz práce na konkrétním požadavku. Tímto v účetním software přímo vznikají podklady pro fakturaci.

[22]Continuous Integration, zkráceně CI

Oba systémy mají své výhody a nevýhody. Logické řešení tedy vidím v jejich vzájemném sloučení a to je přesně úkol, který pro jiné systémy již řeší můj informační systém.

5.3.2 Automatizace nasazení kódu

Software Gitlab nabízí nástroje pro automatické testování a nasazování kódu, avšak k tomuto úkolu vyžaduje na cílových strojích běh relativně komplexní služby (tzv. Gitlab Runner). Pro účely automatického spouštění testů se jedná o skvělý nástroj. K běhu tohoto programu totiž dojde vždy, když někdo na server nahraje změnu zdrojových kódů. Jeho provoz na produkčních serverech však vyvolává obavy o výkon, bezpečnost a také chyby programátorů, kdy se automaticky nasadí nefunkční kód. Navíc nasazování kódu má firma již částečně zautomatizované pomocí Ansible.

Informační systém by mohl monitorovat REST API software Gitlab a pro předem definované akce by spouštěl patřičné Ansible skripty.

Závěr

Cílem práce bylo analyzovat problémy související se správou serverů v společnosti Arit s.r.o. vznikající především ne zcela dokonalou evidencí. Bylo zjištěno, že tyto nedokonalosti v evidenci jsou způsobeny hlavně množstvím nespolupracujících informačních systémů a s tím související časovou náročností vedení evidence. Kvůli omezením jednotlivých systémů a s ohledem na rozšiřitelnost byl vytvořen nový informační systém.

Tento systém pak byl implementován s velkým důrazem na budoucí rozšiřitelnost a znovupoužitelnost kódu. Vzniklá aplikace tak obsahuje hlavní modul definující uživatelské rozhraní, pomocné moduly pro vykreslování prvků a samozřejmě datový modul pro každý zdrojový systém.

Vzhledem ke skutečnosti, že informační systém pracuje přímo s fyzickými servery, byl velký důraz kladen také na bezpečnost. Nejzranitelnější operace proto nevykonává přímo vzniklý systém, ale požádá o jejich vykonání software Ansible.

Využití programu Ansible jde v informačním systému dále než je pouhé ovládání hypervizoru. Systém je schopen spouštět jakékoli další Ansible skripty a v tomto ohledu vznikl systém mnohem univerzálnější než bylo zadáním požadováno.

Uživatelské rozhraní bylo implementováno s ohledem na používání na mobilních zařízeních a s ohledem na co největší možnou jednoduchost.

Informační systém je momentálně ve firmě nasazen v testovacím provozu. Jeho plné nasazení do praxe se zatím nepodařilo. Hlavním důvodem bylo množství nově zaváděného software do firmy, které informační systém využívá a z toho vyplývající nutnost otestovat nejdříve tento software.

Výsledný software plní zadáním stanovené úkoly a případech, kdy to dávalo smysl, je uzpůsoben i pro plnění dalších úkolů.

Některé části implementace pak používají postupy ve firmě dříve neznámé a tyto postupy vyčleňuje do samostatných modulů. Tímto způsobem pak tato práce pomáhá zlepšit i další software firmy.

Literatura

[1] Virtualization Essentials. [Cited 2018-01-05]. Dostupné z: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/ebook/gated-vmw-ebook-virtualization-essentials.pdf

[2] Automation versus Orchestration. [Cited 2018-01-05]. Dostupné z: https://devops.com/automation-versus-orchestration/

[3] What is cloud computing? [Cited 2018-01-05]. Dostupné z: https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/

[4] WHMCS Feature Tour. [Cited 2018-01-05]. Dostupné z: https://www.whmcs.com/tour/

[5] cPanel with KVM. [Cited 2018-01-06]. Dostupné z: https://forums.cpanel.net/threads/cpanel-with-kvm.255611/

[6] VMmanager. [Cited 2018-01-05]. Dostupné z: https://www.ispsystem.com/software/vmmanager

[7] Nations, D.: Improve Your Understanding of Web Applications [online]. [Cited 2018-01-05]. Dostupné z: https://www.lifewire.com/what-is-a-web-application-3486637

[8] 3 Types Of Web Application Architecture [online]. [Cited 2018-01-05]. Dostupné z: https://mobidev.biz/blog/3_types_of_web_application_architecture

[9] HTTP. [Cited 2018-01-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[10] Bidelman, E.: Stream Updates with Server-Sent Events. 2010, [Cited 2018-01-07]. Dostupné z: https://www.w3.org/TR/notifications/

[11] Web Notifications. [Cited 2018-01-07]. Dostupné z: https://www.w3.org/TR/notifications/

[12] Push API. [Cited 2018-01-07]. Dostupné z: https://developer.mozilla.org/cs/docs/Web/API/Push_API

[13] Hunt, A.; Thomas, D.: *Programátor pragmatik: jak se stát lepším programátorem a vytvářet kvalitní software*. Computer Press, 2007, ISBN 9788025116609.

[14] PHP The Wrong Way. [Cited 2018-01-07]. Dostupné z: http://www.phpthewrongway.com

[15] Zabezpečení před zranitelnostmi [online]. [Cited 2017-12-02]. Dostupné z: https://doc.nette.org/cs/2.4/vulnerability-protection

[16] Safe Password Hashing. [Cited 2018-01-07]. Dostupné z: http://php.net/manual/en/faq.passwords.php

[17] Abra Flexibee [online]. [Cited 2017-11-21]. Dostupné z: https://www.arit.cz/flexi/flexiabra-flexibee/

[18] Icinga 2 [online]. [Cited 2017-12-22]. Dostupné z: https://www.icinga.com/products/icinga-2/

[19] What is Ansible? [online]. [Cited 2017-11-23]. Dostupné z: https://www.ansible.com/quick-start-video

[20] Python API [online]. [Cited 2017-12-03]. Dostupné z: http://docs.ansible.com/ansible/latest/dev_guide/developing_api.html

[21] Možnosti integrace [online]. [Cited 2017-12-18]. Dostupné z: https://www.flexibee.eu/api/moznosti-integrace/

[22] Uživatelské tlačítko [online]. [Cited 2017-12-18]. Dostupné z: https://www.flexibee.eu/api/dokumentace/ref/uzivatelske-tlacitko

[23] Standards [online]. [Cited 2017-11-25]. Dostupné z: https://xkcd.com/927/

[24] Usage of server-side programming languages for websites [online]. [Cited 2017-11-26]. Dostupné z: https://w3techs.com/technologies/overview/programming_language/all

[25] Function arguments [online]. [Cited 2018-01-04]. Dostupné z: http://php.net/manual/en/functions.arguments.php#functions.arguments.type-declaration

[26] MVC aplikace & presentery [online]. [Cited 2017-12-07]. Dostupné z: https://doc.nette.org/cs/2.4/presenters

[27] PHP Standards Recommendations [online]. [Cited 2017-12-22]. Dostupné z: http://www.php-fig.org/psr/

[28] Coding Standards [online]. [Cited 2017-12-22]. Dostupné z: https://nette.org/en/coding-standard

[29] Grudl, D.: Best practices pro jmenné prostory v PHP [online]. [Cited 2017-12-22]. Dostupné z: https://phpfashion.com/best-practices-pro-jmenne-prostory-v-php

[30] Grudl, D.: Proč Nette nedodržuje standardy PHP-FIG / PSR? [online]. [Cited 2017-12-22]. Dostupné z: https://phpfashion.com/proc-nette-nedodrzuje-standardy-php-fig-psr

[31] Red Hat Advances Enterprise and Network Automation with New Ansible Offerings [online]. [Cited 2017-12-03]. Dostupné z: https://www.redhat.com/en/about/press-releases/red-hat-advances-enterprise-and-network-automation-new-ansible-offerings

[32] Uživatelské vazby [online]. [Cited 2017-12-02]. Dostupné z: https://www.flexibee.eu/api/dokumentace/ref/uzivatelske-vazby/

[33] pthreads Introduction [online]. [Cited 2017-11-27]. Dostupné z: http://php.net/manual/en/intro.pthreads.php

[34] Böhmer, M.: *Návrhové vzory v PHP*. Computer Press, 2012, ISBN 9788025133385.

[35] Tags [online]. [Cited 2017-12-08]. Dostupné z: http://docs.ansible.com/ansible/latest/playbooks_tags.html

[36] Composer [online]. [Cited 2017-12-02]. Dostupné z: https://doc.nette.org/cs/2.4/composer

[37] Object Query Language Reference [online]. [Cited 2017-12-24]. Dostupné z: https://wiki.openitop.org/doku.php?id=2_4_0:oql:start

[38] Komponenty a ovládací prvky [online]. [Cited 2017-12-10]. Dostupné z: https://doc.nette.org/cs/2.4/components

[39] DI: Tvorba rozšíření. [Cited 2018-01-06]. Dostupné z: https://doc.nette.org/cs/2.4/di-extensions

[40] Dependency Injection [online]. [Cited 2017-12-05]. Dostupné z: https://doc.nette.org/cs/2.4/dependency-injection

[41] Malý, M.: REST: architektura pro webové API [online]. [Cited 2017-12-24]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

Příloha **A**

Slovník

AJAX Asynchronous JavaScript And XML je technologií umožňující asynchronní komunikaci se vzdálenými servery. Obvykle se využívá k aktualizaci konkrétní HTML stránky daty ze serveru..

CSS Cascading Style Sheets je jazykem umožňující ovlivnit způsob vykreslování elementů HTML souborů.

DIC Dependency injection container, „DI kontejner označujeme prvotní továrnu, která vytváří všechny objekty nutné pro spuštění a chod aplikace.“[40].

ERP Enterprise Resource Planning, česky plánování podnikových zdrojů..

fork V open-source světě se jedná o rozdvojení projektů, obvykle kvůli nemožnosti provést změny v originálním projektu..

framework V kontextu informačních technologií se jedná o sadu nástrojů a pravidel usnadňující a popřípadě automatizující některé opakované činnosti..

HTML HyperText Markup Language je značkovací jazyk primárně používaný k tvorbě webových stránek a uživatelských rozhraní webových aplikací..

hypervizor Software určený pro spouštění virtuálních počítačů, často se tak nazývá i samotný fyzický počítač na kterém tento software běží. Obvykle se totiž jedná o jedinou úlohu takového počítače..

JSON JavaScript Object Notation, jedná se o metodu zápisu dat, často použitou pro REST API..

open-source Open-source je program s volně dostupným zdrojovým kódem a licencí umožňující jeho úpravy..

playbook V kontextu Ansible se jedná soubory ve specifickém programovacím jazyce předepisujícím vykonávané úkony..

REST API Representational State Transfer Application Programming Interface, jedná se o metodu práce s daty pomocí HTTP protokolu.[41].

SLA Service level agreement je smlouvou mezi poskytovatel služby a jejím zákazníkem definující rozsah a kvalitu poskytované služby..

URI Uniform Resource Identifier je řetězec znaků určité předem dané struktury sloužící k identifikaci zdroje informací..

Příloha B

Seznam použitých zkratek

GUI Graphical user interface.

OS Operační systém.

XML Extensible markup language.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `preinstaled.zip` ..... archiv obsahující aplikaci, moduly a knihovny
- `thesis.pdf` ..... text práce ve formátu PDF
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `main` ..... hlavní projekt
    - `modules` ..... použité moduly
    - `ansible` ..... použité Ansible skripty
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX