

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Mobilní správa Windows serveru v prostředí KIV

Plzeň, 2016

Libor Váchal

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2016

Libor Váchal

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Ladislavu Pešíčkovi za rady a trpělivost při vedení této práce.

Abstract

This thesis is focused on management of Windows Server operating system with PowerShell using mobile applications for Android. First, PowerShell technology is explored, then activities suited to management are selected. Afterwards, a system is designed and implemented and finally checked for functionality.

Abstrakt

Tato práce se zaměřuje na správu systému operačního systému Windows Server pomocí PowerShellu s využitím mobilní aplikace pro systém Android. Nejdříve jsou prozkoumány možnosti PowerShellu, dále vybrány činnosti vhodné pro správu. Následně je navržen systém, který je implementován a nakonec je ověřena jeho funkcionality.

Obsah

1	Úvod.....	7
2	Problematika správy	8
2.1	Motivace	8
2.2	Přínosy	8
2.3	Cíle.....	8
3	PowerShell.....	10
3.1	Historie.....	10
3.2	Možnosti	12
3.3	Prostředí	13
3.4	Cmdlety.....	16
3.5	Aliasů	17
3.6	Roura.....	18
3.7	Proměnné	19
3.8	Operátory	22
3.9	Skripty.....	24
3.10	Profily	28
3.11	Moduly.....	28
3.12	Plánování	30
3.13	Vzdálená správa.....	31
3.14	Rozhraní WMI	35
3.15	Model CIM	37
3.16	Integrace.....	39
4	Výběr činností administrátora	41
4.1	Active Directory	41
4.2	Úložiště	42
4.3	Služby a procesy	42
5	Požadavky na systém	43
5.1	Účel systému.....	43
5.2	Rozsah systému.....	44
5.3	Uživatelé systému	44
5.4	Omezující podmínky.....	44
5.5	Autorizace	44
5.6	Funkčnost mobilní aplikace	45
5.7	Funkčnost serverové části	50
6	Návrh systému	51
6.1	Technologie	51
6.2	Serverová část	53
6.3	Klientská část.....	55
7	Systém MoWin.....	57
7.1	MoWin – Klient.....	57
7.2	MoWin – Server.....	63
7.3	Powershell skripty.....	68
8	Testování	72
8.1	Testovací případy – mobilní aplikace	73

8.2	Testovací případy – webové rozhraní	76
8.3	Testovací případy – skripty.....	77
9	Možnosti rozšíření.....	80
10	Závěr.....	81
	Seznam zkratk	82
	Literatura	83
	Přílohy.....	87
A	Popis webového API.....	88
B	Obsah CD.....	92
C	Instalační příručka.....	93
C.1	Instalace mobilní aplikace.....	93
C.2	Nasazení serverové aplikace	93
D	Uživatelská příručka	96
D.1	Android aplikace.....	96
D.2	Webová aplikace.....	104

1 Úvod

Cílem této práce je prozkoumání možností správy Windows Serveru pomocí PowerShellu a následné návržení systému, který umožní tuto správu provádět na mobilním zařízení se systémem Android.

Rozšíření mobilních zařízení společně s pokrytím mobilním internetem umožnilo v posledních letech čím dál většímu počtu lidí být „on-line“ téměř kdykoliv a kdekoliv. Mezi uživatele těchto zařízení patří i administrátoři, jejichž pracovní náplní je starat se o správný chod svěřeného systému a to ideálně dvacet čtyři hodin denně. Právě jim by měl navrhovaný systém alespoň částečně usnadnit práci.

Pomocí PowerShellu lze automatizovat rutinní činnosti prováděné při správě Windows Serveru vytvářením skriptů. Propojením tohoto způsobu automatizované správy serveru s mobilním zařízením by pak vznikl systém, umožňující vzdáleně provádět různé akce na spravovaném serveru ať už se jedná o zobrazení spuštěných procesů nebo vytvoření nového uživatelského účtu.

Tato práce si neklade za cíl přenést na mobilní zařízení veškeré možnosti správy Windows Serveru, ale vytvořit systém, který umožní uživateli mobilní aplikace provádět vybrané činnosti správy s budoucí možností snadného rozšíření o další činnosti.

2 Problematika správy

V této kapitole budou popsány typické problémy, které vznikají při správě serveru, z pohledu administrátora. V podkapitolách bude vysvětleno proč tyto problémy řešit, co řešení přinese a cíle, kterých by se mělo dosáhnout.

2.1 Motivace

Správa systému obvykle klade na administrátory velké časové nároky a vyžaduje značnou míru flexibility. Požadavky od uživatelů a klientů, ale také poruchy, se v systému mohou vyskytnout v období, kdy je administrátor není schopen vyřešit (je na cestě, nemá u sebe notebook, apod.).

S výpadkem služeb, ať už se jedná o mailový nebo webový server, ztrácí jejich uživatelé možnost rutinního fungování. Ve výsledku to může vést až k ohrožení provozu celé organizace. Výpadky a poruchy se nedají eliminovat úplně. Je ale možné zvýšit operativnost správců a řešit tak problémy rychleji. Kontrola a správa účtů, procesů, služeb, úložišť a další jsou úkony, které správce systému řeší denně. Umožnění provádět tyto činnosti flexibilně, zvyšuje spolehlivost systému a zároveň usnadňuje práci administrátora.

Motivací je tedy poskytnout nástroj, který sníží časovou prodlevu mezi vznikem události a začátkem jejího řešení. Díky dostupnosti a rozšíření mobilního internetu se jako ideální nástroj jeví mobilní aplikace.

2.2 Přínosy

Jednoznačným přínosem je zkrácení doby nutné k vyřešení vzniklých problémů (ve smyslu dodání nástroje administrátorům, pomocí kterého můžou problém řešit). Dalším přínosem je zvýšení flexibility a operativnosti správců. Podaří-li se vhodně navrhnout uživatelské prostředí výsledné aplikace tak i zpříjemnění práce administrátora.

2.3 Cíle

Cílem této práce je navrhnout a vytvořit systém, který umožní administrátorům pomocí mobilní aplikace vzdáleně spravovat vybrané činnosti Windows Server. Systém by měl být bezpečný a snadno použitelný. Zároveň by měl minimalizovat množství přenášených

dat. Také by měl být snadno rozšiřitelný, aby bylo možné v budoucnu přidávat další funkcionalitu a celkově tak rozšiřovat možnosti správy.

3 PowerShell

Windows PowerShell je platforma od firmy Microsoft, která umožňuje pokrýt a automatizovat různé aspekty správy systému Windows. Je vysoko-úrovňový, úkolově orientovaný a zaměřený na administraci [1].

Základní komponenty platformy jsou [1]:

- Shell.
- Skriptovací jazyk.

Některé vlastnosti PowerShellu [2]:

- Postavený na technologii .NET Framework¹.
- Rozumí a pracuje s objekty, příkladem může být koncept objektové roury.
- Orientace na produkci – umožnění administrátorům zvládat rozsáhlá produkční prostředí kde je kladen důraz na bezpečnost, robustnost a škálovatelnost.
- Zaměření na Windows a Windows aplikace – fungování na všech podporovaných verzích systému Windows.

Jádrem platformy je především shell [2]. Při práci se shellem se jedná o podobný koncept známý z UNIX systémů (Bourne shell, Bourne again shell, C shell, apod.). PowerShell obsahuje interpret skriptovacího jazyka, který čte příkazy zadávané uživatelem z příkazové řádky nebo ze souboru a provádí je.

Tato kapitola se po lehkém přiblížení historie a možností, bude věnovat popisu základních i pokročilejších prvků PowerShellu. Bude představeno prostředí, ve kterém administrace pomocí PowerShellu probíhá a ukázány typické postupy pro dosažení určité akce. Také budou přiblíženy vybrané komponenty, se kterými se běžně pracuje. Dále budou zmíněny možnosti plánování a vzdálené správy.

3.1 Historie

Od dob MS-DOS obsahovala každá verze systému svůj shell. Shell obsahoval vlastní skriptovací jazyk, který se mohl používat ke správě a k automatizaci různých činností. V systémech založených na MS-DOS a v rodině systémů Windows 9x to byl

¹ .NET Framework – platforma od společnosti Microsoft pro vývoj mobilních, webových i desktopových aplikací.

command.com. V systémech z rodiny Windows NT pak *cmd.exe*. Tyto shelly, ale neposkytovaly potřebnou funkcionalitu pro vytváření komplexních skriptů a neumožňovaly tak automatizovat složitější činnosti [1].

V roce 2002 začal Microsoft pracovat na nové platformě s názvem Monad [3]. Cílem bylo poskytnout nástroj orientovaný na administraci, který by umožňoval pomocí příkazů automatizovat správu, což je doménou skriptovacích jazyků [1]. Součástí této platformy byl Monad Shell (MSH). MSH bylo nové objektové skriptovací prostředí určené pro operační systémy Windows. O 4 roky později 25. dubna 2006 byl MSH přejmenován na PowerShell a byla vydána první verze PowerShell 1.0. Po tomto datu se stal součástí všech dalších verzí systému Windows. V době psaní této práce je poslední verze PowerShell 5.0, přehled dalších verzí viz. Tabulka 3.1.1.

PowerShell Verze	Vydáno	Výchozí verze Windows	Dostupný ve verzích Win.
PowerShell 1.0	11/2006	Windows Server 2008	Windows XP SP2 Windows XP SP3 Windows Server 2003 SP1 Windows Server 2003 SP2 Windows Server 2003 R2 Windows Vista Windows Vista SP2
PowerShell 2.0	10/2009	Windows 7 Windows Server 2008 R2	Windows XP SP3 Windows Server 2003 SP2 Windows Vista SP1 Windows Vista SP2 Windows Server 2008 SP1 Windows Server 2008 SP2
PowerShell 3.0	9/2012	Windows 8 Windows Server 2012	Windows 7 SP1 Windows Server 2008 SP2 Windows Server 2008 R2 SP1
PowerShell 4.0	10/2013	Windows 8.1 Windows Server 2012 R2	Windows 7 SP1 Windows Server 2008 R2 SP1 Windows Server 2012
PowerShell 5.0	5/2014	Windows 10	Windows 8.1 Windows Server 2012 R2

Tabulka 3.1.1: Historie PowerShell verzí

3.2 Možnosti

V UNIX systémech je jedním ze základních konceptů propojování výstupu (stdout) jednoho příkazu se vstupem (stdin) dalšího příkazu pomocí roury [4]. PowerShell používá podobný koncept s tím, že výstupem i vstupem je objekt nebo množina objektů. Výstup (objekt) jednoho příkazu může být převeden na vstup dalšího příkazu bez nutnosti formátování nebo manipulace se vstupem/výstupem.

PowerShell umožňuje pomocí příkazů tzv. „cmdletů“ provádět běžné administrátorské činnosti jako jsou správa služeb, procesů, registrů, systémových logů. Je možné pracovat s COM² i WMI³. Platforma nabízí možnost integrovat PowerShell do jiného programu. Poskytuje všechny elementy, které se očekávají od systémového jazyka. Proměnné, cykly, datové struktury, I/O operace a další. Také poskytuje kompletní přístup k .NET Frameworku s možnostmi jako jsou náhrávání .NET tříd, instanciování objektů, získávání metadat z objektů a to jak lokálně tak i vzdáleně [5].

Dalšími vlastnostmi, které stojí za to zmínit, jsou integrace se službami a technologiemi SQL Server⁴, IIS⁵, Hyper-V⁶, Microsoft Exchange⁷, SharePoint⁸, Server Manager⁹. Je integrován do nástrojů od společností jako jsou Intel, Cisco, Citrix, Red Hat apod. Existuje i početná komunita, která vytváří blogy, podcasty, repozitáře skriptů a fóra [5].

Jednou z vlastností je příkazově orientovaná navigace systémem. To například umožňuje procházet systémové registry stejným způsobem jako souborový systém. Viz. následující příkaz (Kód 3.2.1) pro zobrazení záznamu v registru pro program GitForWindows:

```
PS HKLM:\SOFTWARE\GitForWindows> Get-ItemProperty .\  
CurrentVersion : 2.7.0  
InstallPath    : C:\Program Files\Git  
LibexecPath    : C:\Program Files\Git\mingw64\libexec\git-core  
...
```

Kód 3.2.1: Průchod registry - zobrazení záznamu

² COM - Microsoft Component Object Model je platformově nezávislý, distribuovaný, objektově orientovaný systém od společnosti pro vytváření binárních komponent které mohou komunikovat [25]

³ WMI - Windows Management Instrumentation je nástroj, který umožňuje kompletní správu systému

⁴ SQL Server - Databázový systém od společnosti Microsoft

⁵ IIS - Webový server od společnosti Microsoft (z angl. Internet Information Services)

⁶ Hyper-V - Virtualizační platforma od společnosti Microsoft

⁷ Microsoft Exchange - Systém pro komunikace a její správu

⁸ SharePoint - Systém pro spolupráci od společnosti Microsoft

⁹ Server Manager - Systém pro správu serveru od společnosti Microsoft

3.3 Prostředí

PowerShell engine je soubor tříd technologie .NET Framework uložených s příponou *dll*. Pro komunikaci s těmito třídami je potřeba hostitel (host), který engine načte, a poskytne uživateli prostředí, pomocí něhož umožní s engine pracovat [6]. Nejčastěji se jedná o konzoli, ale může to být například i aplikace v ASP.NET nebo aplikace s GUI ovládaná dotykem.

Do systému Windows od verze 7 je standardně integrována PowerShell konzole a její nadstavba PowerShell ISE. Je možné používat i nástroje třetích stran, z nichž některé jsou placené (Primal Script) a některé volně šiřitelné (Admin Script Editor, PowerGUI) [7]. Také existují rozšíření a pluginy pro nejpoužívanější vývojová prostředí (Visual Studio, IntelliJ, Eclipse) často s možností integrace terminálu přímo do IDE¹⁰.

3.3.1 Konzole

Jak bylo zmíněno v kapitole 3.1, je PowerShell součástí distribucí Windows. Je dostupný téměř ve všech distribucích a společně s tím je dostupná interaktivní konzole. Spouštění PowerShell je možné z klasické příkazové řádky zadáním příkazu *powershell* nebo vyhledáním v nabídce Start. Druhý způsob je uživatelsky příjemnější protože spouští konzoli PowerShellu, u prvního způsobu zůstane spuštěná Windows konzole a v ní se spustí shell, který pak provádí příkazy, ale neposkytuje přidanou funkcionalitu PowerShell konzole jako je našeptávání apod.

3.3.2 PowerShell ISE

Jedná se o integrované skriptovací prostředí, které je nadstavbou klasické PowerShell konzole. Obsahuje konzoli, editor skriptů, debugger, podokno command pro snadné dohledávání cmdletů, vylepšenou funkci *tab completion*¹¹ a další funkce. Prostředí je v systémech Windows dostupné od verze PowerShell 2.0.

¹⁰ IDE – vývojové prostředí (z anglického Integrated development environment)

¹¹ Tab Completion – inteligentní doplňování příkazu, parametrů nebo hodnot po zadání prvních několika znaků

3.3.3 Bezpečnost

Uživatel PowerShellu může provádět pouze takové akce, ke kterým má oprávnění účet, pod kterým je přihlášen do systému Windows. Pokud tedy uživatel nemá právo k vytvoření nového konta v Active Directory, nemůže tuto akci provést ani po spuštění PowerShell konzole a provedení příslušného příkazu. PowerShell tak dodržuje bezpečností omezení systému Windows.

3.3.4 Spouštění skriptů

Jedním z bezpečnostních mechanismů je nutnost zadání cesty při spouštění skriptů a to i pokud se nacházíme v aktuálním adresáři. Například při zadání příkazu *dir*, PowerShell zjistí, zda existuje příkaz s takovým jménem, pokud ne, tak zda existuje alias s takovým jménem a skončí. Neprohledává aktuální adresář, to znamená, že pokud by v aktuálním adresáři existoval skript s názvem *dir*, tak by nebyl spuštěn. Spuštění skriptu *dir.ps1* (příponu není nutné uvádět) v aktuálním adresáři by vypadalo následovně (Kód 3.3.1):

```
PS C:\Tools> .\dir
```

Kód 3.3.1: Spuštění skriptu *dir* v aktuálním adresáři

Tento mechanismus tedy zabraňuje nechtěnému spuštění skriptu v aktuálním adresáři, pojmenovaném stejně jako některý z PowerShell příkazů.

Dalším mechanismem pro zabránění nechtěnému spuštění skriptu je výchozí otevírání skriptů v některém z textových editorů nainstalovaných v systému (např. Notepad). Skript s příponou *ps1* není nijak asociován s PowerShell.exe ale pouze s textovým editorem, takže pokud uživatel obdrží například od útočnicka v příloze mailu nebezpečný skript a pokusí se jej otevřít, otevře se skript pouze v textovém editoru.

U cmdletů, které nějakým způsobem mění stav systému (např. cmdlety začínající *Set-*, *Remove-*, *Disable-*), je možné použít atribut *WhatIf*, který zamezí spuštění, a pouze vypíše, co by daný cmdlet provedl. Dalším atributem je *Confirm*, který si při použití vyžádá potvrzení od uživatele viz.Kód 3.3.2:

```
PS C:\> Restart-Service Dhcp -WhatIf
What if: Performing the operation "Restart-Service" on target
"Klient DHCP (Dhcp)".
PS C:\> Restart-Service Dhcp -Confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Restart-Service" on target "Klient DHCP
(Dhcp)".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?]
Help (default is "Y"):
```

Kód 3.3.2: Použití atributů WhatIf a Confirm

3.3.5 Politiky spouštění

Tento koncept funguje na principu digitálního podepisování software. Podpis je šifrovaná informace připojená ke skriptu. Obsahuje identitu podepisujícího a také zaručuje, že skript nebyl od podepsání změněn. V PowerShellu existuje pět úrovní politiky spouštění skriptů [6]:

- **Restricted** – zakazuje spouštění jakýchkoliv skriptů.
- **RemoteSigned** – umožňuje spouštět všechny skripty které byly vytvořeny lokálně. Také umožňuje spouštět skripty, vytvořené na jiné stanici, které ale musí mít digitální podpis vytvořený pomocí důvěryhodného certifikátu. Po podpisu již skript nemůže být změněn.
- **All Signed** – podobné jako RemoteSigned s tím rozdílem, že musí být podepsané i lokálně vytvořené skripty.
- **Unrestricted** – spouští všechny skripty.
- **Bypass** – kompletně vypíná celou politiku spouštění (použití například při vývoji aplikace, která integruje PowerShell uvnitř a bezpečnost řeší na úrovni aplikace).

Zjištění nastavené politiky umožňuje cmdlet *Get-ExecutionPolicy*. Změnu aktuální politiky provede cmdlet *Set-ExecutionPolicy*. Pro provedení tohoto příkazu je potřeba mít administrátorská oprávnění.

Z toho přístupu lze vyčíst, že zabezpečuje pouze to, aby nebylo možné omylem spustit skript, který by neměl být spuštěn. Tato opatření ovšem nezabrání uživateli, spustit například škodlivý kód z nepodepsaného skriptu (stačí skript otevřít, zkopírovat obsah do PowerShell konzole a spustit).

3.3.6 Shrnutí

Zmíněné bezpečnostní prvky nezaručují ani neznemožňují uživateli provádět určitou akci, ale pouze pomáhají v tom, aby takovou akci neprovedl náhodou nebo omylem. Pokud bude mít uživatel oprávnění (v rámci systému Windows) spouštět potenciálně škodlivé akce, bezpečnostní politiky PowerShellu mu v tom nemohou zabránit.

3.4 Cmdlety

Cmdlet je příkaz, který po spuštění vykoná nějakou akci. Implementačně se jedná o třídu napsanou v jazyce C# (nebo jiném jazyce podporovaném v .NET platformě), která provádí určitou činnost. Je možné vytvářet vlastní třídy a vlastní cmdlety a ty pak spouštět v prostředí PowerShellu [8].

Pro cmdlety existuje konvence jejich pojmenování, která má za cíl dát na první pohled představu o tom, co daný cmdlet dělá. Jedná se o tvar *<akce>-<položka>*. Akcí bývají často slovesa: *Get, Set, Find, Search, Start*, ale nemusí to platit vždy, např. *New*. Položka je nejčastěji podstatné jméno a značí entitu, se kterou se bude daná akce provádět např. *Host, Item, Path, Json* nebo *PSDrive*.

Příklad spuštění a výstup cmdletu *Get-Culture* pro zjištění informací o jazykovém nastavení systému pak ilustruje Kód 3.4.1:

```
PS C:\> Get-Culture

LCID          Name          DisplayName
----          -
1029          cs-CZ         Čeština (Česká republika)
```

Kód 3.4.1: Ukázka cmdletu Get-Culture

Syntaxe příkazu s parametry: *<cmdlet> [-parametr] [hodnota]*

Některé parametry jsou poziční, takže může být vynecháno jméno parametru a zadána pouze hodnota. Zda je parametr daného cmdletu poziční nebo ne, se zjistí z nápovědy pomocí příkazu *Get-Help <cmdlet>*. Zároveň některé parametry fungují bez zadané hodnoty. O konkrétní syntaxi daného příkazu je vždy možné se přesvědčit pomocí nápovědy.

Ukázka příkazu s parametrem *Name* pro zjištění informací o disku *C* viz. Kód 3.4.2:

```
PS C:\> Get-PSDrive -Name C

Name           Used (GB)   Free (GB) Provider      Root
-----
C              188,14     2,04  FileSystem   C:\
```

Kód 3.4.2: Ukázka použití parametru

Příkaz bez parametru pouze s hodnotou pro zjištění informací o disku *D* viz. Kód 3.4.3:

```
PS C:\> Get-PSDrive D

Name           Used (GB)   Free (GB) Provider      Root
-----
D              250,97     23,54  FileSystem   D:\
```

Kód 3.4.3: Ukázka použití pozičního parametru

Cmdlety pro základní operace vytváří vývojový tým PowerShellu, na více specifických se podílí i další vývojové týmy Microsoftu. S každou novou verzí roste i počet cmdletů. Ve verzi 4.0 je jich kolem 3600 [9].

3.5 Aliasy

Pro standartní, ale i vlastní cmdlety je možné používat aliasy, což jsou vlastně jen jiné názvy pro daný cmdlet. Pro některé cmdlety již aliasy existují, zároveň je možné definovat i vlastní viz. Tabulka 3.5.1.

Alias	Cmdlet
gdr	Get-PSDrive
cat	Get-Content
fl	Format-List
kill	Stop-Process
%	Foreach-Object

Tabulka 3.5.1: Aliasy cmdletů

K definici vlastního aliasu se použije cmdlet *Set-Alias*. Kód 3.5.1 ilustruje vytvoření aliasu *l* pro výpis adresáře:

```
PS C:\> Set-Alias l Get-ChildItem
```

Kód 3.5.1: Vytvoření aliasu

Pokud je potřeba vytvořit alias pro příkaz, který má být proveden s parametrem, definuje se alias pomocí funkce. Příklad (Kód 3.5.2) ilustruje vytvoření aliasu *ln* pro výpis adresáře pouze s názvy souborů a adresářů:

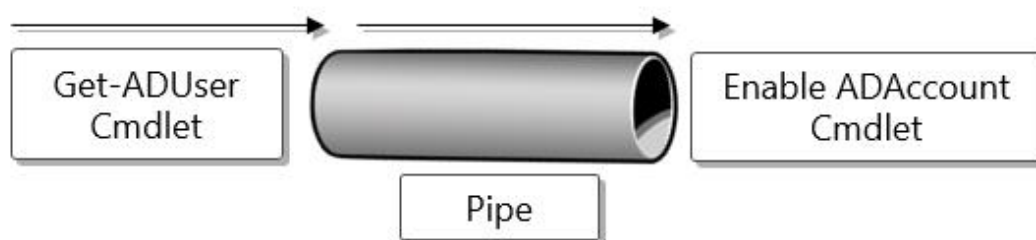
```
PS C:\> function ln {  
    Get-ChildItem -Name  
}
```

Kód 3.5.2: Vytvoření aliasu pomocí funkce

Alias poskytl zrychlení práce, ovšem při používání velkého množství aliasů v delších sledech příkazů může dojít k nepřehlednostem. K používání aliasů je nutné přistupovat rozumně.

3.6 Roura

Jedná se o koncept řetězení výstupů a vstupů jednotlivých příkazů pro dosažení určité akce nebo operace. Pro řetězení se používá znak „|“ označovaný jako roura (z anglického pipe). Jak už bylo zmíněno, při řetězení se předávají objekty. Napojovaný cmdlet tak může pracovat s atributy, metodami nebo uloženými událostmi u vstupního objektu, a plně využívat potenciál tohoto přístupu. Obrázek 3.6.1 pro názornost zobrazuje použití roury pro aktivaci uživatelského účtu ve Windows Active Directory [10].



Obrázek 3.6.1: Ukázka použití roury na Cmdlety pro vybrání a aktivaci uživatele v Active Directory.

Zřetězení příkazů v konzoli by pak vypadalo následovně (Kód 3.6.1):

```
PS C:\> Get-ADUser frantisek | Enable-ADAccount
```

Kód 3.6.1: Zřetězení příkazu

Při řetězení objektů je nutné konkretizovat jak napojit vstupní objekt (i více objektů – záleží na výstupu původního cmdletu) na další cmdlet. Tento proces se nazývá „pipeline parameter binding“ a v PowerShellu existují dvě techniky, kterými napojení realizuje [6]:

- **ByValue** – PowerShell zjistí, které parametry cmdletu jsou schopné akceptovat vstup *ByValue* (to definuje tvůrce cmdletu a je to popsáno v nápovědě ke cmdletu) a pak dle typu vstupního objektu a typu parametru vybere vyhovující parametr.
- **ByPropertyName** – Tato technika se provádí, pokud selže *ByValue*. PowerShell zjistí, které parametry cmdletu jsou schopné akceptovat vstup *ByPropertyName* (to opět definuje tvůrce cmdletu a je to popsáno v nápovědě ke cmdletu). Na tyto parametry jsou pak odeslány hodnoty parametrů ze vstupního objektu, které mají stejné jméno.

V PowerShellu je možné používat externí příkazy jako např. *ipconfig*. Zpracování těchto příkazů je odlišné, výstupem totiž nejsou objekty, ale pouze text. Tento textový výstup se dá dále zpracovávat cmdlety pro zpracování řetězců např. *Select-String*. Je možné použít i opačný proces kdy se při zřetězení cmdletu a externího příkazu převede výstupní objekt cmdletu na řetězec a ten se pošle na standartní vstup externího příkazu. Toto se používá ale pouze zřídka [6].

3.7 Proměnné

Proměnné, stejně jako v jiných skriptovacích jazycích, slouží k uchování informace. Proměnné je pak možné v průběhu skriptu používat dle potřeby. PowerShell používá stejnou syntaxi jako jazyk PHP¹², kdy je proměnná uvozená znakem “\$” za kterým následuje její název. Přiřazovacím operátorem je znak „=“. Ukázka viz. Kód 3.7.1.

```
$pocet = 50
```

Kód 3.7.1: Přiřazení hodnoty do proměnné

¹² PHP - PHP: Hypertext Preprocessor – skriptovací jazyk pro vývoj webových aplikací

Proměnné není potřeba deklarovat, takže kdykoliv ve skriptu potřebujeme uložit nějaká data, zapíšeme pouze název nové proměnné a data přiřadíme. Názvy proměnných jsou case-insensitive (nezáleží na velikosti písmen). PowerShell je slabě typovaný to znamená, že není potřeba u proměnných uvádět jejich typ, ten se určí až za běhu přiřazením hodnoty [11]. Zajímavou možností je přiřazení více hodnot více proměnným, možné použití ukazuje příklad prohození hodnot dvou proměnných viz. Kód 3.7.2:

```
$a = 1
$b = 2
$a, $b = $b, $a
```

Kód 3.7.2: Prohození obsahu dvou proměnných

Předchozí zápis je možný díky tomu, že pomocí znaku čárky je vytvořeno pole. Dále je možné přistupovat k jednotlivým proměnným i celému poli viz. Kód 3.7.3:

```
PS C:\> $a
2
PS C:\> $b
1
PS C:\> $a, $b
2
1
```

Kód 3.7.3: Přístup k proměnným

Zjištění datového typu můžeme provést pomocí zavolání funkce *GetType().Name* na název proměnné viz. Kód 3.7.4:

```
PS C:\> ($a, $b).GetType().Name
Object[]
```

Kód 3.7.4: Zjištění typu proměnné

Typ proměnné je možné definovat také explicitně, příklad ukázaný v Kód 3.7.5 vytvoří proměnou typu *datetime*. Pak je možné využít vlastnost tohoto objektu a zeptat se co za den v týdnu připadá na uložený datum:

```
PS C:\> [datetime]$date = "March 12, 2016"
PS C:\> $date.DayOfWeek
Saturday
```

Kód 3.7.5: Explicitní typování proměnné

Podporovány jsou všechny datové typy dostupné v .NET [8] od klasických jako jsou *array*, *bool*, *byte*, *char*, *decimal*, *int* až po složitější jako je *timespan*, *hashtable*, *regex*, *xml*, *scriptblock*.

Pomocí cmdletu *Set-Strictmode* je možné vynutit inicializaci proměnných jak v shellu, tak ve skriptech. To může pomoci při odhalování chyb, které vznikají například překlepem v názvu proměnné [6].

3.7.1 Systémové a automatické proměnné

Výpis systémových proměnných prostředí se vyvolá příkazem *Get-Childitem env*:. Obsahuje proměnné *Path*, *TEMP*, *ProgramData*, *OS*, *HOMEDRIVE*, *USERPROFILE* a další.

Výpis automatických proměnných (označení typu proměnných vytvářených při startu konzole) se vyvolá pomocí *Get-Childitem variable*:. Obsahuje automatické proměnné, které jsou vytvořeny PowerShellem při spuštění konzole a obsahují informace o konfiguraci PowerShellu. Konfiguraci je možné těmito proměnnými měnit [11].

3.7.2 Viditelnost

Jedná se o omezení přístupu k proměnným, aliasům a funkcím. Důvodem je ochrana těchto zdrojů před úmyslnými i neúmyslnými změnami [12].

Jsou definovány 4 prostory (scope), ve kterých je proměnná viditelná:

- **Global** – reprezentuje scope konzole, proměnná je viditelná a existuje i po ukončení skriptu, který ji definoval.
- **Local** – výchozí scope, potomek local scope může číst proměnné rodiče, ale nemůže je měnit.
- **Private** - proměnná je viditelná pouze v aktuálním scope, potomek tuto proměnnou neuvidí.
- **Script** – každý skript vytváří svůj vlastní scope, který je viditelný pouze v rámci tohoto skriptu, můžou k němu přistupovat všechny funkce a části definované uvnitř skriptu.

3.8 Operátory

Operátory jsou jazykové elementy, které je možné používat v příkazech. Stejně jako většina programovacích a skriptovacích jazyků, poskytuje PowerShell různé typy operátorů. Patří mezi ně porovnávací operátory, výpis viz. Tabulka 3.8.1.

Operátor	Operace
-eq	rovná se
-ne	nerovná se
-gt	je větší
-ge	je větší nebo rovno
-lt	je menší
-le	je menší nebo rovno
-like, -notlike	testování řetězců pomocí wildcards
-match, -cmatch, -notmatch, -cnotmatch	testování řetězců pomocí regulárních výrazů
-contains, -notcontains, -in, -notin	existence/neexistence objektu v kolekci

Tabulka 3.8.1: Operátory pro porovnávání

Za zmínku stojí operátor *contains*, který se volá nad kolekcí a zjišťuje, zda je v kolekci přítomen zadaný objekt. Opačnou operaci poskytuje operátor *notcontains*. U objektu jsou pak porovnávány všechny jeho vlastnosti, jejichž hodnoty se musí rovnat.

Aritmetické operátory používají v PowerShellu standartní označení známé i z jiných jazyků (+, -, /, *, %, ++, --). Je možné používat i zkrácené verze pro aritmetickou operaci a přiřazení (+=, /=, *=). Zajímavou možností je „násobení“ řetězce viz. Kód 3.8.1:

```
PS C:\> "wow! " * 3  
wow! wow! wow!
```

Kód 3.8.1: Ukázka "násobení" řetězce

Pro práci s řetězci a poli je možné využít operátory:

- **replace** – vyhledání a nahrazení sub-řetězce v řetězci,
- **split** – převod řetězce na pole,
- **join** – převod pole na řetězec.

Zjišťování typů a konverzi umožňují operátory:

- **is** – testuje zda je objekt zadaného typu,
- **isnot** – inverzní k operaci *is*,
- **as** – přetypuje objekt na zadaný typ.

Ukázka zmíněných operátorů viz. Kód 3.8.2.

```
PS C:\> $a = "one", "two", 3
PS C:\> $a -is [array]
True
PS C:\> $a -replace 3, "three"
one
two
three
PS C:\> $a -join "-"
one-two-3
PS C:\> $a[2] -as [float]
3
```

Kód 3.8.2: Ukázka použití operátorů

Další možností je formátování řetězců operátorem *f*. Použití ilustruje Kód 3.8.3:

```
PS C:\> "{0}: {1} hodil na kostce cislo {2}" -f (Get-Date), (Get-Content Env:\USERNAME), (Get-Random -Maximum 6 -Minimum 1)
14.4.2016 22:07:40: Libor hodil na kostce cislo 2
```

Kód 3.8.3: Použití operátoru *f* pro formátování řetězce

Na levé straně od operátoru je řetězec obsahující zástupné symboly do kterých se doplní výsledky příkazů z pravé strany. Zástupné symboly jsou označeny *{0}*,*{1}*,*{2}* kdy číslo určuje index příslušného příkazu na pravé straně.

Kompletní výpis operátorů a jejich možností je dostupný v nápovědě PowerShellu v sekci *about_Operators*.

3.9 Skripty

Skript není nic jiného než posloupnost příkazů uložených do souboru, který je možné opětovně spouštět jak manuálně tak automaticky [8]. Přípona takových souborů je *ps1*. Skripty mohou být velmi jednoduché - jedno nebo víceřádkový příkaz uložený do souboru, ale i složitější, například skript pro načtení jmen uživatelů z textového souboru, vytvoření jejich kont v Active Directory a odeslání mailu s přihlašovacími údaji.

Skripty obecně obsahují následující části, přičemž nemusí obsahovat všechny [1]:

- **Aplikační logika** – řeší konkrétní problém, kvůli kterému skript vzniká.
- **Odchytávání chyb** – ošetření situací, kdy provádění nějakého příkazu vyvolá chybu.
- **Validace vstupu** – ošetření vstupních dat, kdykoliv jsou nějaká zadávána uživatelem nebo jako vstupní parametry skriptu.
- **Logování** – ukládání informací o průběhu skriptu do souboru.
- **Konstrukce PowerShell jazyka** – pomocí konstrukcí se dává celý skript dohromady tak, aby prováděl to, k čemu byl vytvořen.

3.9.1 Podmínky a cykly

Podmínky umožňují větvení programu na základě splnění (resp. nesplnění) daného logického výrazu. Ve výrazech se používají logické operátory zmíněné v kapitole 3.8. Používají se řídicí konstrukce *if*, *else*, *switch*. Zápis je podobný jako v jiných programovacích jazycích viz. Kód 3.9.1.

```
if ($i -lt 10) {
    "Smaller than 10"
}
else {
    "10 or greater"
}

switch ($i) {
    1 {"one"}
    5 {"five"}
    10 {"ten"}
    default { "other" }
}
```

Kód 3.9.1: Ukázka použití podmínek pro větvení

Cykly slouží k opakovanému provádění určité akce za splnění (resp. nesplnění) daného logického výrazu. Zápis je opět velmi podobný jako u jiných programovacích jazyků. Dostupné typy cyklů a řídicí příkazy viz. Kód 3.9.2:

```
while(condition) { ... }
do { ... } while (condition)
do { ... } until (condition)
for ($i = 0; condition; $i++) {...}
foreach ($var in $collection) {...}

break
continue
return
exit
```

Kód 3.9.2: Cykly a řídicí příkazy

3.9.2 Funkce

Pro opakované používání bloku kódu nebo kvůli oddělení určitých logických celků v rámci skriptu, je možné definovat funkce. Definice funkcí musí předcházet jejímu použití, není tak možné na prvním řádku skriptu volat funkci, která je definována nakonci [2]. Funkce je možné definovat i v prostředí konzole shellu. Ukázka definice a použití funkce pro konverzi a zaokrouhlení bytů na gigabyty viz Kód 3.9.3.

```
PS C:\> function convertToGigaBytes([long] $bytes, [int] $places)
{
    $result = [math]::round($bytes / 1GB, $places)
    $result
}

PS C:\> convertToGigaBytes 45652347845 3
42,517
```

Kód 3.9.3: Definice a použití funkce pro konverzi bytů na gigabyty

Z příkladu uvedeném v Kód 3.9.3 je možné vidět, že při volání funkce se parametry nezadávají do kulatých závorek za název funkce, ale jsou pouze odděleny mezerou. Dalším nezvyklým znakem je absence řídicí konstrukce *return*, pro vrácení návratové hodnoty se použije jen název proměnné.

3.9.3 Výjimky

Při provádění příkazů v konzoli nebo v průběhu skriptu se mohou vyskytnout dva typy chyb:

- **Terminating** – provádění skriptu je při výskytu tohoto typu chyb ukončeno.
- **Nonterminating** – provádění skriptu pokračuje dál.

Cmdlety v PowerShellu obsahují parametr *ErrorAction*, kterým je možné nastavit co se má stát v případě výskytu *nonterminating* chyby. Toto nastavení pak funguje pouze pro jedno konkrétní provádění příkazu, globální nastavení pro všechna provádění všech příkazů se provede pomocí automatické proměnné *ErrorActionPreference*. Lze nastavit následující chování:

- **Inquire** – zeptá se uživatele, zda má skript pokračovat.
- **Continue** – zobrazí chybu a pokračuje dál.
- **SilentlyContinue** – nezobrazí chybu a pokračuje dál.
- **Stop** – zobrazí chybu a ukončí provádění.

Pomocí nepovinného parametru *ErrorVariable* lze u cmdletu nastavit proměnnou do které se vloží výjimka. Standardně se výjimky ukládají do automatické proměnné *Error*. Jedná se o pole, které obsahuje výjimky od té nejaktuálnější až po tu nejstarší. Výjimka, která je vytvořena při výskytu chyby je objekt, a jako objekt je uložena i v poli *Error* [6].

Do PowerShellu verze 2.0 se výjimky zpracovávaly pomocí konstrukce *trap*. Jednalo se o koncept kdy se při výskytu chyby v některém příkazu skriptu, skočilo na začátek skriptu, kde musel být definovaný blok *trap* (blok musel být definovaný předtím, než mohlo dojít k chybě). V *trap* bloku se provedly příslušné příkazy a pak se pomocí konstrukce *exit* skript ukončil nebo se pomocí *continue* pokračovalo ve skriptu dál, od dalšího příkazu za příkazem kde chyba vznikla [6].

Od verze 2.0 se zpracování výjimek provádí pomocí konstrukce *try*, *catch* a *finally*. Blok *try* obsahuje kód, ve kterém se může vyskytnout chyba. Při výskytu výjimky se spustí kód v bloku *catch*, který provede definovanou akci. *Catch* bloků může být více a každý z nich může reagovat na jiný typ výjimky. Kód v bloku *finally* (nepovinný blok) proběhne vždy, ať už se výjimka vyskytla nebo ne.

3.9.4 Skriptování s využitím objektů

Instalací modulu (více o modulech v kapitole 3.11) `PSClass` [13] je možné vnést do skriptování koncept OOP¹³. S tímto modulem je možné jednoduše a intuitivně definovat nové třídy, využívat dědičnost, polymorfismus i zapouzdření. Tříďe je možné definovat, vlastnosti, metody i konstruktor a celý koncept se tak přibližuje vysoko úrovněným jazykům jako je Java, C#, Python apod. Je potřeba zdůraznit, že se jedná jenom o simulaci tohoto konceptu, modul z principu nemůže nijak rozšiřovat architekturu PowerShellu, ale pomocí cmdletu `New-PSClass` pouze umožňuje psát skripty takovým způsobem, které se tomuto konceptu podobají. V příkladu (Kód 3.9.4) je ukázka definice jednoduché třídy `Car` použitím funkcionality modulu `PSClass`.

```
$CarClass = New-PSClass Car {  
  
    note -static ObjectCount 0  
    method -static DisplayObjectCount {  
        "$($this.ClassName) has $($this.ObjectCount) instances"  
    }  
  
    note -private Type  
    note -private Color  
  
    constructor {  
        param ($type,$color)  
        $private.Type = $type  
        $private.Color = $color  
  
        $CarClass.ObjectCount += 1  
    }  
  
    property Type -get {  
        $private.Type  
    }  
  
    property Color -get {  
        $private.Color  
    } -set {  
        param($newColor)  
        Write-Host "Repainting $($this.Class.ClassName) from  
'$($private.Color)' to '$($newColor)'"  
        $private.Color = $newColor  
    }  
  
    method -override ToString {  
        "This $($this.Type) is $($this.Color)"  
    }  
}
```

Kód 3.9.4: Ukázka definice třídy za pomoci modulu `PSClass`

¹³ OOP – objektově orientované programování

Vytvoření instance třídy a ukázka použití by vypadalo následovně (Kód 3.9.5):

```
PS C:\> $myFiat = $CarClass.New("Sedan","Green")

PS C:\> $myFiat.ToString()
This Sedan is Green

PS C:\> $myFiat.Color = "Blue"
Repainting Car from 'Green' to 'Blue'

PS C:\> $myFiat.ToString()
This Sedan is Blue

PS C:\> $CarClass.DisplayObjectCount()
Car has 1 instances
```

Kód 3.9.5: Vytvoření instance a použití třídy

3.10 Profily

Profily jsou skripty, které slouží k nastavení prostředí (např. definování aliasů, funkcí nebo načtení historie příkazů) při spuštění shellu. Při zavření konzole končí platnost všech vytvořených aliasů, funkcí, importovaných modulů, historie příkazů apod. Pokud jsou tato nastavení umístěna do profilů, je možné je pak opětovně používat bez nutnosti dalšího nastavování. Existují ve dvou typech [14]:

- **Globální pro všechny uživatele** – typicky umístěný v instalačním adresáři PowerShellu *c:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1*.
- **Uživatelské** – typicky umístěné v dokumentech *c:\User*(Username)*\documents\Windows PowerShell*.

Při spuštění shellu se spouští jak globální skript, tak i skript aktuálního uživatele. Tímto způsobem je možné přizpůsobovat prostředí dle aktuálních požadavků, jak pro všechny uživatele, tak i pro jednotlivce.

3.11 Moduly

PowerShell poskytuje dva způsoby rozšíření. Prvním je PSSnapin. PSSnapin je knihovna napsaná v některém z .NET jazyků, jedná se tedy o soubor s příponou *dll*. Před použitím v PowerShellu je potřeba PSSnapin nejdříve instalovat a registrovat v systému. Od PowerShellu verze 2.0 jsou preferovaným způsobem rozšiřování Moduly [6].

Modul je balík skriptů, funkcí a cmdletů, který po instalaci zpřístupní novou funkčnost. Idea modulů je umožnit snadnou rozšiřitelnost PowerShellu pomocí skriptů, které již někdo vytvořil. Moduly mohou být nejenom textové, ale také binární [15]. Zjištění nainstalovaných modulů provede cmdlet *Get-Module* s parametrem *ListAvailable*.

```
PS C:\Intel> Get-Module -ListAvailable | Select-Object Name,
ModuleType, Version | Format-List

Name      : CimCmdlets
ModuleType : Manifest
Version   : 1.0.0.0

Name      : ISE
ModuleType : Script
Version   : 1.0.0.0

Name      : Microsoft.PowerShell.Management
ModuleType : Manifest
Version   : 3.1.0.0

Name      : Microsoft.WSMan.Management
ModuleType : Manifest
Version   : 3.0.0.0

Name      : PSDesiredStateConfiguration
ModuleType : Binary
Version   : 1.0
```

Kód 3.11.1: Zjištění nainstalovaných modulů

V prvních verzích bylo nutné načíst moduly při každém spuštění PowerShellu. Od verze 3.0 existuje nová funkce „module autoloading“, která zajišťuje automatické načítání modulů [6]. To proběhne až při zavolání skriptu, funkce nebo cmdletu, které jsou součástí daného modulu. Cesta k modulu musí být umístěna (případně může být přidána) v proměnné prostředí *PSModulePath*, kde se nachází cesty, které autoloading používá. Pro manuální přidání modulu do aktuální relace se používá cmdlet *Import-Module*.

Při importování modulů může dojít ke konfliktům v rámci jména cmdletů a funkcí. Stejná jména totiž už v systému mohou být používána. PowerShell pak při zadání cmdletů s konfliktním jménem spustí ten, který byl importován jako poslední [6]. Konfliktům je možné se částečně vyhnout použitím parametru *Prefix* u cmdletu *Import-Module*. Jména importovaných cmdletů a funkcí z modulu jsou pak doplněna o definovaný prefix a s tímto prefixem je potřeba je i spouštět.

3.12 Plánování

Od verze 3.0 PowerShell umožňuje plánovat spuštění uložených příkazů a skriptů. Pro skript je vytvořen tzv. *job*, který se zaregistruje v plánovači úloh systému Windows. Job obsahuje *trigger*, který definuje kdy se má daný *job* spouštět a skript, který se bude spouštět. PowerShell pro tento *job* vytvoří adresář se jménem (ve výchozím nastavení umístěn ve složce `\AppData\Local\Microsoft\Windows\PowerShell\ScheduledJobs`). V tomto adresáři bude umístěn soubor s definicí *jobu* a adresář *Output*, ve kterém se po každém provedení *jobu* vytvoří podadresář s časovým označením a v něm soubor, který bude obsahovat výstup spuštěného skriptu nebo příkazu ve formátu XML [6].

Ukázka vytvoření naplánované úlohy, která každý den v 18:15 zjistí právě spuštěné procesy, by vypadala následovně (Kód 3.12.1):

```
PS C:\> $trigger = New-JobTrigger -Daily -At '18:15'

PS C:\> Register-ScheduledJob -Name DailyProcess -ScriptBlock
{Get-Process} -Trigger $trigger
```

Kód 3.12.1: Vytvoření úlohy, která denně zjišťuje spuštěné procesy

Kód 3.12.2 ilustruje použití cmdletu *Get-Job* pro vypsání informací o proběhlých úlohách:

```
PS C:\> Get-Job DailyProcess | Select-Object Id, Name, State,
PSBeginTime, PSEndTime | Format-List

Id           : 2
Name         : DailyProcess
State        : Completed
PSBeginTime  : 25.4.2016 18:15:03
PSEndTime    : 25.4.2016 18:15:07

Id           : 3
Name         : DailyProcess
State        : Completed
PSBeginTime  : 26.4.2016 18:15:01
PSEndTime    : 26.4.2016 18:15:04

Id           : 4
Name         : DailyProcess
State        : Completed
PSBeginTime  : 27.4.2016 18:15:02
PSEndTime    : 27.4.2016 18:15:05
```

Kód 3.12.2: Vypsání informací o proběhlých úlohách

Výstup proběhlé úlohy je pak možné vypsat pomocí *Receive-Job* s *id* požadované úlohy viz. Kód 3.12.3:

```
PS C:\Users\Libor> Receive-Job -id 2
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
330	28	7668	1084	130	0,86	6488	AAM Updates
333	50	169972	185296	370	21,40	7220	AcroRd32
236	17	8928	17864	110	0,16	8232	AcroRd32
76	8	1236	4124	42	0,00	1156	armsvc
160	11	2632	7688	81	0,06	1316	atieclxx

Kód 3.12.3: Zobrazení výstupu proběhlé úlohy (oříznutý výstup)

Do verze 2.0 bylo možné plánovat spouštění skriptů pomocí plánovače úloh, bylo ale nutné nastavit vlastnosti spouštění přímo v tomto nástroji. Verze 3.0 také používá plánovač úloh, ale nastavení a správu integruje přímo do prostředí PowerShellu [16].

3.13 Vzdálená správa

Do PowerShellu 2.0 bylo nutné pro vzdálené spouštění příkazů instalovat specializované balíky. Od této verze to již není nutné, a správa je velmi podobná jako při ovládání lokální stanice [8]. Standartní vzdálená komunikace pro připojení k hostitelským stanicím používá protokoly DCOM¹⁴ a RPC¹⁵. Nastavení těchto metod často vyžaduje otevření mnoha různých portů a spuštění dalších podpůrných služeb. Od verze Windows 7 existuje v systémech služba WinRM, která je implementací protokolu WS-Management a poskytuje kompatibilní metodu přístupu ke vzdáleným systémům [17]. Vzdálená správa je v PowerShellu označována jako PowerShell Remoting.

WinRM se stará o komunikaci, autentizaci i správu vzdálených připojení. Poskytuje komunikační služby nejenom pro PowerShell, ale i pro jiné aplikace. Jakmile služba obdrží nějaká data, jsou označena specifickým tagem (např. tag pro PowerShell) určujícím, které aplikaci mají být adresována. Pak zajistí doručení dat k příslušné aplikaci i zpracování odpovědí a výsledků, které mají být poslány zpět [6].

¹⁴ DCOM – protokol pro komunikaci softwarových komponent (z anglického Distributed Component Object Model)

¹⁵ RPC – technologie pro spouštění kódů na vzdálené stanici (z anglického Remote Procedure Call)

WinRM operuje na dvou portech 5985 pro nezabezpečený přenos přes HTTP a na 5986 pro zabezpečený HTTPS, které se dají změnit. Služba obsahuje množství nastavení, které umožňují konfigurovat kdo, co a jak může používat [6].

V PowerShellu je možné u některých cmdletů nalézt parametr *computerName*, který je možné použít pro spuštění cmdletu na vzdálené stanici. Tyto cmdlety nejčastěji používají zmíněné proprietární protokoly DCOM a RPC [8]. Dále existují cmdlety pro vzdálenou správu využívající WinRM, jedná se o *Invoke-Command* a jakékoliv cmdlety obsahující v názvu *PSSession*.

Pro používání vzdálené správy je nutné mít na klientské i hostitelské stanici následující komponenty (uvedené jsou minimální verze):

1. Microsoft .NET Framework 2.0
2. Windows PowerShell 2.0
3. WinRM 2.0 service

Aby bylo možné spravovat stanici na dálku, je nutné toto na vzdálené stanici povolit. Windows Server 2012 má tuto možnost povolenou už při instalaci, u jiných verzí se vzdálená správa povolí cmdletem *Enable-PSRemoting* [18].

3.13.1 Autorizace

WinRM pracuje na dvou úrovních přihlašování, uživatelská úroveň a úroveň stroje [6].

- **Uživatelská** – zahrnuje delegaci pověření aktuálního přihlášení do Windows na vzdálený stroj. Vzdálená stanice pak převezme přihlášení a umožní vykonávat jakékoli akce, ke kterým má daný uživatel oprávnění.
- **Strojová** – vyžaduje přihlášení ke vzdálené stanici a nastavení stanice jako důvěryhodné (úpravou konfigurace WS-Management). Výhodou tohoto přístupu může být bezpečnost, stanice se totiž připojí pouze ke vzdáleným stanicím definovaným v konfiguraci.

3.13.2 Session

Pro připojení ke vzdálené stanici se používá cmdlet *Enter-PSSession*. Tento příkaz vytvoří relaci mezi klientskou a hostitelskou stanicí. Příkazy se pak do konzole zadávají stejně jako při práci s lokální stanicí. Před připojením je možné zjistit, zda je na vzdálené stanici spuštěna služba WinRM, k tomu slouží cmdlet *Test-WSMan*. V příkladu (Kód 3.13.1) je ukázáno využití *Enter-PSSession* pro připojení ke stanici s názvem *MYSERVER2012* a účtu *Administrator* v doméně *KIV*. Po připojení je spuštěn příkaz pro získání několika informací o systému:

```
PS C:\> Enter-PSSession MYSERVER2012 -Credential
KIV\Administrator

[MYSERVER2012]: PS C:\> Get-CimInstance Win32_OperatingSystem |
Select-Object Caption, InstallDate, OSArchitecture, BootDevice,
CSName | FL

Caption           : Microsoft Windows Server 2012 R2 Standard
InstallDate      : 29. 10. 2015 14:29:41
OSArchitecture   : 64-bit
BootDevice       : \Device\HarddiskVolume1
CSName           : MYSERVER2012
```

Kód 3.13.1: Připojení ke vzdálené stanici pomocí *Enter-PSSession*

Heslo ke vzdálenému účtu, se zadává do dialogu, který se otevře po spuštění cmdletu *Enter-PSSession*.

V příkladu (Kód 3.13.1) je vidět změna kontextu v konzoli, kdy se po připojení na vzdálenou stanici přidá na začátek řádku název vzdálené stanice. V tuto chvíle se všechny zadané příkazy provádějí na vzdálené stanici. Odpojení provede cmdlet *Exit-PSSession*.

Tímto způsobem lze na vzdálené stanici spouštět téměř všechno s několika výjimkami [6]:

- Nelze spustit příkazy, které vytvářejí vzdálené připojení.
- Nelze spustit příkazy, které spouští aplikace s grafickým rozhraním.
- Nelze spustit program, který obsahuje svůj vlastní shell.
- Skripty, lze spouštět pouze pokud to dovoluje Execution policy (viz.kapitola 3.3.5) na vzdálené stanici.

- Pro připojení je nutné zadat název stanice. IP adresu nebo DNS¹⁶ jméno je možné použít, pouze pokud je umístěno v seznamu důvěryhodných připojení.

3.13.3 Invoke-Command

Pomocí cmdletu *Invoke-Command* je možné posílat jeden nebo více příkazů přes síť na jednu nebo více stanic. Vzdálená stanice provede příkazy za použití vlastních zdrojů (tzn. příkazy musí na vzdálené stanici existovat). Výsledky těchto příkazů jsou objekty, které jsou serializovány do formátu XML. V tomto formátu jsou poslány přes síť na původní stanici, která objekty deserializuje a vloží do roury (výstup je tak možné dál lokálně zpracovávat) [6].

Příkaz (Kód 3.13.2: Zjištění informací o vzdálené stanici pomocí Invoke-Command), pro zjištění informací o systému (stejných jako v předchozí kapitole) by vypadal následovně:

```
PS C:\> Invoke-Command -ComputerName MYSERVER2012 -Credential
KIV\Administrator -ScriptBlock {Get-CimInstance
Win32_OperatingSystem | Select-Object Caption, InstallDate,
OSArchitecture, BootDevice, CSName | FL}

Caption           : Microsoft Windows Server 2012 R2 Standard
InstallDate      : 29. 10. 2015 14:29:41
OSArchitecture   : 64-bit
BootDevice       : \Device\HarddiskVolume1
CSName           : MYSERVER2012
```

Kód 3.13.2: Zjištění informací o vzdálené stanici pomocí Invoke-Command

V tomto případě se není nutné připojovat a odpojovat pomocí relace, ale vše je provedeno v rámci jednoho příkazu. V příkazu je důležitý parametr *ScriptBlock*, který v těle obsahuje prováděné příkazy. Je možné spustit i uložený skript, v tom případě by se místo parametru *ScriptBlock* použil parametr *FilePath* s cestou k příslušnému skriptu. Heslo ke vzdálenému účtu se opět zadává až po spuštění cmdletu. Také je možné vidět stejný výstup jako v příkladu v předchozí kapitole.

Invoke-Command představuje jednoduchý způsob, jak je možné provést údržbu nebo zjistit informace z několika stanic najednou.

¹⁶ DNS – systém pro přiřazování symbolických jmen IP adresám (z anglického Domain Name System)

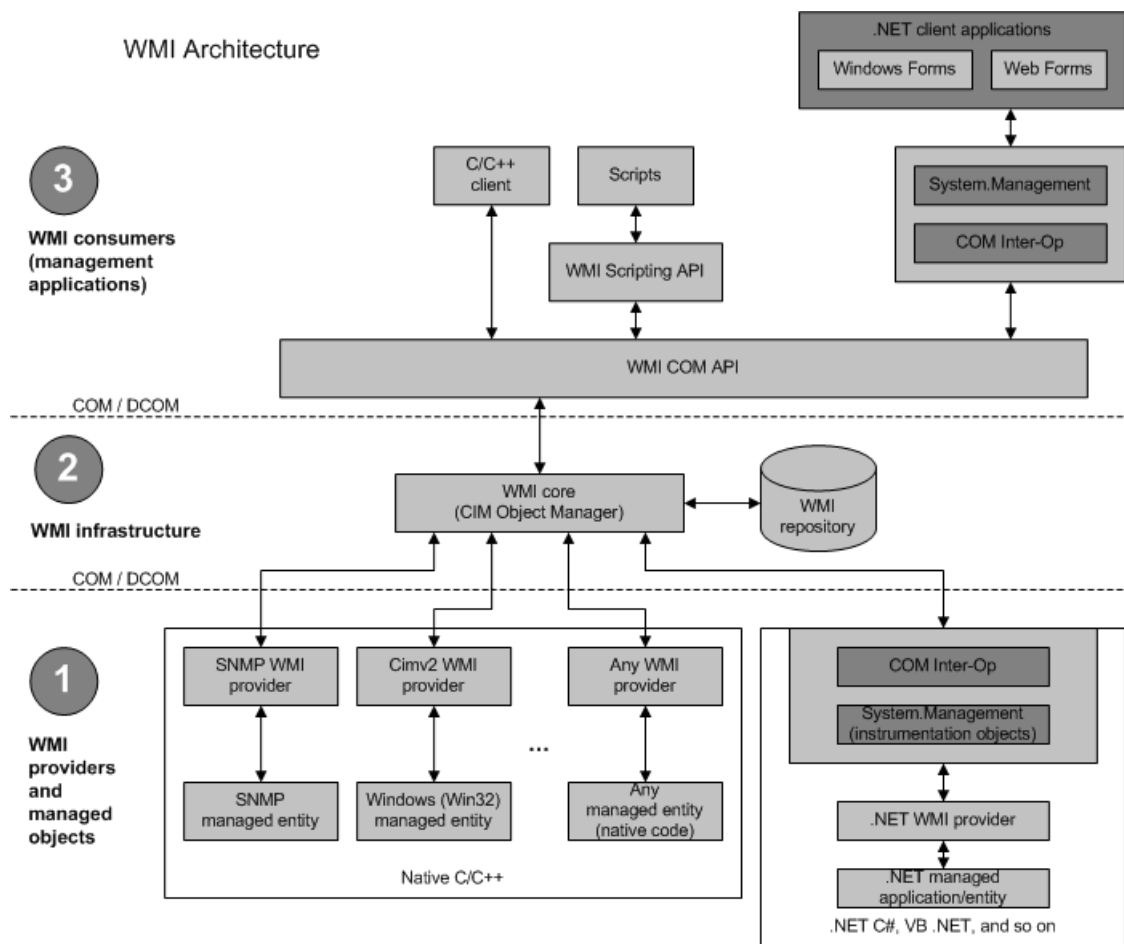
3.14 Rozhraní WMI

Toto rozhraní umožňuje spolupráci operačních systémům a hardwaru od různých výrobců. Je implementací WS-Management protokolu což je protokol vyvinutý skupinou výrobců hardwaru a softwaru jako veřejný standard pro vzdálenou výměnu dat týkajících se správy systému pro jakékoliv zařízení, které tento protokol podporuje [19]. Protokol je založený na SOAP¹⁷.

WMI rozhraní se skládá ze třech částí [8] [20]:

- 1) **Poskytovatelé (WMI providers)** – sem patří vše, k čemu lze rozhraním přistupovat. Jedná se o systém souborů, síťové komponenty, protokoly událostí, soubory, složky, disky, služba Active Directory apod. Provider získává data z objektu, který spravuje a poskytuje tato data WMI. Zároveň zpracovává zprávy z WMI a provádí nad objektem příslušné operace. Provider se skládá ze zkompileované *dll* knihovny, *MOF* souboru, což je textový soubor, který definuje třídy, pro které provider vrací a zpracovává data. Příkladem je *Win32 provider* který má třídu *Win32_LogicalDisk* poskytující informace o logických jednotkách v systému.
- 2) **Infrastruktura (WMI Infrastructure)** – je komponenta operačního systému Windows známá jako *WMI service*. Skládá se ze dvou složek, samotného jádra služby (WMI core) a úložiště (WMI repository). Úložiště sdružuje třídy definované providery, a je organizováno pomocí jmenných prostorů. V úložišti jsou uložena pouze statická data o objektech jako například již zmíněné definice tříd. Jádro komunikuje s providery, ze kterých dynamicky získává potřebné informace. Služba WMI je mezivrstvou mezi poskytovateli, úložištěm a aplikacemi pro správu.
- 3) **Příjemci (WMI Consumers)** – příjemce zpracovává data z rozhraní WMI. Může jím být např. cmdlet nebo softwarový program pro správu či reportování, a nebo jakýkoliv jiný nástroj spouštějící WMI dotazy.

¹⁷ SOAP – protokol pro výměnu zpráv založený na XML (z anglického Simple Object Access Protocol)



Obrázek 3.14.1: zobrazuje architekturu a části rozhraní WMI, které byly popsány výše. Zdroj [20].

Architekturu přibližuje Obrázek 3.14.1. Ukázku použití pak Kód 3.14.1 kde se pomocí WMI rozhraní vypíše stav služeb v systému:

```
PS C:\> Get-WmiObject -Class win32_service -Namespace
"root\cimv2" | Format-List Name, State
```

Kód 3.14.1: Ukázka použití WMI rozhraní

Parametr *Class* určuje WMI třídu se kterou se bude pracovat. Parametr *Namespace* specifikuje jmenný prostor, kde je třída uložena. Zřetězení s *Format-List* jen filtruje vypisované vlastnosti.

Oříznutý výstup z předchozího příkladu (Kód 3.14.1) pak vypadá následovně (Kód 3.14.2):

```
Name : AdobeARMservice
State : Running

Name : AdobeFlashPlayerUpdateSvc
State : Stopped

Name : AeLookupSvc
State : Stopped
```

Kód 3.14.2: Výstup příkazu Get-WmiObject

Pro získání informací z rozhraní je také možné použít syntaxi typu SQL. Příkaz pro zjištění informace o fyzickém disku pak ilustruje Kód 3.14.3:

```
PS C:\> Get-WmiObject -Query "Select * from Win32_DiskDrive"

Partitions : 4
DeviceID    : \\.\PHYSICALDRIVE0
Model       : ST500LM0 ST500LM000-SSHD- SCSI Disk Device
Size        : 500105249280
Caption     : ST500LM0 ST500LM000-SSHD- SCSI Disk Device
```

Kód 3.14.3: Získání informací z WMI pomocí SQL syntaxe

Pro vyhledání všemožných WMI tříd, je vhodný webový portál Microsoft Developer Network [21]. Pro vypísání všech dostupných WMI tříd v systému je možné použít cmdlet *Get-WmiObject -List*.

3.15 Model CIM

PowerShell od verze 3.0 přináší možnost pracovat s modelem CIM. Common Information Model (CIM) je standard, který poskytuje obecnou definici správy informací týkajících se systémů, sítí, aplikací a služeb. WMI je implementací tohoto modelu v systému Windows [6]. Z pohledu abstrakce je tedy model CIM niž než WMI.

Cmdlety poskytují nový způsob načítání existujících instancí rozhraní WMI. Jednou z výhod je rychlejší načítání dat a informací z WMI než pomocí cmdletu *Get-WmiObject*. Další vylepšení je používání protokolu WS-MAN (a tedy WinRM) pro připojování ke vzdáleným stanicím (stejný protokol jako používá PowerShell Remoting). Díky snadné konfigurovatelnosti služby WinRM (jak bylo zmíněno v kapitole 3.13) je možné použít model CIM i pro snadnou správu vzdálených stanic. S modelem CIM přichází také několik cmdletů, které usnadňují práci s WMI rozhraním [8].

3.15.1 Vyhledávání tříd

Pomocí cmdletu *Get-CimClass* je možné pomocí zástupných znaku vyhledat WMI třídu u které přesně neznáme jméno. Zadáním následujícího příkazu a následným použitím tabelátoru nebo kombinací kláves CTRL + Mezerník můžeme listovat třídami, které obsahují slovo *system* viz. Kód 3.15.1:

```
PS C:\> Get-CimClass -ClassName *system*
```

Kód 3.15.1: Ukázka dohledávání CIM tříd pomocí kláves CTRL + Mezerník

Vyhledávání je možné rozšířit o další podmínky, např. hledání třídy, která poskytuje metodu obsahující název *boot*. Příklad s filtrováním, které vrátí pouze jména tříd viz. Kód 3.15.2:

```
PS C:\> Get-CimClass -ClassName *system* -MethodName *boot* |  
Format-List CimClassName, CimClassMethods  
  
CimClassName      : CIM_OperatingSystem  
CimClassMethods   : {Reboot, Shutdown}  
  
CimClassName      : Win32_OperatingSystem  
CimClassMethods   : {Reboot, Shutdown, Win32Shutdown,  
Win32ShutdownTracker...}
```

Kód 3.15.2: Příklad filtrování CIM tříd s metodou obsahující slovo boot

3.15.2 Asociace

Součástí modelu CIM je cmdlet *Get-CimAssociatedInstance*, který umožňuje používání asociací mezi třídami. Nejdříve je potřeba načíst instanci CIM třídy do proměnné. Proměnná se pak použije v dalším příkazu, kdy se načtou instance, které se k uložené instanci vážou.

V příkladu (Kód 3.15.3) jsou vyhledány logické disky a uloženy do proměnné *disk* a dále je vybrán pomocí asociace fyzický disk (pomocí indexu 0 je vybrána první logická jednotka), který se k logickému disku váže.

```
PS C:\> $disk = Get-CimInstance win32_logicaldisk  
PS C:\> Get-CimAssociatedInstance $disk[0] | Format-List
```

Kód 3.15.3: Použití asociace mezi CIM třídami pro zjištění který fyzický disk se váže k logickému

Výstup předchozích příkazů uvedených v Kód 3.15.3 pak vypadá následovně (Kód 3.15.4):

```
NumberOfBlocks : 398831687
BootPartition  : False
Name           : Disk #0, Partition #1
PrimaryPartition : True
Size           : 204201823744
```

Kód 3.15.4: Výstup příkazu pro zjištění vazby mezi fyzickým a logickým diskem (výstup je oříznutý)

3.16 Integrace

PowerShell je možné integrovat do vlastních aplikací, jak již bylo zmíněno v kapitole 3.2. Zde (Kód 3.16.1) je uvedeno jak jednoduše, lze integraci provést do triviální konzolové aplikace. Podobným způsobem by vypadala i integrace například do webové aplikace postavené na technologii ASP.NET.¹⁸ Webové aplikace v ASP.NET jsou typicky implementovány v jazyce C#, je ale možné použít i jiné jazyky, které .NET Framework podporuje [23]. ASP.NET aplikace běží na webovém serveru IIS¹⁹. Pro integraci PowerShellu stačí do příslušné třídy připojit knihovnu *System.Management.Automation*. Pak je možné využít třídu *PowerShell* z této knihovny a pomocí příslušných metod spouštět PowerShell příkazy, skripty, odchyťávat výjimky, zpracovávat výstup apod. Následující příklad (Kód 3.16.1) ilustruje možné použití PowerShellu (jazyk C#):

```
using System;
using System.Collections.ObjectModel;
using System.Management.Automation;

namespace ConsoleApplication1 {
    class Program {
        static void Main(string[] args) {
            RunScript();
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void RunScript() {
            var shell = PowerShell.Create();
            shell.Commands.AddScript("Get-Process");
            Collection<PSObject> results = shell.Invoke();
            Console.WriteLine("Running processes:");
            foreach (var psObject in results) {
                Console.WriteLine(psObject.Properties["ProcessName"].Value);
            }
        }
    }
}
```

Kód 3.16.1: Ukázka integrace PowerShellu do konzolové aplikace v jazyce C#

¹⁸ ASP.NET – je součástí platformy .NET pro vývoj webových aplikací.

¹⁹ IIS – Webový server od společnosti Microsoft (z anglického Internet Information Services).

V příkladu (Kód 3.16.1) je implementována primitivní konzolová aplikace, která po spuštění vypíše do konzole názvy aktuálně běžících procesů. Ve funkci *RunScript* je vytvořena instance PowerShell engine a uložena do proměnné *shell*. Po té je pomocí metody *AddScript* vložen cmdlet pro získání všech běžících procesů. Výsledné procesy jsou jako objekty uloženy do kolekce *results* a potom jsou v cyklu jednotlivě procházeny. U každého procesu je pak vypsána hodnota vlastnosti *ProcessName* (což odpovídá názvu procesu).

4 Výběr činností administrátora

Jak je zřejmé z předchozí kapitoly, pomocí PowerShellu je možné spravovat velmi rozsáhlou oblast. Navrhovaný systém včetně mobilní aplikace nemůže (ani to není žádoucí) celou tuto oblast obsáhnout. Je tedy potřeba vybrat několik vhodných činností.

Vybrané činnosti budou rozděleny do tří kategorií. V první kategorii budou činnosti, které budou nějakým způsobem pracovat a manipulovat s uživatelskými účty ve službě Active Directory. Ve druhé kategorii budou činnosti, které se zaměřují na práci s úložištěm. Činnosti ze třetí kategorie poskytnou možnost spravovat služby a procesy.

4.1 Active Directory

Active Directory je rozšiřitelná adresářová služba, která uchovává informace o Windows doméně a umožňuje centralizovaně spravovat síťové prostředky. Mezi tyto prostředky patří informace o uživateli, stanicích, jejich skupinách, účtech apod. Služba je založena na standardním protokolu LDAP²⁰. Adresáře jsou spravovány doménovými řadiči [22].

Vybrané činnosti vztahující se k Active Directory jsou následující:

- **Získání uživatele podle zadaného loginu** – umožní zjistit, zda uživatelský login existuje, kdy byl vytvořen, datum posledního přihlášení, kdy účet vyprší a podobně.
- **Získání členství uživatele** – zjistí jakých skupin je uživatel členem.
- **Získání všech uživatelů** – vrátí seznam účtů všech uživatelů a zjistí, zda jsou účty povoleny.
- **Získání neaktivních uživatelů** – vrátí seznam uživatelů, kteří se po vymezenou dobu nepřihlásili do systému.
- **Získání neaktivních stanic** – vrátí seznam stanic, ke kterým se po vymezenou dobu nikdo nepřihlásil.

²⁰ LDAP – standardizovaný protokol pro ukládání, komunikaci a přístup k datům pro adresářové služby (z anglického Lightweight Directory Access Protocol).

4.2 Úložiště

- **Zjištění informací o úložišti** – zjistí informace o místních diskových jednotkách, jako je název svazku, kapacita, využití/volné místo.

4.3 Služby a procesy

- **Získání všech služeb** – vrátí seznam všech služeb v systému a zjistí, zda aktuálně běží nebo jsou zastaveny.
- **Získání všech procesů** – vrátí seznam aktuálně běžících procesů a informace o aktuálně spotřebovávaných zdrojích (operační paměť, využití procesoru).
- **Získání informací o službě** – vrátí bližší informace o službě, jako je typ spouštění, závislé služby a naopak i služby, na kterých je závislá aktuální.
- **Získání informací o procesu** – vrátí bližší informace o procesu, jako je priorita, využití procesorového času, datum spuštění procesu apod.

5 Požadavky na systém

V této kapitole budou popsány konkrétní požadavky na systém a také funkčnost systému, které vycházejí z předchozí kapitoly.

Základním požadavkem na systém, který je přímo zmíněn v zadání této práce, je vytvoření klientské aplikace pro platformu Android²¹. Aplikace umožní vzdáleně provádět správu systému Windows Server. Konkrétní činnosti správy a jejich popis již byly zmíněny v předchozí kapitole 4.

Tabulka 4.3.1 zobrazuje pro přehlednost znovu jednotlivé činnosti bez bližšího popisu.

Správa Active Directory	Správa služeb a procesů	Správa Úložiště
Získání uživatele	Získání informací o službě	Zjištění informací o úložišti
Získání členství uživatele	Získání všech služeb	
Získání všech uživatelů	Získání informací o procesu	
Získání neaktivních uživatelů	Získání všech procesů	
Získání všech stanic		
Získání neaktivních stanic		

Tabulka 4.3.1: Funkčnosti mobilní aplikace

5.1 Účel systému

Systém slouží ke správě Windows Serveru pomocí mobilní aplikace. Mobilní aplikace umožní operativní řešení vybraných problémů a pomůže tak částečně zefektivnit celý proces správy serveru.

²¹ Android – Otevřený operační systém určený pro mobilní zařízení.

5.2 Rozsah systému

Na spravované stanici bude nasazena serverová část systému, která bude poskytovat API přístupné pomocí webových služeb. Klientské mobilní aplikace budou používat toto API pro komunikaci se serverovou částí a provádění příslušných operací. Uživatelé mobilní aplikace se pomocí jména a hesla přihlásí ke konkrétnímu serveru. Na tomto serveru bude společně se serverovou částí systému umístěna databáze uživatelů, sloužící k autorizaci. Jedna klientská aplikace bude moci spravovat více serverových stanic (ne současně).

5.3 Uživatelé systému

V systému budou definovány dvě uživatelské role:

- **Admin** – bude typicky osoba pověřená správou dané stanice. Bude moci používat veškerou funkčnost mobilní aplikace a spravovat tak příslušnou stanici.
- **Supervisor** – bude osoba, která bude mít na starost organizaci Adminů. Bude moci spravovat účty a role, ale nebude mít přístup do klientské aplikace.

Pokud bude uživatel přiřazen do obou rolí, bude moci využívat veškerou funkcionalitu.

5.4 Omezující podmínky

Klientská mobilní aplikace vyžaduje zařízení se systémem Android s minimálním API 19, což je verze Android 4.4. Dále je potřeba zajistit připojení tohoto zařízení k internetu.

Spravovaným serverem pak bude Windows Server 2012R2 se službami Active Directory a IIS.

5.5 Autorizace

Provádět správu systému pomocí mobilní aplikace bude možné až po ověření identity uživatele. Při spuštění aplikace bude uživatel vyzván k zadání jména nebo IP adresy serveru, ke kterému se bude chtít připojit, a také k zadání přihlašovacího jména a hesla k příslušnému serveru. Přihlašovací jména budou unikátní v rámci jednoho serveru. Účty bude vytvářet uživatel se speciálním pověřením (s rolí supervisor), obyčejní uživatelé (s rolí admin) nebudou moci žádné účty vytvářet ani spravovat. Správa účtu bude prováděna pomocí webového rozhraní, které bude součástí serverové aplikace. Správa účtu nebude přístupná v mobilní aplikaci.

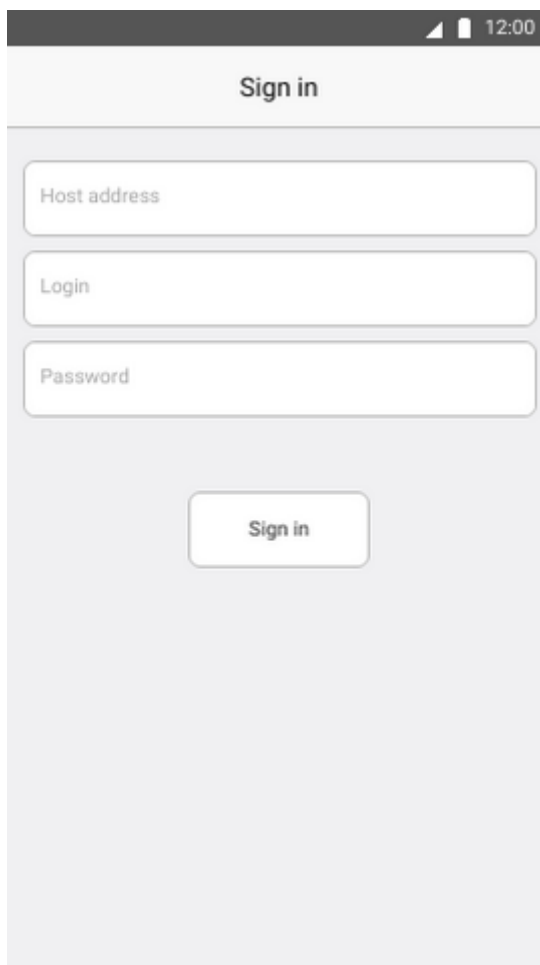
Většina služeb poskytovaných přes API serveru bude dostupná pouze pro autorizované uživatele. Aby nebylo nutné při každém požadavku neustále přenášet uživatelské jméno a heslo, bude pro autorizaci použit token s omezenou platností. Po vypršení platnosti tokenu nebude možné dále spouštět služby, které vyžadují autorizaci, a bude nutné opětovným zadáním hesla token obnovit.

5.6 Funkčnost mobilní aplikace

Celé uživatelské rozhraní aplikace je od začátku navrhováno pro používání ve vertikální poloze (v rámci platformy Android označované jako *portrait*).

5.6.1 Přihlášení

Po spuštění aplikace bude uživatel vyzván k vyplnění názvu nebo IP adresy serveru a k zadání přihlašovacích údajů jména a hesla vázaných k serveru. Pokud vše proběhne bez komplikací (správná adresa serveru a správné přihlašovací údaje s příslušným oprávněním) otevře se část aplikace, v které již bude možno provádět správu. Obrázek 5.6.1 zobrazuje přihlašovací část aplikace.



Obrázek 5.6.1: Rozvržení přihlašovací obrazovky mobilní aplikace

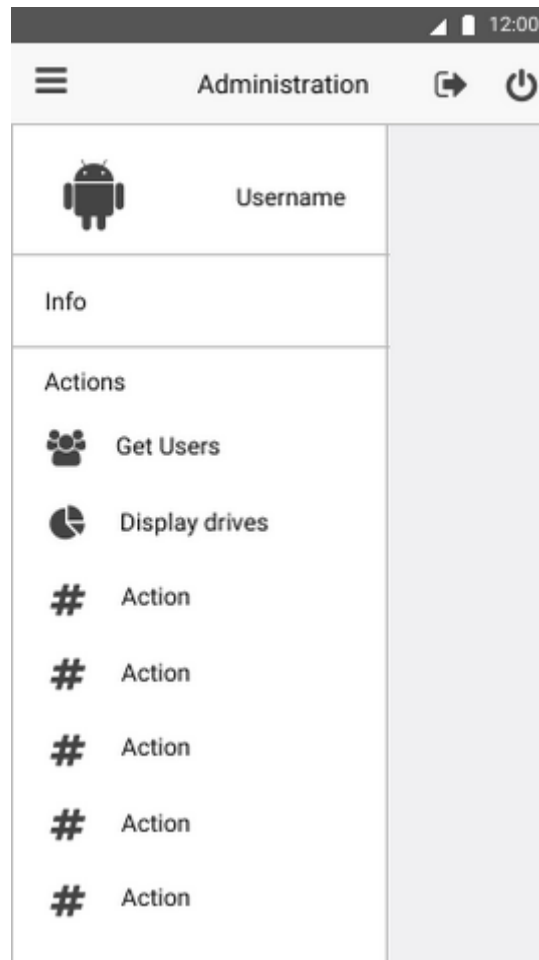
Pokud dojde během přihlašovacího procesu k chybě, bude o tom uživatel informován zobrazením správy o chybě. Chybovým stavem může být: nedostupnost internetové spojení, špatná adresa serveru, příliš dlouhá doba připojování k serveru nebo špatné přihlašovací údaje. Zároveň budou validovány všechny vstupy tak, aby obsahovaly pouze zvolené povolené znaky.

Kvůli zpříjemnění práce s aplikací se po opětovném spuštění aplikace automaticky předvyplní poslední vložená adresa serveru a login uživatele (pouze pokud bylo přihlášení v předchozí relaci úspěšné). Heslo z důvodu bezpečnosti ukládáno nebude a bude potřeba jej vyplnit znovu.

5.6.2 Správa

Po přihlášení se otevře hlavní část mobilní aplikace s uvítací obrazovkou. V horní části se budou nacházet tlačítka pro odhlášení a pro ukončení aplikace. Odhlášení vrátí uživatele do části přihlašování, kde se tak bude moci například připojit k jiné stanici.

Ukončení aplikace kompletně zavře aplikaci a po opětovném spuštění bude potřeba znovu se přihlásit. Obě akce budou potvrzovány pomocí dialogových oken, aby se předešlo případnému nechtěnému zavření nebo odhlášení z aplikace. Dále bude v horním menu tlačítko pro otevření postranního panelu, ve kterém bude zobrazen login aktuálně přihlášeného uživatele a dále položky jednotlivých činností správy. Postranní panel bude moci být zobrazen také přejetím po obrazovce z levé strany doprava. Rozložení horního menu a postranní panel zobrazuje Obrázek 5.6.2.

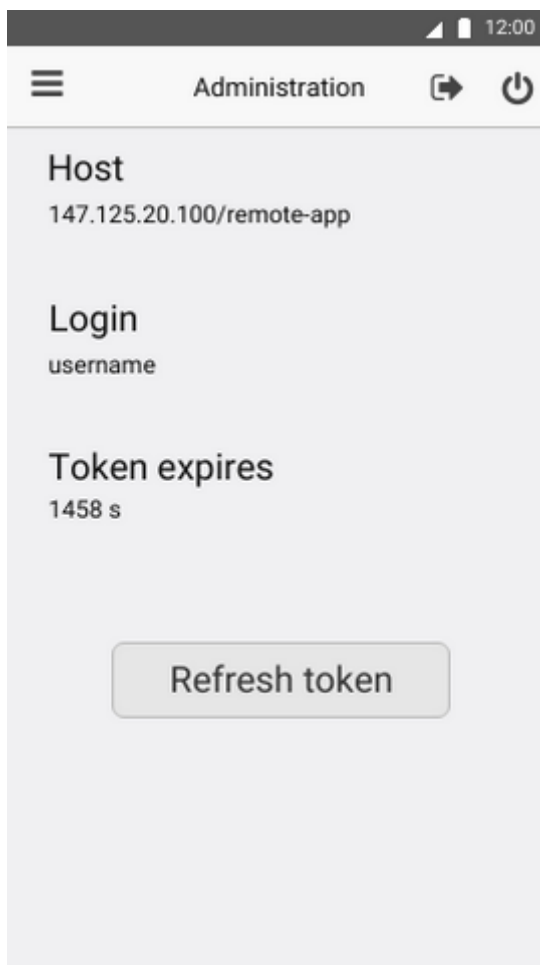


Obrázek 5.6.2: Rozložení hlavního menu aplikace

Po výběru příslušné akce se postranní panel schová a zobrazí se rozhraní pro ovládání konkrétní činnosti. Panel bude možné kdykoliv znovu otevřít a pohybovat se mezi činnostmi dle libosti.

Kromě jednotlivých panelů činností, bude aplikace obsahovat informační panel, ve kterém bude zobrazeno, k jakému serveru je aplikace momentálně připojena,

pod jakým účtem a čas do vypršení autorizačního tokenu. Na tomto panelu bude tlačítko pro obnovení autorizačního tokenu. Po stisknutí pro obnovení tokenu se z důvodu bezpečnosti otevře dialog pro zadání hesla k aktuálnímu účtu. Po zadání správného hesla se přihlášení obnoví. Pokud vyprší platnost tokenu, bude možné dále procházet aplikací, ale při spuštění některé z akcí serverová část z důvodu neautorizovaného přístupu akci neprovede a informuje o tom uživatele. Obrázek 5.6.3 zobrazuje navrhované rozvržení tohoto panelu.



Obrázek 5.6.3: Info panel s možností obnovení přihlášení

Konkrétní činnosti správy budou mít layout složený z dále popsaných částí s případnými změnami záviselými na konkrétní činnosti. V horní části bude název příslušné činnosti. U názvu bude tlačítko, které formou tooltipu zobrazí bližší informace o tom, co konkrétní činnost provádí a případně co se očekává, že poskytne za informace. V horní sekci bude také umístěno tlačítko, které spustí provádění dané činnosti. Pod touto částí budou umístěny vstupy, u kterých se očekává, že je uživatel vyplní (pokud to daná činnost vyžaduje a pokud je vyplnění povinné pro správnou funkčnost činnosti). V dolní části pak bude umístěn výstup. Ten bude nejčastěji ve formě textového okna, které se naplní různými informacemi v závislosti na dané činnosti. Další formou výstupu budou grafy konkrétně u zobrazení informací o velikosti a dostupné kapacitě disků. Pokud se bude jednat o činnosti jako je zobrazení běžících procesů, bude v dolní části zobrazen seznam s možností rolování. Ukázka typického rozvržení činnosti viz. Obrázek 5.6.4.



Obrázek 5.6.4: Ukázka typického rozvržení panelu činnosti

U činností, které mění stav systému jako je zastavení / restart služby nebo zastavení procesu, bude před provedením zobrazen dialog pro potvrzení prováděné akce, aby se alespoň částečně předešlo nechtěným změnám. U činností, kde se provádí pouze dotazy na stav systému, se žádný dialog nezobrazí a akce se provede ihned po spuštění.

Pokud dojde během provádění činnosti k nějaké chybě nebo neočekávané události, bude o tom uživatel informován pomocí dialogu. Chyby a události, které se mohou vyskytnout, mohou být jak v rámci problémů s připojením, tak v rámci autorizace (vypršení autorizačního tokenu), ale také v rámci samotných činností (například pokus o zastavení služby, která neběží). Na tyto situace je potřeba myslet při implementaci a výjimky ošetřit.

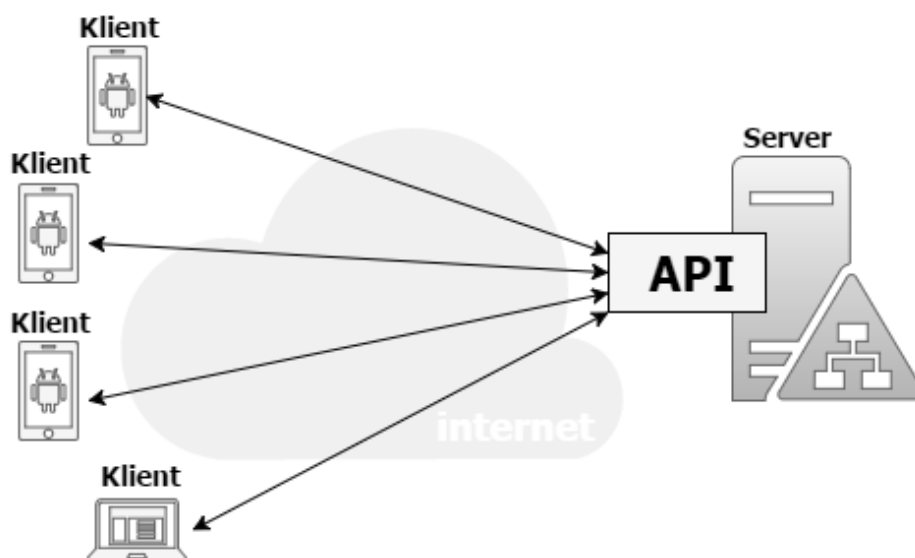
5.7 Funkčnost serverové části

Hlavní funkcí serverové části je poskytnout rozhraní pro mobilní aplikaci, pomocí kterého bude prováděna správa. Další funkcí bude poskytnutí rozhraní pro správu účtů. Serverová část aplikace musí také zajistit autorizaci uživatelů. Zároveň je potřeba zajistit provádění konkrétních činností správy.

6 Návrh systému

V této kapitole bude popsán konkrétní návrh systému. Budou zmíněny použité technologie a také důvody proč byly zvoleny. Bude popsána architektura celého řešení i podsystemů.

Základní rozvržení systému přibližuje Obrázek 6.1. Jednotlivé mobilní aplikace využívají služeb, které poskytuje aplikace umístěná na serveru. Server, na kterém je aplikace umístěna, je zároveň doménovým řadičem. Služby pak zajišťují správu pomocí PowerShell skriptů.



Obrázek 6.1: Rozvržení systému pro mobilní správu

6.1 Technologie

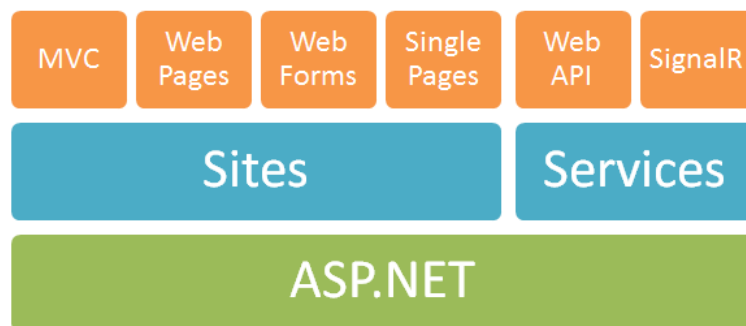
Pro mobilní aplikaci je volba technologie jasná a vychází ze zadání, kde je požadováno, aby vytvořená aplikace využívala platformu Android.

U serverové části aplikace už možnost volby konkrétní technologie je. Důležitým faktorem výběru vhodné technologie je možnost provázání s PowerShellem pomocí kterého bude probíhat vlastní správa. Pro vytváření webových aplikací existuje velké množství technologií, platforem a jazyků. Mezi tyto technologie patří PHP, Java, Ruby on Rails, Python, NodeJs apod. Pro vyvíjený systém se jako nejvhodnější kandidát jeví

platforma .NET. Jak již bylo zmíněno v kapitole 3.16, je integrace PowerShellu do aplikací postavených na této platformě poměrně snadná a přímočará. Navíc se jedná o stejnou platformu, na které je PowerShell postaven. Dalším plusem je využití IIS webserveru pro nasazení aplikace. Tento webserver je standardně integrován ve Windows Serveru a odpadá tak potřeba instalace webserveru nutného pro nasazení aplikace. Stejně tak je i platforma .NET součástí instalace Windows Serveru. Zvolená technologie pro serverovou část aplikace je tedy platforma .NET, konkrétně její část poskytující možnosti pro vývoj webových aplikací ASP.NET.

6.1.1 ASP.NET

ASP.NET je framework pro vývoj webových aplikací a je součástí platformy .NET. Umožňuje vyvíjet jednoduché webové aplikace pomocí HTML²², CSS²³ a Javascript²⁴, ale také rozsáhlé aplikace využívající principy a technologie jako je SOA²⁵. Také podporuje frameworky pro vývoj mobilních aplikací pomocí webových technologií jako například PhoneGap²⁶ [23]. Obrázek 6.1.1 ukazuje technologie a služby, které tento Framework poskytuje.



Obrázek 6.1.1: Technologie a služby poskytované ASP.NET frameworkem

²² HTML – značkovací jazyk pro tvorbu webových stránek (z anglického HyperText Markup Language).

²³ CSS – jazyk sloužící pro popis způsobů zobrazení elementů hlavně webových stránek (z anglického Cascading Style Sheets).

²⁴ Javascript – interpretovaný programovací jazyk využívaný obvykle pro interakci prvků na klientské straně webových aplikací.

²⁵ SOA – architektura softwarových aplikací, založená na myšlence poskytnout funkcionalitu jednotlivých komponent pomocí služeb dostupných přes nečastěji přes web [26] (z anglického Service Oriented Architecture).

²⁶ PhoneGap – framework umožňující vyvíjet mobilní aplikace pomocí technologií HTML5, CSS3, Javascript.

Pro návrh systému bude využito principů architektury MVC, kdy budou jednotlivé komponenty rozděleny do vrstev a každá vrstva bude mít na starost část funkcionality.

6.2 Serverová část

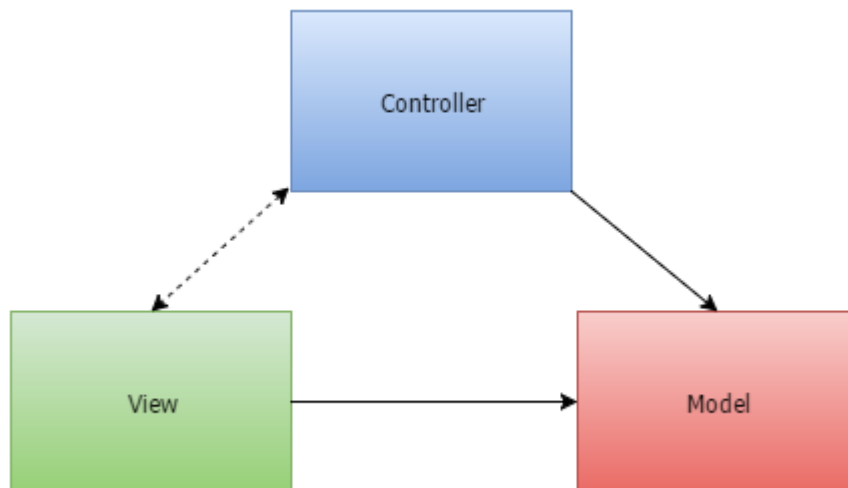
Serverová část aplikace bude jak již bylo zmíněno v předchozí kapitole vytvořena pro platformu ASP.NET. Aplikace bude poskytovat webové API, pomocí kterého k ní budou přistupovat klientské aplikace. Také bude poskytovat webové rozhraní pro správu uživatelů. Poskytované služby budou identifikovány pomocí URI²⁷.

Aplikace bude ukládat informace potřebné pro autorizaci do databáze. Jako databázový engine bude použit Microsoft SQL Server 2012 Express. Jedná se o free verzi MSSQL Serveru s omezením některých funkcionalit. Tento systém je zvolen z důvodu předpokládané snadné integrace do celého systému, jelikož se jedná o databázový systém přímo od společnosti Microsoft.

Správu serveru bude aplikace provádět spouštěním připravených PowerShell skriptů. Skripty budou vytvořené standardně ve formě textových souborů s příponou *ps1* a budou umístěné v konkrétní složce v rámci aplikace. Spouštění skriptů bude probíhat v aplikaci na základě zpracovaných požadavků, které budou přicházet od klientů.

Architektura aplikace bude postavena na třívrstvé architektuře MVC. Vrstvy modelu jsou Model, View a Controller. Obrázek 6.2.1 zobrazuje umístění jednotlivých vrstev architektury.

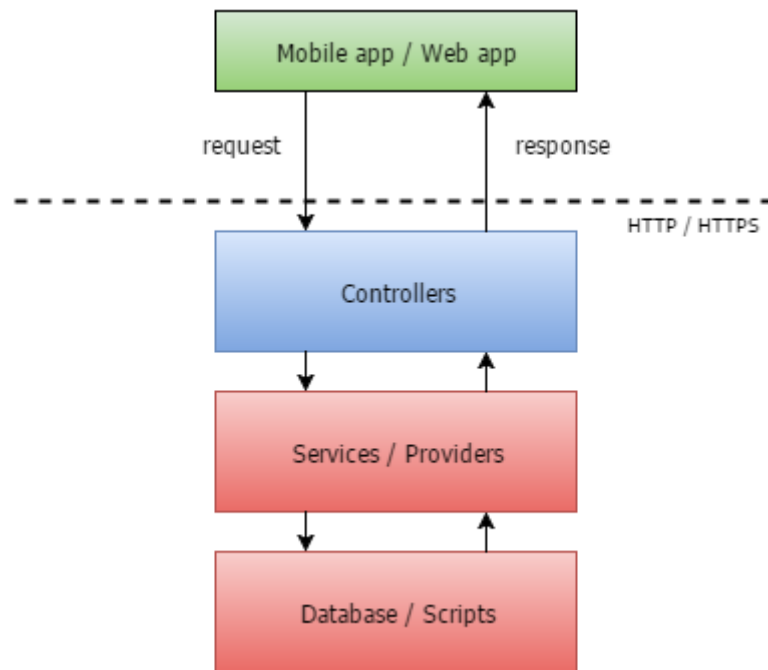
²⁷ URI – odkaz konkrétní zdroj nebo službu pomocí unikátního identifikátoru (z anglického Uniform Resource Identifier).



Obrázek 6.2.1: MVC architektura

Popis funkce vrstev:

- **View** - bude mít na starost zobrazení dat uživateli. Pro část aplikace, která se bude starat o správu uživatelů, to bude výstup ve formě HTML. Pro část aplikace, která bude poskytovat webové rozhraní, to bude výstup ve formě některého strojově čitelného a snadno přenositelného formátu např. XML nebo JSON.
- **Controller** – podle vstupních požadavků, které budou přicházet od klientů (což bude mobilní aplikace a webové rozhraní), bude controller rozhodovat o dalších akcích, které se budou provádět. O provedení se pak postará sám a nebo k tomu využije vrstvu Model.
- **Model** – bude obsahovat hlavní logiku aplikace a vše co do ní spadá. To bude například spouštění skriptů, ukládání uživatelů do databáze apod. Hlavní funkce této části bude na základě určitých parametrů poskytnout a zpracovat příslušná data.

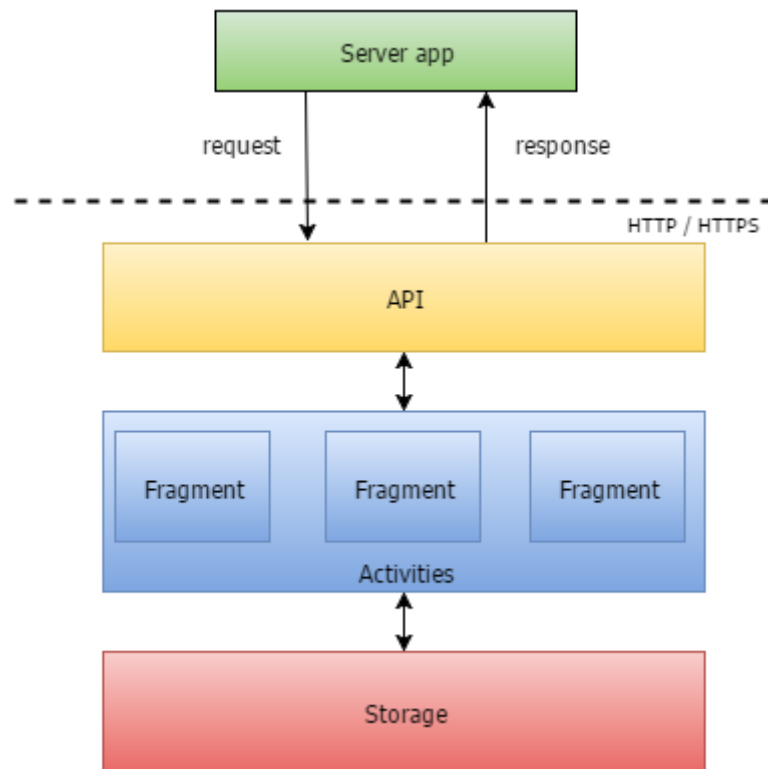


Obrázek 6.2.2: Základní rozvržení architektury serverové aplikace

Obrázek 6.2.2 zobrazuje základní architekturu serverové části, která vychází z třívrstvé architektury. Od klientů přichází přes HTTP protokol nebo přes jeho zabezpečenou nadstavbu HTTPS požadavky do aplikace. Požadavky jsou filtrovány a přiřazeny konkrétnímu controlleru, který se stará o jejich zpracování. Controller při zpracování využívá služeb, které poskytují servisní třídy z vrstvy Model. Služby v rámci zpracování například operují s PowerShell skripty, které načítají ze souborového systému, spouští a zpracovávají jejich výstup. Služby také v rámci zpracování a správy účtů operují s databází.

6.3 Klientská část

Mobilní část bude vytvářena pro platformu Android, z toho přirozeně vyplývá použití jazyka Java, který je pro vývoj na této platformě standardem. Aplikace bude ovládána pomocí uživatelského rozhraní. To bude definovat formu zobrazení dat a možnosti interakce s jednotlivými prvky rozhraní. Data, která bude rozhraní zobrazovat, budou získávána ze serverové části aplikace. K ukládání potřebných dat bude použito některé z interních úložišť, které Android poskytuje.



Obrázek 6.3.1: Základní rozvržení architektury mobilní aplikace

Obrázek 6.3.1 zobrazuje základní rozvržení architektury mobilní aplikace. Základní komponentou aplikace bude Activity, která poskytuje okno aplikace a umožňuje interakci s uživatelem. Součástí Activity pak budou Fragments, které poskytnou konkrétní prvky uživatelského rozhraní a jejich obsluhu. Komponenta API bude zajišťovat komunikaci se serverovou částí aplikace a poskytovat volání příslušných služeb dostupných na serveru. Bude zpracovávat jejich výstup a předávat ho do Aktivit a Fragmentů. Komponenta Storage bude zajišťovat ukládání potřebných dat a zpřístupňovat je Aktivitám a Fragmentům.

7 Systém MoWin

V následující kapitole bude popsána vlastní implementace celého systému.

7.1 MoWin – Klient

MoWin-Klient je mobilní aplikace pro systém Android. V implementaci bude popsáno uživatelské rozhraní, které je poskytováno pomocí aktivit a fragmentů, způsob ukládání a získávání dat, komunikace se serverovou částí, nutná oprávnění a další důležité části.

7.1.1 Správa stavu aplikace

Pomocí třídy *AppController* jsou spravovány parametry jako je umístění a přístupnost úložiště, chování klávesnice a invalidace přihlašovacího tokenu. Třída dědí od třídy *Application* což je základní třída pro udržování a správu stavu aplikace. Je navržena pomocí vzoru *singleton* což zajišťuje, že bude v rámci aplikace používána pouze jedna instance a tím bude zaručen konzistentní stav. Část kódu třídy *AppController* zobrazuje Kód 7.1.1.

```
public class AppController extends Application {

    private static AppController instance;
    public static final String PREF_NAME = "cz.mowin";
    public static final String CERT_FILE_NAME = "certificate.pfx";
    private static SharedPreferences pref;

    @Override
    public void onCreate() {
        super.onCreate();
        instance = this;
        pref = this.getSharedPreferences(AppController.PREF_NAME,
getAppContext().MODE_PRIVATE);
    }

    public static AppController getInstance() {
        return instance;
    }

    public static Context getAppContext() {
        return instance.getApplicationContext();
    }

    ...
}
```

Kód 7.1.1: Ukázka části třídy pro správu stavu aplikace AppController

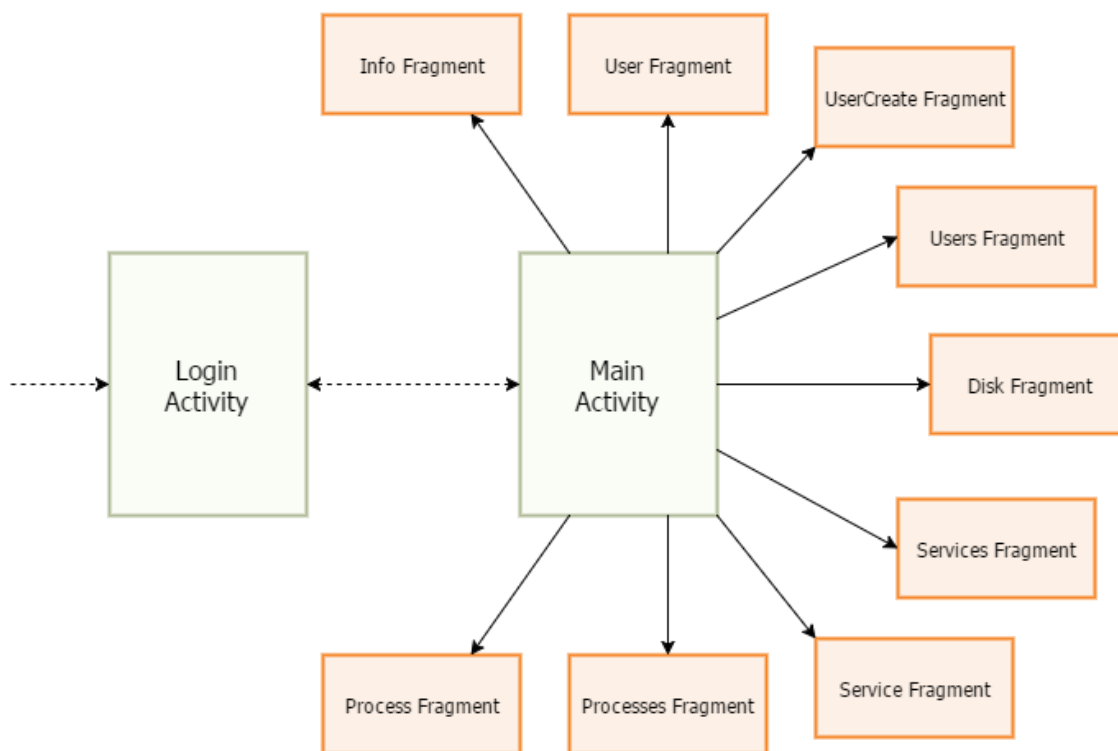
7.1.2 Zabezpečení připojení

Při přihlašování v *LoginActivity* se nejdříve ze serveru stáhne a uloží certifikát pomocí funkce *getCert()* kterou poskytuje třída *Api*. Tento proces probíhá pomocí

nezabezpečeného přenosu. Až po té je proveden pokus o přihlášení pomocí funkce *getToken()*, kdy se už pomocí zabezpečeného přenosu posílá uživatelské jméno a heslo. Pokud jsou přihlašovací údaje validní, server vrátí autorizační token, který je uložen do systémového úložiště aplikace a používá se v další komunikaci. S tokenem se vrací i informace o době jeho platnosti, která je také uložena a následně zobrazena v *InfoFragment*, kde pak uživatel vidí, jaká doba zbývá do vypršení platnosti. Jakmile platnost tokenu vyprší, jakékoliv požadavky na server začnou vracet chybu o vypršení autorizace a informovat o tom uživatele pomocí vyskakovacího okna. Token je pak možné obnovit v části *InfoFragment*, z důvodu bezpečnosti je ale při obnově nutné zadat přihlašovací heslo.

7.1.3 Uživatelské rozhraní

Aplikace se skládá ze dvou aktivit, *LoginActivity* a *MainActivity*. Po spuštění aplikace je spuštěna *LoginActivity*. Ta má za úkol získat od uživatele adresu serveru a přihlašovací údaje a po té zajistit autorizaci uživatele. Po úspěšné autorizaci je aktivita ukončena a je spuštěna *MainActivity*. Tato aktivita poskytuje vysunovací ovládací panel, který zobrazuje dostupné činnosti správy, mezi kterými je možné se přesunovat kliknutím na konkrétní položku na panelu. Po kliknutí je panel schován a do aktivity je načten příslušný fragment, který zobrazí uživatelské rozhraní a prvky pro ovládání dané činnosti. V horní části je vždy zobrazeno menu s prvky pro otevření ovládacího panelu, odhlášení a ukončení aplikace. Odhlášení ukončí *MainActivity* a vrátí aplikaci zpět na *LoginActivity*. Obrázek 7.1.1 zobrazuje výše popsaný proces.



Obrázek 7.1.1: Diagram aktivit a fragmentů

Provádění všech operací, které vyžadují přenos dat ze serveru, je prováděno pomocí knihovny *Volley*. Při provádění je uživateli zobrazen dialog o aktuálním provádění činnosti a je znemožněna další práce dokud se nevrátí odpověď ze serveru nebo nevyprší timeout volání.

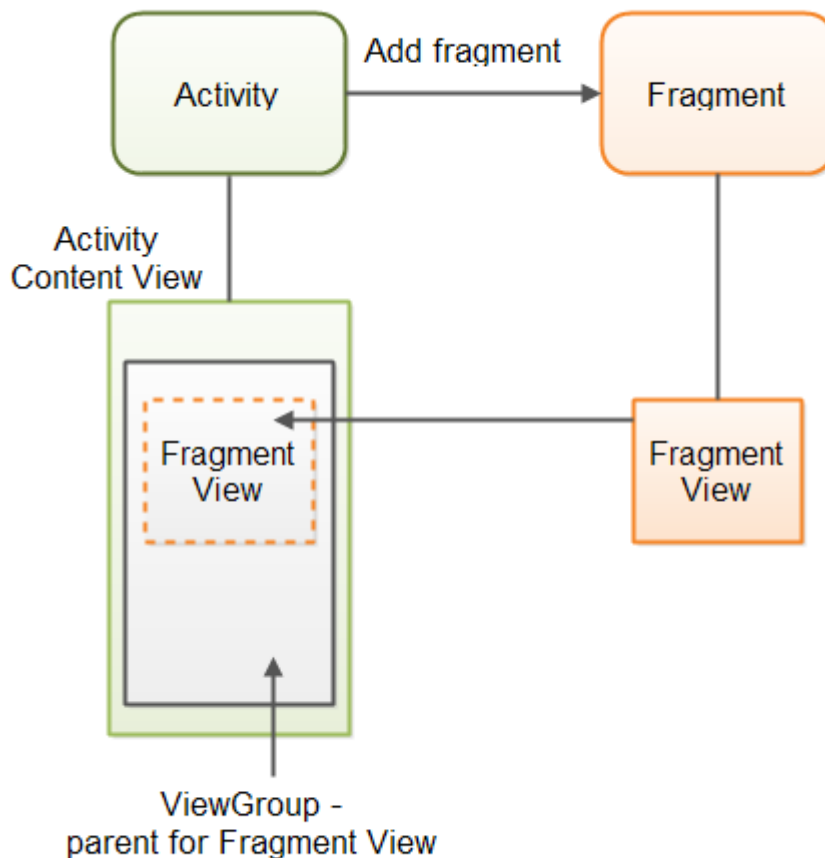
Aplikace má v manifestu (hlavní konfigurační soubor aplikace) nastavenou orientaci u jednotlivých aktivit na výšku (*portrait*). To zamezuje přetáčení obrazovky při natáčení mobilního zařízení a také přidává na přehlednosti. Uživatelské rozhraní bylo od začátku navrhováno pro tento typ zobrazení. Také je zde povoleno oprávnění k využívání Internetu, které je nutné pro komunikaci se serverovou částí systému.

7.1.4 Činnosti správy

Při výběru příslušné činnosti z menu pomocí funkce *onNavigationItemSelected()*, kterou poskytuje *MainActivity* je načten příslušný fragment pomocí funkce *changeFragment()*. Ten pak zpřístupňuje uživatelské rozhraní pro danou činnost viz. Obrázek 7.1.2. Fragменты využívají pro získávání informací funkce z třídy *Api*, která provádí příslušná volání na server a vrací odpověď. Odpovědi jsou pak ve fragmentu deserializovány

na připravené *POJO* objekty s kterými dále pracují. Chybové stavy jsou odchyťvány a zpracovány přímo ve třídě *Api*.

Definice rozložení prvků pro jednotlivé fragmenty, je vždy uložena v XML souboru což je standardní způsob v systému android. Layouty jsou pak uloženy ve složce *res/layout/fragment_{název fragmentu}.xml*. Každý fragment obsahuje v horní části uživatelského rozhraní název a tlačítko (ve formě otazníku) pro otevření tooltipu, který zobrazuje informace o tom, co daná činnost provádí.



Obrázek 7.1.2: Diagram připojení fragmentu do aktivity

Info Fragment

Tento fragment neprovádí žádnou správu. Slouží k zobrazení adresy serveru, ke kterému je aplikace aktuálně připojena, zobrazuje login aktuálně přihlášeného uživatele a dobu za kterou vyprší platnost autorizačního tokenu. Ta je zobrazena jako počet vteřin do

vypršení a každou sekundu se aktualizuje. Fragment obsahuje tlačítko pro obnovu tokenu, to po kliknutí vyzve k zadání přihlašovacího hesla a provede přihlášení stejným způsobem, jako je popsáno v kapitole 7.1.2.

User Fragment

Fragment pro zobrazení informací o uživateli z Active Directory. Obsahuje vstup pro zadání loginu hledaného uživatele, tlačítko pro spuštění operace, textový výstup s informacemi o uživateli, který zobrazuje informace ve formě *Property: Value* a výstup pro zobrazení členství ve skupinách. Skupiny jsou pro větší přehlednost zobrazeny pod sebou. Oba výstupy jsou scrollovatelné. Pro vrácení informací se využívá funkce *getUser()* a *getUserMembership()* z třídy *Api*. Funkce *getUserMembership()* je volána automaticky po úspěšném vrácení informací pomocí *getUser()*. Uživatele je možné také deaktivovat, k tomu se používá funkce *disableUser()* z třídy *Api*.

UserCreate Fragment

Pro vytvoření nového uživatele v Active Directory slouží *UserCreateFragment*. Ten obsahuje vstupy pro jméno, login, heslo a výstup, kam se v případě úspěšného vytvoření načtou informace o nově vytvořeném uživateli. Pro vytvoření se využívá funkce *createUser()* s parametry login, jméno a heslo a pokud je úspěšné, tak i *getUser()* pro vrácení informací.

Users Fragment

Zobrazuje seznam všech uživatelů v Active Directory. V seznamu je uveden login, uživatelské jméno, a zda je uživatelský účet povolen. Pro vykreslení je použit seznam *ListView*, který obsahuje položky s informacemi o uživateli. Rozvržení položek je uloženo v *item_user.xml*. Pomocí funkce *getUsers()* z třídy *Api* je vrácen seznam uživatelů s potřebnými informacemi o každém z nich. Data jsou pomocí adapteru *UsersAdapter* přemapována do příslušných polí definovaných v layoutu položek.

Disk Fragment

Fragment slouží k zobrazení informací o dostupných úložištích na severu. Jedná se o logické diskové jednotky, u kterých je zobrazena kapacita a její využití. Kapacita a využití je zobrazeno pomocí koláčových grafů, kdy je pro každou jednotku zobrazen jeden graf. Grafy jsou zobrazeny využitím knihovny *MPAndroidChart*. Fragment obsahuje jednořádkový výstup, do kterého je načten seznam jednotek a pod něj

pak jednotlivé grafy. Pro vrácení informací se využívá funkce *showDiskSpace()* z třídy *Api*. Parametrem této funkce je název diskové jednotky (pokud není zadán, vrací se všechny jednotky).

Services Fragment

Tento fragment zobrazuje dostupné služby v systému, a stav zda jsou spuštěny nebo zastaveny. Pro vykreslení je použit seznam *ListView* který obsahuje položky s informacemi o službě. Rozvržení položek je uloženo v *item_services.xml*. Pomocí funkce *getServices()* z třídy *Api* je vrácen seznam služeb s potřebnými informacemi o každé z nich. Data jsou pomocí adapteru *ServicesAdapter* přemapována do příslušných polí definovaných v layoutu položek. Kliknutím na jednotlivé položky se přejde na *Service Fragment* s bližšími informacemi o konkrétní službě.

Service Fragment

Fragment zobrazuje informace o konkrétní službě. K tomu využívá *TextView*. Data načítá pomocí funkce *getService()* z třídy *Api*. Dále umožňuje danou službu zastavit, spustit nebo restartovat pomocí *Api* funkcí *stopService()*, *startService()* a *restartService()*. Po spuštění některé z funkcí je automaticky provedena *getService()*, aby se zobrazoval aktuální stav. Všechny funkce používají společný parametr název služby.

Processes Fragment

Tento fragment zobrazuje aktuálně spuštěné procesy v systému. Pro vykreslení je použit seznam *ListView* který obsahuje položky s informacemi o procesu. Rozvržení položek je uloženo v *item_processes.xml*. Pomocí funkce *getProcesses()* z třídy *Api* je vrácen seznam procesů s potřebnými informacemi o každém z nich. Data jsou pomocí adapteru *ProcessesAdapter* přemapována do příslušných polí definovaných v layoutu položek. Kliknutím na jednotlivé položky se přejde na *Process Fragment* s bližšími informacemi o procesu.

Process Fragment

Slouží k zobrazení podrobných informací o konkrétním procesu. Zobrazení výstupu je provedeno pomocí *TextView*. Data se načítají pomocí funkce *getProcess()* z třídy *Api*. Proces je také možné zastavit, k tomu slouží *Api* funkce *stopProcess()* jejíž parametrem je *Id* procesu. Po zastavení se automaticky zavolá *getProcess()*, aby se ověřilo, zda proces skutečně skončil.

7.2 MoWin – Server

MoWin-Server je webová aplikace implementovaná v technologii ASP.NET. Bude popsáno API, které poskytuje pro spuštění skriptů a také funkčnost jako je logování, správa uživatelů, statistiky.

7.2.1 Přehled

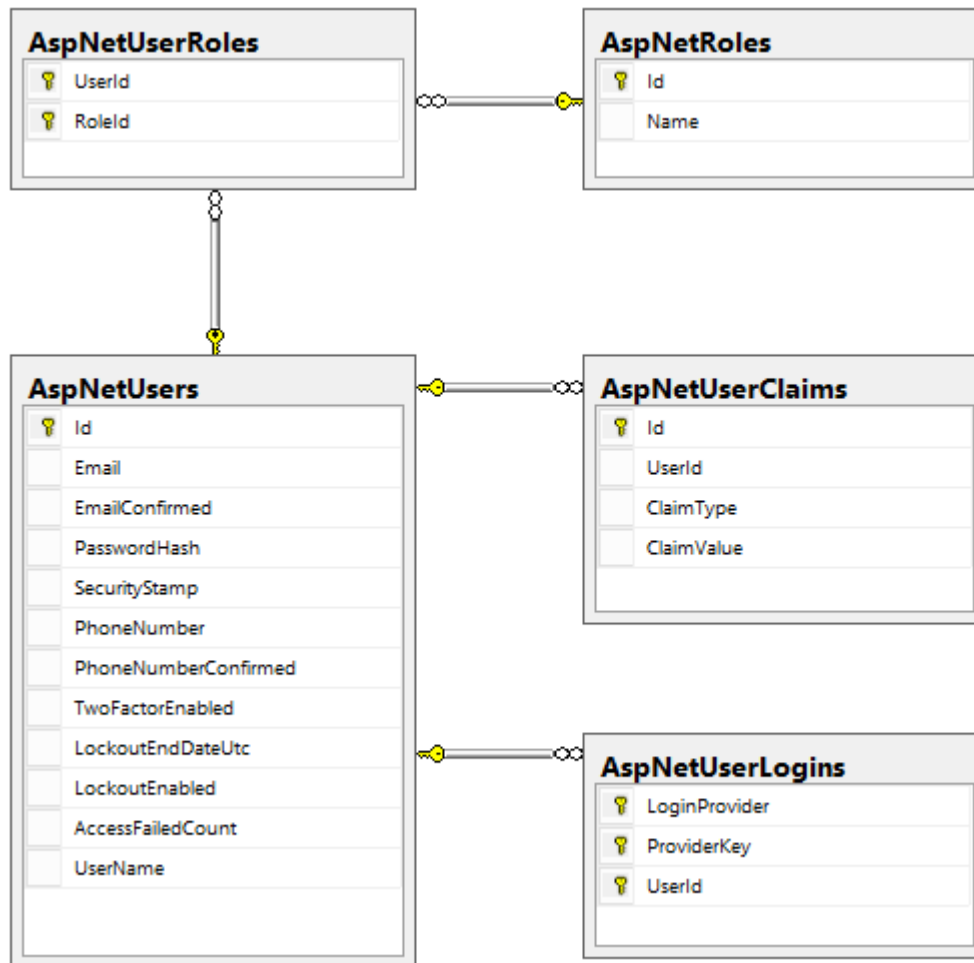
Aplikace je rozdělena do několika částí. Stěžejní částí je API pro spuštění připravených skriptů. Dále je to část pro správu uživatelů a jejich rolí. Logování, které ukládá informace o provedených API voláních a statistiky umožňující zobrazovat využívání jednotlivých skriptů, aktivitu uživatelů, úspěšné i neúspěšné volání a podobně. Přístup k těmto částem je omezen pouze pro uživatele s rolí *supervisor*. Znázornění závislostí jednotlivých tříd je uvedeno v diagramech (Obrázek E.1 a Obrázek E.2) na konci příloh.

7.2.2 Datový model

Pro ukládání dat, které jsou potřebné pro správu uživatelů je použita struktura databáze zobrazená viz. Obrázek 7.2.1. Při práci s databází se maximálně využívá *Entity Framework*. Ten zajišťuje ORM mapování databázových tabulek a jejich vztahů na objekty a zpřístupňuje práci s uživateli pomocí třídy *UserManager*. Struktura databáze byla vygenerována pomocí modulu *ASP.NET Identity* řešícího autorizační proces.

Popis tabulek v diagramu:

- **AspNetUsers** – ukládá nejdůležitější informace o uživateli jako je jméno a heslo.
- **AspNetRoles** – ukládá jednotlivé role.
- **AspNetUserRoles** – pomocná tabulka zajišťující vazbu M:N mezi uživateli a rolemi.
- **AspNetUserClaims** – ukládá tzv. *Claim* což je alternativa k uživatelské roli. Jedná se o standardně generovanou tabulku v *ASP.NET Identity* viz. 7.2.3.
- **AspNetUserLogins** – slouží k ukládání informací o externích poskytovatelích přihlášení (např. Facebook).



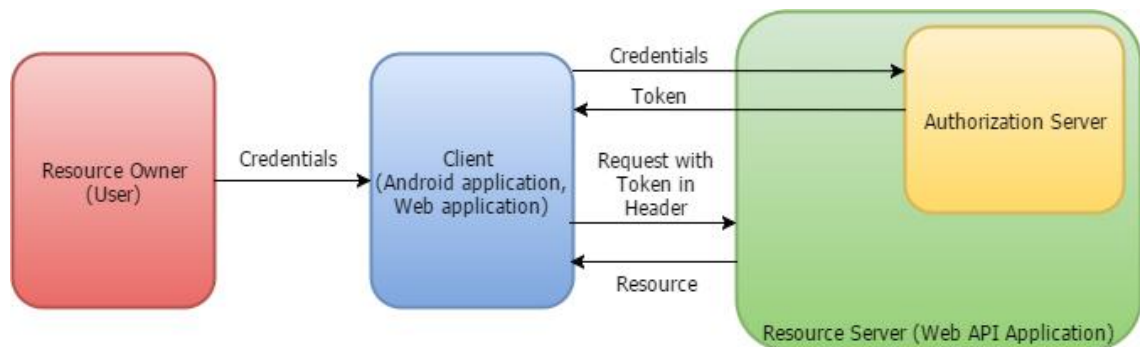
Obrázek 7.2.1: ER diagram databáze

7.2.3 Autorizace

Pro autorizaci uživatelů je použit modul *ASP.NET Identity*. Tento modul se stará o konkrétní přihlašování, pro ukládání informací je použit výchozí *Entity Framework provider*, který spravuje databázi, řeší ukládání a načítání informací o uživatelích. Implementace konkrétních funkcí jako je například registrace nebo načtení rolí uživatele je ve třídě *AuthRepository*.

Přihlašování z mobilní nebo ze serverové části aplikace probíhá následujícím způsobem. Ve třídě *AuthorizationServerProvider* je ve funkci *GrantResourceOwnerCredentials* zjištěno, jestli požadavek pochází z mobilní nebo serverové části aplikace. Dle uživatelského jména, role a původu požadavku je ověřeno, zda má uživatel práva se přihlásit. V případě úspěchu je vytvořen objekt *ClaimsIdentity*, který modul *ASP.NET*

Identity uloží do databáze pro následné ověřování jednotlivých API volání, a klientovy je vrácen autorizační token. Token je pak uložen do Cookies prohlížeče a v dalších voláních se pomocí filtru *RequireLogin* ověřuje, zda je přítomen (po vypršení platnosti se smaže). Tento proces (u mobilní aplikace se token ukládá do paměti aplikace) používají obě části aplikace, protože i u serverové části je přihlašování prováděno asynchronně pomocí Javascriptu. Proces je znázorňuje Obrázek 7.2.2.



Obrázek 7.2.2: Diagram autorizace požadavku

Díky použití *ASP.NET Identity* je možné v budoucnu systém rozšířit o přihlašování pomocí služeb třetích stran jako je např. Facebook, Google nebo OpenID.

7.2.4 Spouštění skriptů

Při voláních, které zajišťují správu serveru je použit *PscriptsController*, který obsahuje metody *RunScript* a *RunScriptPost*. První metoda zpracovává požadavky poslané metodou GET a druhá metodou POST. To jaká metoda se použije je definováno v mobilní části aplikace. Metoda GET je používána pro spouštění skriptů, které nevyžadují žádné vstupní parametry u metody POST naopak pro ty, které vstupní parametry vyžadují. Obě

```

[HttpGet]
public string RunScript(string param) {
    var ps = new PscriptsServices();
    return
    ps.RunScript(param, this.Request.GetQueryNameValuePairs().GetEnumerator());
}

[HttpPost]
public string RunScriptPost(string param, FormCollection formdata) {
    IDictionary<KeyValuePair<string, string>> valueMap
    =WebAPIUtils.Convert(formdata);
    var ps = new PscriptsServices();
    return ps.RunScript(param, valueMap);
}
  
```

Kód 7.2.1: Metody *RunScript* a *RunscriptPost* v *PscriptsController*

metody obsahují parametr *param*, který slouží pro identifikaci spouštěného skriptu. Implementaci metod viz Kód 7.2.1.

Identifikátor skriptu a případně parametry jsou pak předány metodě *RunScript* ze třídy *PscriptsServices*. Tato třída má na starost veškerou práci se skripty – načtení, identifikování, spuštění a zpracování do výstupního formátu JSON. V metodě *RunScript* je nejprve zjištěno, zda se identifikátor spouštěného skriptu nachází v konfiguračním souboru skriptů. Pokud ano, je z tohoto souboru načtena cesta ke skriptu. Ta je pak předána společně s parametry, se kterými se má skript spouštět, do vytvořeného běhového prostředí PowerShellu. To se vytváří ze třídy *PowerShell* z knihovny *System.Management.Automation*. Po té je skript spuštěn. V případě výskytu *Terminating* chyby je tato chyba zachycena výjimkou *RuntimeException*. Při výskytu *Nonterminating* chyby je tato chyba zachycena v objektu třídy *PowerShell.Streams.Error*. V obou případech je vytvořena chybová odpověď, která je vrácena klientovi. Při standardním průběhu, je výstup serializován do JSON formátu a odeslán klientovi.

Validace vstupních parametrů skriptů je řešena až na úrovni skriptů. Tento návrh přispívá ke snadné rozšiřitelnosti o další skripty, kdy se ve webové aplikaci nemusí řešit validace pro každý skript zvlášť, ale využívá se k tomu možností PowerShellu.

Skripty jsou uloženy ve složce *App_Data* (konkrétně v podadresáři *PScripts*) což je standardní umístění ASP.NET webových aplikací určené pro ukládání dat, které nemají být přístupné z webu. Dále je zde uložen konfigurační soubor *pscripts.xml*, který obsahuje identifikátor skriptu (pod jakým jménem bude přístupný přes API) a konkrétní cestu.

Rozšíření možností správy o novou funkcionalitu je jednoduchá procedura kdy stačí vytvořit nový PowerShell skript, umístit jej do adresáře *PScripts* (nebo i jinde) a do konfiguračního souboru *pscripts.xml* přidat zvolený identifikátor a cestu ke skriptu. Při volání url s příslušným identifikátorem nového skriptu se pak skript standardně spustí a není potřeba implementovat nic dalšího.

7.2.5 Logování

Pro budoucí dohledávání změn, které byly v systému provedeny pomocí API, je v aplikaci implementováno logování. Každé volání jakékoliv akce v API kontrolerech je uloženo a je možné si ho prohlédnout pomocí webového rozhraní. Ukládány jsou

informace jako je jméno uživatele, který volání provedl, čas, kontroler, název akce, název skriptu, parametry, odpověď serveru, IP adresa požadavku a další.

O zachytávání požadavků se stará třída *ApiLogHandler* umístěná ve složce *Filters*. Požadavky jsou ukládány do adresáře *Log* umístěného ve složce *App_Data*. Ukládají se do textových souborů po dnech to znamená, že všechny požadavky provedené například 1.6.2016 budou uloženy v souboru *Log/01-06-2016.txt*. Požadavky jsou uloženy v textové formě ve formátu JSON, což umožňuje pozdější snadné zpracování.

O zobrazení zalogovaných požadavků se stará třída *LogController* respektive příslušný *View* (*View/Log/Index.cshtml*). *LogController* poskytuje metodu *getLog()*, která pomocí zadaného parametru (datum) načte příslušný soubor s logy a vrátí ho do *View* kde je odpověď zpracována v Javascriptu a jednotlivé logy zobrazeny v tabulce. Při kliknutí na log je zobrazeno více detailů, neprovádí se již žádné volání metod, metoda *getLog()* vrací veškerá data. To, které informace se mají zobrazovat již zajišťuje logika v Javascriptu. Logy v tabulce lze řadit a filtrovat podle různých parametrů, tuto funkcionalitu zajišťuje použitý plugin *DataTables* pro knihovnu *jQuery*.

7.2.6 Statistiky

Pro zobrazení souhrných informací o zalogovaných požadavcích slouží sekce *Statistics*. *View* je uložen ve stejné složce jako u logů (*View/Log/Stats.cshtml*). Stejně tak pro zpracování dat se využívá *LogController*, ale již jiná metoda (*getLogs()*). Ta jako vstupní parametr používá pole datumů. Metoda načte všechny logy dle vstupních datumů, deserializuje data a provádí nad nimi agregační LINQ²⁸ dotazy. Výstup je pak serializován do formátu JSON a poslán do *View*, kde je zpracován pomocí Javascriptu. Statistika jsou zobrazovány formou sloupcových a koláčových grafů. Pro vykreslení je použita Javascript knihovna *Chart.js*. Pro výběr datumů jsou použity kalendáře z knihovny *jQuery UI*. Při práci s datumem je pro usnadnění formátování použita knihovna *Moment.js*.

7.2.7 Správa uživatelů

Pro správu uživatelů a jejich rolí slouží část *Users*. Pro zpracování dat využívá *UsersController* a pro provádění konkrétních akcí (například vytvoření nebo získání

²⁸ LINQ – dotazovací jazyk integrovaný do C# (z anglického Language INtegrated Query)

informací o uživateli z databáze) pak hlavně metody třídy *AuthRepository*. Je implementována pomocí několika *Views*:

- **Index** – zobrazuje všechny uživatele v tabulce. Umožňuje filtrovat a řadit podle libovolných parametrů, obsahuje odkazy na bližší informace o uživateli, na správu rolí a také na vytvoření nového uživatele.
- **User** – zobrazuje detail uživatele a umožňuje změnit uživateli heslo (pokud není v roli supervisor).
- **Create** – zobrazuje formulář pro vytvoření uživatele. Model včetně parametrů validace je implementován v modelové třídě *UserRegModel*.
- **Roles** – zobrazuje informace o členství uživatele v rolích a formuláře pro vymazání a přidání do rolí. Uživatel nemůže sám sebe vymazat z role supervisor.

7.3 Powershell skripty

Třetí částí systému jsou PowerShell skripty, které provádí správu. Standardně jsou umístěny ve složce *App_Data* webové aplikace. Konfiguračním souborem kde se skripty registrují pro zpracování webovou aplikací je soubor *pscripts.xml* v *App_Data*. Ukázka konfiguračního souboru viz Kód 7.3.1. Jednotlivé skripty jsou obaleny elementem *Pscript*, který obsahuje definici jména služby (parametr *Name*), pod kterou bude dostupná v API a cestu ke skriptu v rámci souborového systému v elementu *Path*.

```
<Pscripts>
  <Pscript>
    <Name>createUser</Name>
    <Path>\PScripts\createUser.ps1</Path>
  </Pscript>
  <Pscript>
    <Name>disableUser</Name>
    <Path>\PScripts\disableUser.ps1</Path>
  </Pscript>
  ...
</Pscripts>
```

Kód 7.3.1: Konfigurační soubor pro registraci skriptů do webové aplikace

7.3.1 Činnosti správy

Na základě vybraných činností správy definovaných v kapitole 4, byly vytvořeny následující skripty.

Správa účtů v Active Directory:

1. **createUser** – vytvoření nového uživatele v Active Directory. Vstupními parametry jsou jméno uživatele, login a heslo. Úspěšný průběh skriptu vytvoří nové uživatelské konto a vrátí prázdnou odpověď. Prázdná odpověď je v PowerShellu obvyklý výstup u cmdletů, které vytváří, mažou nebo jinak mění stav některé části systému. Proto je u takových příkazů prázdný výstup žádoucí. Při výskytu chyb naopak vrací identifikátor a popis chyby. Stejný přístup je použit i v dalších skriptech.
2. **disableUser** – zakázání uživatelského konta v Active Directory. Vstupní parametr je login uživatele. Úspěšný průběh skriptu zakáže uživatelské konto a vrátí prázdnou odpověď.
3. **getInactive** – získá z Active Directory uživatele a stanice, kteří se po dobu alespoň 30 dnů nepřihlásily do systému. Úspěšný průběh skriptu vrací seznam uživatelů a stanic s názvem, datem posledního přihlášení a typem (uživatel nebo stanice).
4. **inactiveUsers** – získá z Active Directory pouze uživatele, kteří se po dobu alespoň 30 dnů nepřihlásily do systému. Úspěšný průběh skriptu vrací seznam uživatelů s názvem a datem posledního přihlášení.
5. **inactiveComputers** – získá z Active Directory pouze stanice, ke kterým se po dobu alespoň 30 dnů nikdo nepřihlásil. Úspěšný průběh skriptu vrací seznam stanic s názvem a datem posledního přihlášení.
6. **getUser** – získá z Active Directory informace o uživateli. Vstupním parametrem je login uživatele. Úspěšný průběh skriptu vrací informace o uživateli, mezi nimiž je login, jméno, zda je účet aktivní, datum vytvoření účtu, datum posledního přihlášení, datum poslední změny hesla apod.
7. **getUserMembership** – získá z Active Directory informace o tom jakých skupin je uživatel členem. Vstupním parametrem je login uživatele. Úspěšný průběh skriptu vrací seznam skupin.

8. **getUsers** – získá z Active Directory seznam všech uživatelských účtů. Úspěšný průběh skriptu vrací seznam uživatelů s loginem, jménem a parametrem zda je účet aktivní.

Správa služeb a procesů:

1. **getProcess** – získá informace o procesu. Vstupním parametrem je id procesu. Úspěšný průběh skriptu vrací informace o procesu - jméno, popis, čas spuštění, využití procesorového času od spuštění procesu, využití paměti a zda proces odpovídá.
2. **stopProcess** – zastaví daný proces. Vstupním parametrem je id procesu. Úspěšný průběh skriptu zastaví proces a vrátí prázdnou odpověď.
3. **getProcesses** – získá seznam aktuálně běžících procesů v systému. Úspěšný průběh skriptu vrací seznam procesů s informacemi – id, jméno, využití procesoru od spuštění, popis a využití paměti.
4. **getService** – získá informace o službě poskytované systémem. Vstupním parametrem je název služby. Úspěšný průběh skriptu vrací informace o službě - jméno, zobrazované jméno (kompletní název), způsob spuštění a stav služby.
5. **getServices** - získá seznam aktuálně dostupných služeb v systému. Úspěšný průběh skriptu vrací seznam služeb s informacemi – jméno, způsob spuštění a stav služby.
6. **startService** – spustí danou službu. Vstupním parametrem je název služby. Úspěšný průběh skriptu spustí službu a vrátí prázdnou odpověď.
7. **stopService** – zastaví danou službu. Vstupním parametrem je název služby. Úspěšný průběh skriptu zastaví službu a vrátí prázdnou odpověď.
8. **restartService** – restartuje danou službu. Vstupním parametrem je název služby. Úspěšný průběh skriptu restartuje službu a vrátí prázdnou odpověď.

Správa úložiště:

1. **showDiskSpace** – získá informace o kapacitách úložišť. Vrací informace o názvu jednotek, volné, využitě a celkové kapacitě diskových jednotek.

Je samozřejmě možné vytvářet další skripty a rozšiřovat tak možnosti systému. Vytvářený skript by měl obsahovat definici a validaci vstupních parametrů pokud to tvůrce skriptu

uzná za vhodné. Soubor skriptu se pak umístí do složky webové části aplikace *App_Data/PScripts*. Po té, se zaregistruje název a cesta ke skriptu do konfiguračního souboru *pscripts.xml* (ukázka na začátku této kapitoly). Nyní bude skript již spustitelný přes webové rozhraní.

8 Testování

Vývoj webové části aplikace probíhal ve vývojovém prostředí Visual Studio 2015 Community edition. Aplikace byla nejprve vyvíjena a testována pouze na lokálním stroji, pro verzování zdrojového kódu byl použit systém Git. Jakmile to bylo možné (byla hotová kostra systému, umožňující pomocí URL adresy spustit skript), začalo se testovat i na produkčním serveru. Produkční server byla virtuální stanice se systémem Windows Server 2012R2. Projekt byl automaticky publikován přímo z vývojového prostředí pomocí FTP²⁹ protokolu do složky na produkčním serveru, odkud se již automaticky načetl do webového kontejneru služby IIS. Testování webového rozhraní pak probíhalo z internetového prohlížeče. Testování webového API pak pomocí rozšíření Postman pro prohlížeč Chrome, což umožňovalo snadné odesílání, úpravu a kontrolu příslušných požadavků a odpovědí aplikace. Ve chvíli kdy byla implementována funkcionality volání služeb do mobilní aplikace, tak pak i z mobilní aplikace.

Vývoj skriptů probíhal v prostředí PowerShell ISE. Zpočátku byly vytvářeny skripty na lokálním PC se systémem Windows 7. Na tomto systému nebyly aktivní služby Active Directory, proto byly vyvíjeny pouze skripty týkající se procesů, služeb a úložišť. V pozdější fázi se přešlo na vývoj na produkčním serveru, kde již bylo Active Directory aktivní, a mohlo se tak využívat příkazů a cmdletů, které pracují s Active Directory. Testování a ověřování správnosti výstupu probíhalo kontrolou výstupů skriptů s některým z nástrojů dostupných v systému (Active Directory Administrative Center, Task Manager, Computer Management). Například porovnání výstupu skriptu pro získání běžících procesů s běžícími procesy ve správci úloh nebo vytvoření nového uživatele v Active Directory pomocí skriptu, a následné přihlášení do systému pomocí nově vytvořeného účtu.

Mobilní část aplikace byla vyvíjena ve vývojovém prostředí Android Studio. Stejně jako pro serverovou část byl použit verzovací systém Git. Aplikace pak byla průběžně testována na zařízeních:

- Mobilní telefon DOOGEE X5 se systémem Android 5.1 (API 22) s úhlopříčkou 5“ s rozlišením 1280x720 pixelů.

²⁹ FTP – síťový protokol pro přenos souborů (z anglického File Transfer Protocol).

- Tablet BN NookHD+ se systémem Android 4.4.4 (API 19) s úhlopříčkou 9“ s rozlišením 1920x1080 pixelů.
- Emulátor Android 4.4.2 (API 19) s úhlopříčkou 4“ s rozlišením 800x480 pixelů.
- Emulátor Android 6.0 (API 23) s úhlopříčkou 4“ s rozlišením 800x480 pixelů.

Na všech zařízeních byla aplikace spustitelná a prováděla to, co se od ní očekávalo. Subjektivně se zdá nejvhodnější používat aplikaci na displejích s úhlopříčkou 4“ – 5“, u displejů s úhlopříčkou 6“ a větších se objevuje „nevyužité místo“. Primárně je ale aplikace navrhována pro mobilní telefony, které se v těchto úhlopříčkách příliš nevyskytují. Aplikace by měla fungovat na všech verzích systému s API 19 a větší což je aktuálně 77,1% všech zařízení se systémem android [24]. Kompilace aplikace probíhala s nejaktuálnějším API 23.

Testování probíhalo na lokální Wi-fi síti, kde byl virtuální server připojen tak, jako kdyby se jednalo o skutečnou stanici a k ní se připojovali mobilní zařízení stejně, jako by tomu bylo, pokud by byl server umístěn v internetu s veřejnou adresou. Vzdáleně také probíhalo testování webového rozhraní. Kromě standartní funkčnosti bylo testováno i chování aplikace v případě výpadků internetového spojení a zastavení serverové části aplikace.

8.1 Testovací případy – mobilní aplikace

Zde jsou uvedeny testovací případy, které byly používány pro otestování funkčnosti mobilní aplikace. Testování probíhalo z mobilní aplikace. Mimo uvedené scénáře byly také testovány případy jako je zadání chybného hesla při přihlašování, neexistujícího uživatele v Active Directory apod. Tyto případy nejsou dále explicitně uvedeny, protože jejich chování je podobné – systém na ně reaguje vypsáním konkrétní chyby pomocí dialogu.

Název	Zobrazení uživatele
Podmínky	Otevřena sekce User, uživatel existuje v Active Directory
Kroky	Vyplnit login uživatele, kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí informace o uživateli

Název	Deaktivace uživatele
Podmínky	Otevřena sekce User, uživatel existuje v Active Directory
Kroky	Vyplnit login uživatele, kliknout na tlačítko Disable
Výsledek Android	Ve výstupu se zobrazí informace o uživateli
Výsledek Systém	Uživatel je deaktivován a k účtu se nelze přihlásit

Název	Vytvoření uživatele
Podmínky	Otevřena sekce Create User, uživatelský login neexistuje v Active Directory
Kroky	Vyplnit login uživatele, jméno a heslo, kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí informace o uživateli
Výsledek Systém	Vytvořen nový uživatelský účet v Active Directory

Název	Zobrazení uživatelů
Podmínky	Otevřena sekce Users, uživatelé existují v Active Directory
Kroky	Kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí seznam uživatelů s indikací stavu

Název	Zobrazení kapacity úložišť
Podmínky	Otevřena sekce Drives
Kroky	Kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí grafy kapacit diskových jednotek

Název	Zobrazení dostupných služeb
Podmínky	Otevřena sekce Services
Kroky	Kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí seznam dostupných služeb s indikací stavu

Název	Zobrazení informací o službě
Podmínky	Otevřena sekce Service
Kroky	Kliknout na vybranou službu v seznamu služeb
Výsledek Android	Ve výstupu se zobrazí informace o službě

Název	Zastavení služby
Podmínky	Otevřena sekce Service, služba je spuštěna
Kroky	Kliknout na tlačítko Stop
Výsledek Android	Ve výstupu se zobrazí informace o službě
Výsledek Systém	Služba je zastavena

Název	Spuštění služby
Podmínky	Otevřena sekce Service, služba není spuštěna
Kroky	Kliknout na tlačítko Start
Výsledek Android	Ve výstupu se zobrazí informace o službě
Výsledek Systém	Služba je spuštěna

Název	Restartování služby
Podmínky	Otevřena sekce Service, služba je spuštěna
Kroky	Kliknout na tlačítko Restart
Výsledek Android	Ve výstupu se zobrazí informace o službě
Výsledek Systém	Služba je spuštěna

Název	Zobrazení běžících procesů
Podmínky	Otevřena sekce Processes
Kroky	Kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí seznam běžících procesů

Název	Zobrazení informací o procesu
Podmínky	Otevřena sekce Process
Kroky	Kliknout na vybraný proces v seznamu procesů
Výsledek Android	Ve výstupu se zobrazí informace o procesu

Název	Zastavení procesu
Podmínky	Otevřena sekce Process
Kroky	Kliknout na tlačítko Stop
Výsledek Android	Ve výstupu se zobrazí informace o ukončení procesu
Výsledek Systém	Proces neběží

Název	Zobrazení neaktivních účtů
Podmínky	Otevřena sekce Inactive, existují uživatelské účty a stanice ke kterým se po dobu 30 dní nikdo nepřihlásil
Kroky	Kliknout na tlačítko Run
Výsledek Android	Ve výstupu se zobrazí seznam neaktivních účtů a stanic

8.2 Testovací případy – webové rozhraní

Zde jsou uvedeny testovací případy použité pro otestování webového rozhraní. Testování probíhalo z webového prohlížeče. Mimo uvedené scénáře byly také testovány případy jako je zadání chybného hesla při přihlašování, odebrání uživatele z role ve které není apod. Tyto případy nejsou dále explicitně uvedeny, protože jejich chování je podobné – systém na ně reaguje vypsáním konkrétní chyby na webové stránce.

Název	Vytvoření uživatele
Podmínky	Otevřena sekce Create New User, uživatel neexistuje
Kroky	Vyplnit uživatelské jméno a heslo, kliknout na tlačítko Create
Výsledek	Vytvořený uživatel se může přihlásit do systému pomocí mobilní aplikace, je zobrazen v seznamu uživatelů

Název	Změna hesla uživatele
Podmínky	Otevřena sekce User, uživatel není v roli supervisor (mimo aktuálně přihlášeného uživatele)
Kroky	Vyplnit nové heslo, kliknout na tlačítko Reset password
Výsledek	Vytvořený uživatel se může přihlásit do systému pomocí mobilní aplikace pomocí nového hesla

Název	Přidání uživatele do role
Podmínky	Otevřena sekce Manage User Roles, uživatel není v roli
Kroky	Vyplnit uživatelské jméno, vybrat roli ze seznamu, kliknout na tlačítko Add to role
Výsledek	Uživatel je přidán do role a může se přihlásit k mobilní aplikaci (role admin) nebo do webového rozhraní (role supervisor)

Název	Odebrání uživatele z role
Podmínky	Otevřena sekce Manage User Roles, uživatel je v roli
Kroky	Vyplnit uživatelské jméno, vybrat roli ze seznamu, kliknout na tlačítko Delete from role
Výsledek	Uživatel je odebrán z role a nemůže se přihlásit k mobilní aplikaci (odebrána role admin) nebo do webového rozhraní (odebrána role supervisor)

Název	Zjištění logů
Podmínky	Otevřena sekce Logs, existuje log
Kroky	Vybrat datum logu kliknutím na den v kalendáři
Výsledek	V tabulce je zobrazen seznam Api volání pro zadaný datum

Název	Zjištění podrobných informací o volání
Podmínky	Otevřena sekce Logs, existuje log, je zobrazen seznam volání v tabulce
Kroky	Vybrat volání, kliknutí na příslušné volání v tabulce
Výsledek	Zobrazení podrobných informací o volání ve výstupu Call details

Název	Zjištění statistických informací
Podmínky	Otevřena sekce Statistics, existují logy pro zadané období
Kroky	Vybrat rozsah datumů kliknutím na dny v kalendáři From a To
Výsledek	Zobrazení agregovaných informací v grafech

8.3 Testovací případy – skripty

Zde jsou uvedeny testovací případy pro PowerShell skripty. Testování probíhalo spuštěním skriptů z konzole a kontrolou výstupu. Mimo uvedené případy byly testovány také případy, jako je zadání chybných parametrů, spuštění skriptu na systému, který nepodporuje použité cmdlety apod. Tyto případy nejsou dále explicitně uvedeny, protože jejich chování je podobné – systém na ně reaguje vypsáním konkrétní chyby do konzole.

Název	Zobrazení informací o uživateli
Podmínky	Běžící služba Active Directory, uživatel existuje
Kroky	Spustit skript getUser s parametrem login
Výsledek	Zobrazení informací o uživateli

Název	Zobrazení informací o členství uživatele ve skupinách
Podmínky	Běžící služba Active Directory, uživatel existuje
Kroky	Spustit skript getUserMembership s parametrem login
Výsledek	Zobrazení seznamu skupin, kterých je uživatel členem

Název	Zobrazení informací o uživateli
Podmínky	Běžící služba Active Directory, uživatelé existují
Kroky	Spustit skript getUsers
Výsledek	Zobrazení seznamu uživatelů

Název	Vytvoření uživatele
Podmínky	Běžící služba Active Directory, uživatel neexistuje
Kroky	Spustit skript createUser s parametry jméno uživatele, login a heslo
Výsledek	Uživatel je vytvořen v Active Directory

Název	Deaktivace uživatele
Podmínky	Běžící služba Active Directory, uživatel existuje
Kroky	Spustit skript disableUser s parametrem login uživatele
Výsledek	Uživatel je deaktivován v Active Directory a k účtu se není možné přihlásit

Název	Zobrazení neaktivních uživatelů
Podmínky	Běžící služba Active Directory, existují neaktivní uživatelé a stanice
Kroky	Spustit skript getInactive
Výsledek	Zobrazení seznamu uživatelů a stanic ke kterým se nikdo nepřihlásil po dobu alespoň 30 dnů

Název	Zobrazení informací o procesu
Podmínky	Proces je spuštěn
Kroky	Spustit skript getProcess s parametrem id procesu
Výsledek	Zobrazení informací o běžícím procesu

Název	Zastavení procesu
Podmínky	Proces je spuštěn
Kroky	Spustit skript stopProcess s parametrem id procesu
Výsledek	Proces je neběží

Název	Zobrazení informací o procesech
Podmínky	Procesy jsou spuštěny
Kroky	Spustit skript getProcesses
Výsledek	Zobrazení seznamu běžících procesů

Název	Zobrazení informací o službě
Podmínky	Služba existuje
Kroky	Spustit skript getService s parametrem název služby
Výsledek	Zobrazení informací o službě

Název	Zobrazení informací o službách
Podmínky	Služby existují
Kroky	Spustit skript getServices
Výsledek	Zobrazení seznamu dostupných služeb

Název	Zobrazení informací o diskových jednotkách
Podmínky	Existují diskové jednotky
Kroky	Spustit skript showDiskSpace
Výsledek	Zobrazení seznamu jednotek s informacemi o využití kapacity

Název	Spuštění služby
Podmínky	Služba existuje a není spuštěna
Kroky	Spustit skript startService s parametrem název služby
Výsledek	Služba je spuštěna

Název	Zastavení služby
Podmínky	Služba existuje a je spuštěna
Kroky	Spustit skript stopService s parametrem název služby
Výsledek	Služba je zastavena

9 Možnosti rozšíření

System je vzhledem k návrhu serverové části připraven na rozšíření možností správy vytvořením dalších PowerShell skriptů a jejich registraci v konfiguračním souboru. Touto jednoduchou procedurou se automaticky zpřístupní nová služba ve webovém API. Protože serverová část poskytuje unifikované webové API přístupné pomocí HTTP protokolu, je možné implementovat správu také do dalších zařízení, jako jsou mobilní zařízení se systémem iOS nebo Windows Phone, ale také například do webových aplikací využívajících Javascript. U serverové části je prostor pro rozšíření například u statistik, kde by bylo vhodné sledovat, jak dlouho trvají konkrétní požadavky. Také by mohlo být vhodné vytvořit systém pro export grafů statistik.

Aplikace by také mohla implementovat systém elektronické pošty ať už s vlastním konfigurovaným mailovým serverem nebo s využitím některé služby třetích stran. Bylo by pak možné posílat supervizorům různé statistické informace, ale také zapomenutá hesla. Tato funkcionality by našla využití i v mobilní aplikaci, kdy by se například nově vytvářenému uživateli v Active Directory pomocí serverového API poslal mail s informacemi o novém účtu.

10 Závěr

Cílem této práce bylo vytvořit systém, který bude schopen pomocí mobilního zařízení a PowerShellu provádět správu vybraných činností na vzdálené stanici se systémem Windows Server.

V práci byla nejprve nastíněna motivace a přínosy tohoto systému. Po té byly prozkoumány možnosti PowerShellu pro správu. Následně byly vybrány činnosti, které by měl systém umět spravovat. Vzhledem k vybraným činnostem, omezením a zadání byly po té specifikovány požadavky na systém. Po specifikaci požadavků na systém došlo na konkrétní návrh všech částí systému. Dále byla v práci popsána implementace navrženého systému a následně popsány způsoby testování a vývoje. Na konec byly zmíněny možnosti rozšíření.

Výsledkem práce je systém, který umožňuje pomocí Android aplikace spravovat vzdálený server. Využívá k tomu API dostupné přes webové rozhraní. Webová část aplikace pak zajišťuje správu systému spouštěním příslušných skriptů. Nad rámec zadání bylo do webové části systému integrováno logování API požadavků a zobrazování statistik.

Protože systém nabízí unifikované API, je možné v budoucnu snadno integrovat klientské aplikace pro další mobilní platformy jako je iOS nebo Windows Phone a rozšiřovat tak použitelnost systému. Také je možné snadno rozšiřovat možnosti správy vytvářením nových skriptů a jejich registrací do API.

Seznam zkratek

V práci byly použity následující zkratky:

- .NET – platforma od společnosti Microsoft
- Aktivita – hlavní komponenta Android aplikace
- API – Application Programming Interface
- ASP – Active Server Pages
- C# - programovací jazyk
- CIM – Common Information Model
- Cmdlet – Speciální příkaz v PowerShellu
- COM – Component Object Model
- Cookies – data ukládaná do úložiště webového prohlížeče
- CSS – Cascading Style Sheets
- DCOM – Distributed Component Object Model
- DLL – Dynamic link library
- DNS – Domain Name System
- Fragment – komponenta Android aplikace
- FTP – File Transfer Protocol
- GET – metoda HTTP protokolu
- Git – verzovací systém
- GUI – Graphical User Interface
- HTML – Hypertext Markup Language
- HTTP – Hypertext Transfer Protocol
- HTTPS – Hypertext Transfer Protocol Secure
- Chrome – webový prohlížeč
- I/O – Input / Output
- IDE – Integrated Development Environment
- IIS – Internet Information Services
- IP – Internet Protocol
- ISE – Integrated Scripting Environment
- Java – programovací jazyk
- Javascript – programovací jazyk

- JSON – Javascript Object Notation
- LDAP - Lightweight Directory Access Protocol
- LINQ – Language Integrated Query
- MS-DOC – Microsoft disk operating system
- MSH – Monad shell
- MSSQL Server – Microsoft SQL Server
- MVC – Model View Controller
- OOP – Object oriented programming
- PHP – PHP: Hypertext Preprocessor
- POJO – Plain Old Java Object
- POST – metoda HTTP protokolu
- PSSnapin – knihovna v technologii .NET
- Python – programovací jazyk
- RPC – Remote Procedure Call
- SOA – Service Oriented Architecture
- SOAP – Simple Object Access Protocol
- UNIX – operační systém
- URI – Uniform Resource Identifier
- URL – Uniform Resource Locator
- WinRM – Windows Remote Management
- WMI – Windows Management Instrumentation
- WS-Management – Web Services Management
- XML – Extensible Markup Language

Literatura

- [1] LEE, Thomas et al. *Windows PowerShell 2.0 Bible*. Indianapolis: John Wiley & Sons, Inc., 2011. ISBN 978-1-118-02198-9.
- [2] HOLMES, Lee. *Windows PowerShell Cookbook*. Third Edition. Sebastopol: O'Reilly Media Inc., 2013. ISBN 978-1-449-32068-3.
- [3] SNOVER, Jeffrey P. *Monad Manifesto* [online]. [cit. 2016-04-05] Dostupné z: <http://www.jsnover.com/Docs/MonadManifesto.pdf>.
- [4] *Pipes: A Brief Introduction* [online]. [cit. 2016-04-13] Dostupné z: <http://www.linfo.org/pipes.html>.
- [5] FINKE Douglas. *Windows PowerShell for Developers*. First Edition. Sebastopol: O'Riley Media Inc., 2012. ISBN 978-1-449-32270-0.
- [6] JONES, Don, SIDDAWAY, Richard, HICKS, Jeffery. *PowerShell in Depth: An administrator's guide*. Shelter Island, NY: Manning Publications Co., 2013. ISBN 978-1-617-29055-8.
- [7] GEIER, Eric. *Powerfull PowerShell Tools and Utilities* [online]. [cit. 2016-04-20] Dostupné z: <http://www.serverwatch.com/server-tutorials/slideshows/6-powerful-powershell-tools-utilities.html>.
- [8] WILSON, Ed. *PowerShell: Průvodce skriptováním*. 1. vyd. Brno: Computer Press, 2015. ISBN 978-80-251-4386-5
- [9] MORAVEC, David. *Server Management & PowerShell* [online]. [cit. 2016-04-22] Dostupné z: <https://blogs.technet.microsoft.com/technetczsk/2013/08/08/tden-s-windows-server-2012-r2-server-management-powershell/>.
- [10] RICCIONI, Michael. *Upgrading Your Skills to MCSA Server 2012 – PowerShell V3.0 Part 2 (70-417)* [online]. [cit. 2016-04-20] Dostupné z: <http://www.michaelriccioni.com/upgrading-your-skills-to-mcsa-server-2012-powershell-v3-0-part-2-70-417/>.

- [11] WELTNER Tobias. *Chapter 3.Variables - Master-PowerShell* [online]. [cit. 2016-04-23] Dostupné z:
<http://powershell.com/cs/blogs/ebookv2/archive/2012/03/10/chapter-3-variables.aspx>.
- [12] *PowerShell Documentation about_Scopes* [online]. [cit. 2016-04-23] Dostupné z:
<https://technet.microsoft.com/en-us/library/hh847849.aspx>.
- [13] *PSClass - Object Oriented Scripting in Powershell* [online]. [cit. 2016-04-23] Dostupné z: <https://psclass.codeplex.com/>.
- [14] SCHWICHTENBERG, Holger. *Essential PowerShell*. First printing. Boston: Pearson's Education Inc., 2008. ISBN 978-0-672-32966-1.
- [15] MORAVEC, David. Skripty a moduly. *Seriál Windows PowerShell* [online]. Část 13. [cit. 2016-04-23] Dostupné z:
<https://blogs.technet.microsoft.com/technetczsk/2010/10/12/seril-windows-powershell-skripty-a-moduly-st-13/>.
- [16] DRISCOLL, Adam. *Microsoft Windows PowerShell 3.0 First Look*. First publishing. Birmingham: Pack Publishing Ltd., 2012. ISBN 978-1-84968-644-0.
- [17] STANEK, Wiliam R. *Windows PowerShell 2.0:Administrator's Pocket Consultant*. First edition. Redmond: Microsoft Press, 2009. ISBN 978-0-73562-595-2.
- [18] SEGUIS, Steve. *Windows PowerShell 2 For Dummies*. Hoboken: Wiley Publishing, Inc., 2009. ISBN: 978-0-470-37198-5.
- [19] *WS-Management Protocol* [online]. [cit. 2016-04-25] Dostupné z:
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa384470\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa384470(v=vs.85).aspx).
- [20] *WMI Architecture* [online]. [cit. 2016-04-25] Dostupné z:
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa394553\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa394553(v=vs.85).aspx).

- [21] *Win32 Classes* [online]. [cit. 2016-04-25] Dostupné z:
[https://msdn.microsoft.com/cs-cz/library/aa394084\(v=vs.85\).aspx](https://msdn.microsoft.com/cs-cz/library/aa394084(v=vs.85).aspx).
- [22] STANEK, Wiliam R. *Active Directory Kapesní rádce administrátora*. 1. vyd.
Brno: Computer Press a.s., 2009. ISBN 978-80-251-2555-7.
- [23] *Get Started with ASP.NET* [online]. [cit. 2016-04-25] Dostupné z: Available:
<http://www.asp.net/get-started>.
- [24] *Dashboards / Android Developers* [online]. [cit. 2016-05-28] Dostupné z:
<https://developer.android.com/about/dashboards/index.html>.
- [25] *The Component Object Model* [online]. [cit. 2016-05-10] Dostupné z:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363(v=vs.85).aspx).
- [26] GORTON, Ian. *Essential Software Architecture*. Second Edition. New York:
Springer, 2011. ISBN 978-3-642-19175-6.

Přílohy

A Popis webového API

V této příloze je uveden výpis webových služeb, které poskytuje serverová část aplikace. Jednotlivá volání vyžadují autorizaci, která je zajištěna pomocí tokenu, který se posílá v hlavičce požadavku jako parametr *Authorization* s hodnotou „*Bearer <token>*”.

Volání služeb správy začíná tvarem:

https://<adresa_serveru:port>/<aplikace>/api/pscripts/

URI	/runscriptpost/createUser
Metoda	POST
Parametry	Name [string], Login [string], Password [string]
Popis	Slouží k vytvoření uživatele v Active directory

URI	/runscript/getInactive
Metoda	GET
Parametry	
Popis	Vrací neaktivní uživatele a stanice po dobu více jak 30 dní

URI	/runscriptpost/getProcess
Metoda	POST
Parametry	Id [string]
Popis	Vrací informace o procesu

URI	/runscript/getProcesses
Metoda	GET
Parametry	
Popis	Vrací aktuálně běžící procesy

URI	/runscriptpost/getService
Metoda	POST
Parametry	Name [string]
Popis	Vrací informace službě

URI	/runscript/getServices
Metoda	GET
Parametry	
Popis	Vrací seznam dostupných služeb

URI	/runscriptpost/getUser
Metoda	POST
Parametry	User [string]
Popis	Vrací informace o uživateli z Active Directory

URI	/runscriptpost/getUserMembership
Metoda	POST
Parametry	Name [string]
Popis	Vrací informace o členství uživatele ve skupinách Active Directory

URI	/runscript/getUsers
Metoda	GET
Parametry	
Popis	Vrací informace o uživateli z Active Directory

URI	/runscriptpost/restartService
Metoda	POST
Parametry	Name [string]
Popis	Restartuje danou službu

URI	/runscript/showDiskSpace
Metoda	GET
Parametry	
Popis	Vrací informace o využití úložišť

URI	/runscriptpost/startService
Metoda	POST
Parametry	Name [string]
Popis	Spustí danou službu

URI	/runscriptpost/stopProcess
Metoda	POST
Parametry	Id [string]
Popis	Zastaví zvolený proces

URI	/runscriptpost/stopService
Metoda	POST
Parametry	Name [string]
Popis	Zastaví zvolenou službu

Dalšími dostupnými voláními jsou:

URI	https://<adresa_serveru:port>/<aplikace>/api/token
Metoda	POST
Parametry	
Popis	Vrací autorizační token, pro zadané uživatelské jméno a heslo

URI	http://<adresa_serveru:port>/<aplikace>/api/account/getcert
Metoda	GET
Parametry	
Popis	Stáhne ze serveru certifikát

URI	http://<adresa_serveru:port>/<aplikace>/api/account/ping
Metoda	GET
Parametry	
Popis	Slouží k testování dostupnosti API, vrací odpověď „hello“

B Obsah CD

V této příloze je popsán obsah přiloženého CD.

V kořenové složce je uložen adresář *src*, který obsahuje zdrojové kódy klientské i serverové aplikace. Složka *release* v kořenovém adresáři obsahuje instalační soubor pro instalaci klientské aplikace *release/client/mowin.apk* a soubory pro nasazení serverové aplikace ve složce *release/server*. Také obsahuje návod jak aplikace zprovoznit. Složka *doc* pak obsahuje vygenerovanou dokumentaci android i server aplikace ve formě html stránek. Serverová aplikace obsahuje navíc dokumentaci ve formátu „Windows nápovědy“ v souboru *doc/server/windows/ WinRemoteAdministration.chm*. Složka *doc/dip* pak obsahuje zdrojové soubory diplomové práce. Vlastní práce je pak uložena přímo v kořenové složce.

C Instalační příručka

V této příloze je uveden návod na instalaci a nasazení klientské aplikace na mobilní zařízení a serverové aplikace na webový server.

C.1 Instalace mobilní aplikace

K instalaci je potřeba přesunout na mobilní zařízení *apk* instalační soubor a po té jej v mobilním zařízení otevřít. Mobilní zařízení je nutné připojit k PC. Je na uživateli zda pomocí USB kabelu nebo pomocí Wi-Fi, důležitý je pouze přesun instalačního souboru na zařízení. Stejně tak je na uvážení uživatele, do které složky instalační soubor nakopíruje. Uvedený návod používá pro připojení USB kabel a pro uložení složku *Downloads* na mobilním zařízení.

1. Připojení mobilního zařízení k PC.
2. Zkopírování instalačního souboru *src/client/mowin.apk* z příloženého CD do složky *Downloads* v mobilním zařízení.
3. Vyhledání instalačního souboru v mobilním zařízení a spuštění.
 - a. Je možné, že na mobilním zařízení bude blokováno instalací z neznámých zdrojů. Toto je potřeba povolit v nastavení telefonu. V závislosti na systému, obvykle *Nastavení > Zabezpečení > Neznámé zdroje* (po instalaci je doporučeno toto nastavení opět zablokovat). Po povolení je potřeba provést krok 3 znovu.
4. Po otevření instalace bude uživatel informován o službách systému, ke kterým si aplikace žádá přístup (úplný přístup k síti) - pokračujte tlačítkem instalace.

Po instalaci je možné aplikaci spustit, měla by naběhnout logovací obrazovka se zadáním přihlašovacích údajů. Pokud je již nainstalovaná serverová část aplikace, je možné se přihlásit s výchozím účtem *supervisor / 6o3hwr*.

C.2 Nasazení serverové aplikace

Předpokladem pro nasazení aplikace je již běžící Windows Server 2012R2 s aktivními službami IIS a Active Directory (pokud není Active Directory aktivní, nebudou v rámci aplikace dostupné služby pro správu uživatelů a stanic) a MSSQL Server.

Postup nasazení:

1. Připojení k serveru s právy Administrátora.
2. Vytvoření adresáře, ve kterém bude umístěna aplikace a odkud bude webový server aplikaci načítat (např. *C:\mowin*).
3. Zkopírování obsahu složky *src\server\mowin* z příloženého CD do vytvořené složky *C:\mowin* na serveru.
4. Přesun do adresáře *C:\mowin\App_Data* a vygenerování certifikátu spuštěním skriptu *C:\mowin\App_Data\createCert.ps1* s parametrem DNS jména nebo IP adresy, pod kterým je server přístupný (v příkladu je to adresa 192.168.0.106) a heslem k vnitřnímu úložišti v certifikátu (je potřeba např. v případě exportu privátního klíče). Vygenerovaný certifikát je pak uložen ve složce *App_Data* a automaticky zkopírován do lokálního úložiště certifikátů na serveru.
5. Nastavení webové aplikace. Je potřeba otevřít administraci IIS, pod webovým serverem otevřít složku *Sites* a v menu *Actions* vybrat položku *Add Website*. V dialogovém okně nastavit název nové aplikace (je jedno jaký, pro přehlednost zvoleno *mowin*) a fyzickou cestu k adresáři, který byl vytvořen v kroku 2. V části *Binding* nastavit IP adresu, na které bude aplikace přístupná (v příkladu 192.168.0.106). U *binding* je potřeba ponechat standartní nastavení portů 80 pro nezabezpečený a 443 pro zabezpečený přenos, na jiných portech nemůže pracovat mobilní aplikace. *Application pool* nechat nastaven na *mowin* aby se vytvořil nový. V tomto dialogu je to vše.
6. Následně je potřeba nastavit mapování pro zabezpečenou verzi. Z menu *Actions* je potřeba vybrat *Bindings* u aplikace *mowin*. V následujícím dialogu dát *Add*. Nyní v dialogu vybrat typ protokolu *https* opět zvolit IP adresu (192.168.0.106) a z menu vybrat SSL certifikát vytvořený v kroku 4.
7. V záložce *Application Pools* je potřeba vybrat *mowin*. V dialogu *Advanced Settings* pak vybrat pro vlastnost *Identity* hodnotu *LocalSystem*. To zaručí spouštění skriptů s právy lokálního systému.
8. Nyní by měla být aplikace již veřejně dostupná. Zbývá nastavit připojení k databázi. Je potřeba ze složky *App_Data* přesunout soubory *OwinAuthDbContext.mdf* a *OwinAuthDbContext_log.ldf* do datového adresáře MSSQL Serveru, na testovacím serveru to byla cesta:

c:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA

9. V souboru *C:\mowin\Web.config* je ještě potřeba nakonfigurovat připojení databáze. Je nutné upravit kód v elementu *connectionStrings* a u něj vlastnost *connectionString*:
- Data Source** – Název počítače\název instance MSSQL Serveru (pokud je na serveru nainstalována pouze jediná instance, měl by stačit název počítače).
 - User ID** – účet pro připojení k serveru (použit účet *sa* což je výchozí účet pro správu SQL Serveru, který se vytváří automaticky).
 - Password** – heslo k účtu.
 - AttachDBFilename** – cesta k *mdf* souboru s databází.

```
<connectionStrings>
  <add name="OwinAuthDbContext"
connectionString="Data Source=MYSERVER2012;User ID=sa;Password=Ad
min123;AttachDBFilename=c:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\OwinAuthDbContext.mdf;"
        providerName="System.Data.SqlClient"/>
</connectionStrings>
```

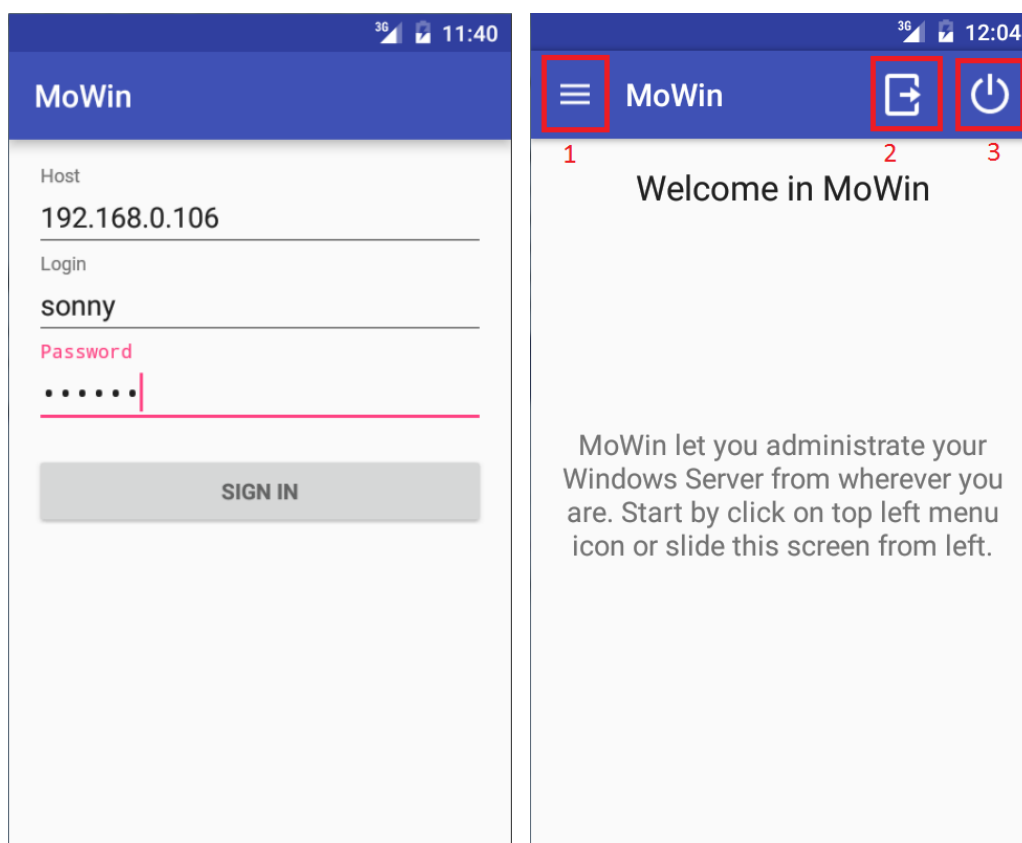
Nyní by aplikace měla být nakonfigurovaná a přístupná na adrese <https://192.168.0.106>. Při prvním požadavku, který bude nějakým způsobem pracovat s databází (např. Login) se databáze automaticky připojí. Výchozí účet pro přihlášení je *supervisor / 6o3hwr*.

D Uživatelská příručka

V této části bude uveden návod na používání a představeny možnosti mobilní aplikace a webového rozhraní.

D.1 Android aplikace

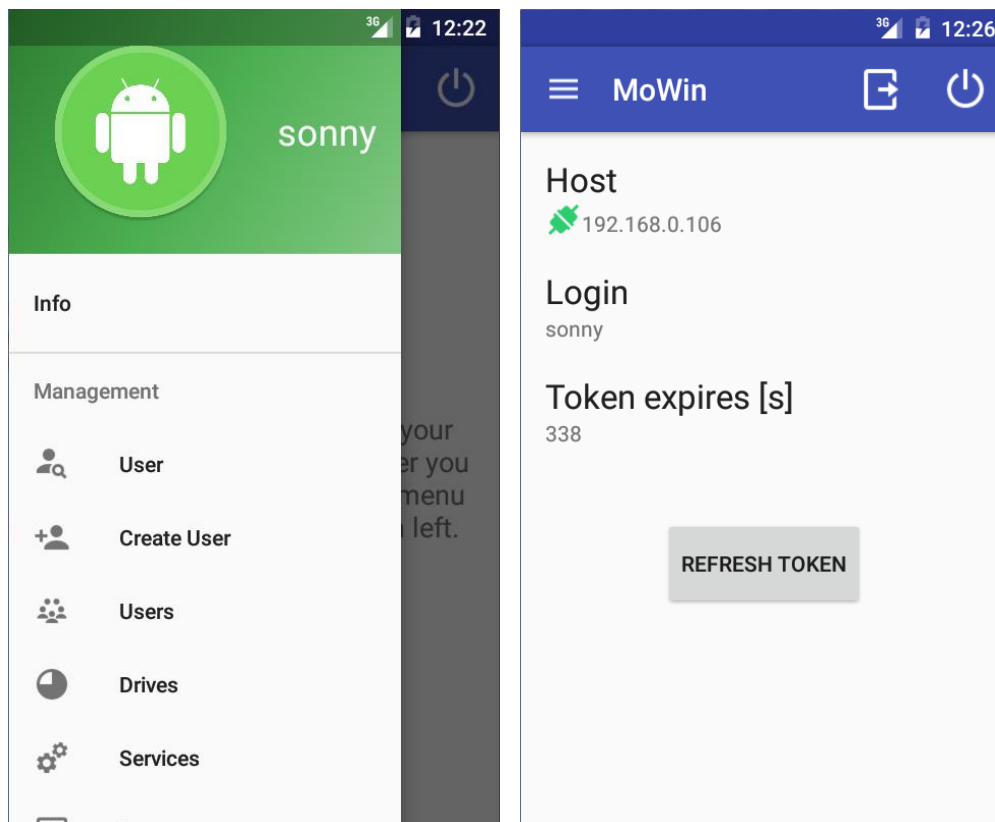
Po spuštění aplikace je otevřena přihlašovací aktivita pro zadání adresy serveru, ke kterému se má aplikace připojit. Zároveň uživatel zadává přihlašovací jméno a heslo, pro které má na daném serveru vytvořen účet (účet musí být přiřazen v roli *admin*). Kliknutím na tlačítko *Sign In* proběhne pokus o přihlášení. Pokud je úspěšné, je přihlašovací aktivita ukončena a otevře se hlavní aktivita, ve které je již možné provádět vlastní správu viz. Obrázek D.1.1.



Obrázek D.1.1 :Úvodní aktivita a hlavní aktivita po úspěšném přihlášení

V horní části aplikace je menu, které je zobrazeno napříč celou aplikací. Obsahuje název aplikace a několik ikon. Ikona [1] slouží k otevření ovládacího panelu, ikona [2]

k odhlášení z aktuálního serveru a ikona [3] ukončuje aplikaci viz. Obrázek D.1.1. Při odhlašování a ukončování je uživatel dotázán zda chce danou akci opravdu provést.

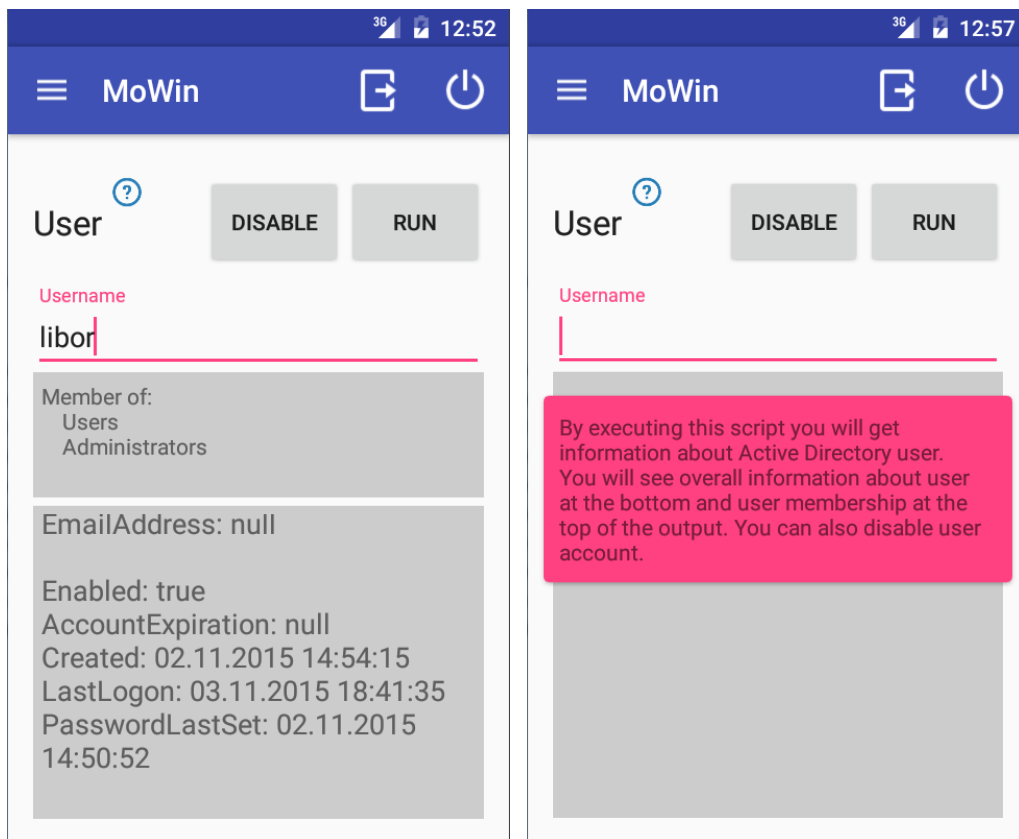


Obrázek D.1.2: Otevřený ovládací panel na snímku vlevo, na snímku vpravo informační fragment

Obrázek D.1.2 zobrazuje otevřený ovládací panel, kde je v horní části uživatelské jméno přihlášeného administrátora, pod ním položka *Info* a dále již jednotlivé klikatelné položky pro správu serveru. Ovládací panel je možné kdykoliv otevřít pomocí ikony z menu nebo přejetím po obrazovce zleva doprava. Stejně tak je možné přecházet mezi libovolnými činnostmi pomocí kliknutí na příslušnou činnost v panelu. Obrázek také zobrazuje otevřený informační fragment (položka *Info*) s informacemi o adrese serveru, ke kterému je aplikace připojena, uživatelské jméno přihlášeného administrátora a čas do vypršení autorizačního tokenu. Tlačítkem *Refresh Token* se otevře dialog pro zadání uživatelského hesla a přihlášení se obnoví. Doba platnosti tokenu záleží na nastavení v serverové části aplikace, standardně je nastavena na 30 minut.

Obrázek D.1.3 ukazuje fragment pro získání informací o uživateli. Ikona otazníku u názvu činnosti zobrazuje nápovědu co činnost provádí a je dostupná u všech činností. U této činnosti se po zadání loginu a kliknutí na tlačítko *Run* zobrazí informace o uživateli

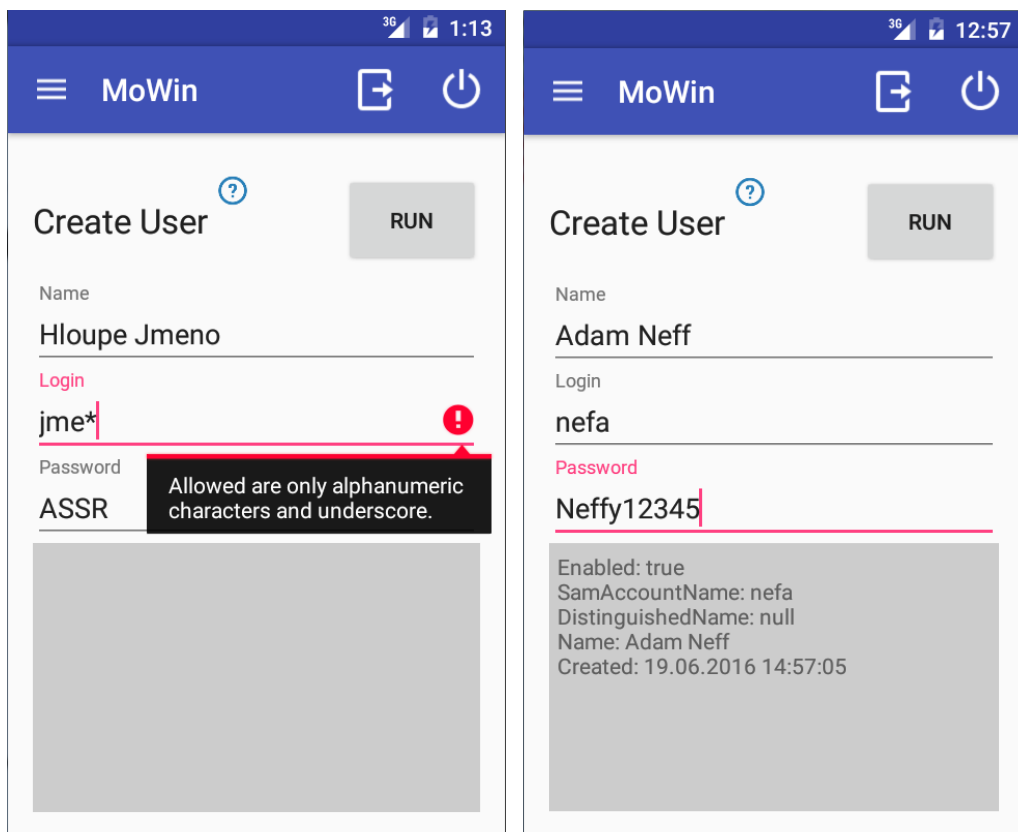
v dolním výstupním panelu a v panelu nad ním členství uživatele ve skupinách. Oba panely jsou scrovatelné.



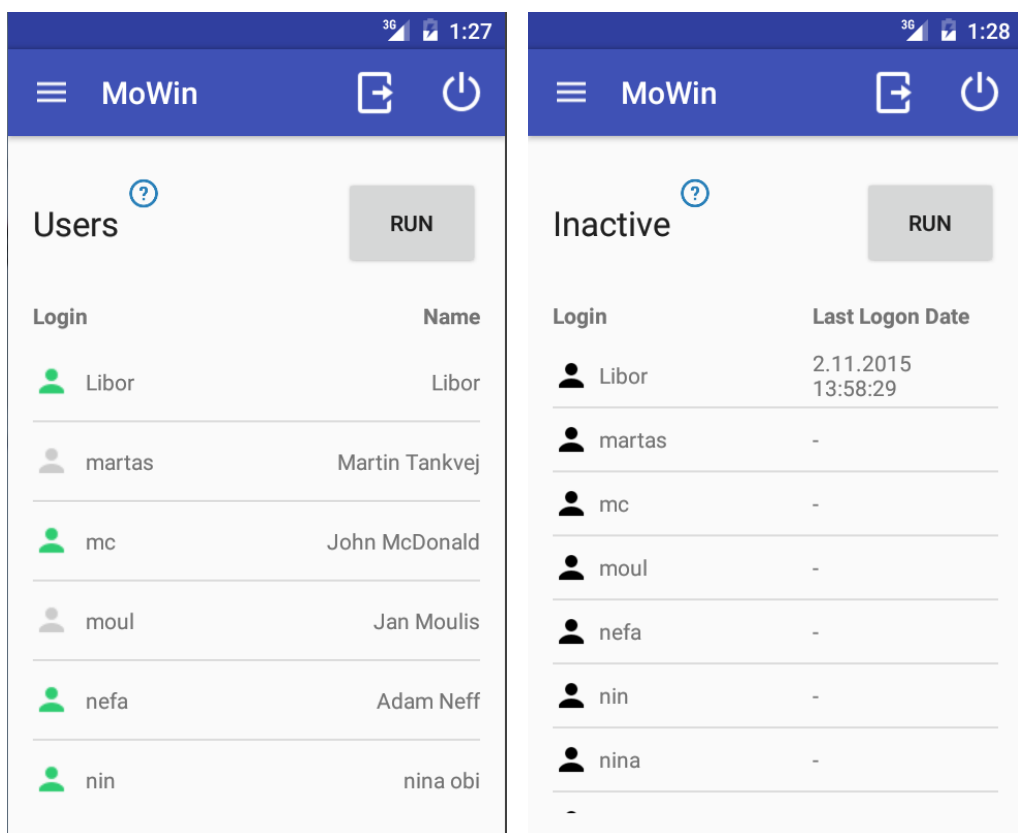
Obrázek D.1.3: Získání informací o uživateli a zobrazení nápovědy

Další činností je vytvoření nového uživatele. Obrázek D.1.4 ukazuje vytváření nového uživatele kde je zadán nevalidní vstup a ukázán dialog informující uživatele o tom co může vstup obsahovat za znaky. Na další obrazovce jsou již správně zadány všechny vstupy a zobrazen výstup ze serveru, který informuje o úspěšném vytvoření nového uživatele.

Obrázek D.1.5 zobrazuje obrazovky činnosti pro zobrazení všech uživatelů a činnosti pro zobrazení neaktivních uživatelů a stanic. U činnosti s uživateli je zobrazen status účtu kdy zelený panáček znamená aktivní účet a šedý neaktivní. Dále je zobrazen login a celé jméno. U neaktivních uživatelů je pak zobrazen typ, kdy panáček značí uživatele a počítač značí stanici. Dále je zobrazen název stanice nebo login uživatele a datum posledního přihlášení nebo pouze pomlčka (uživatele se nikdy nepřihlásil).

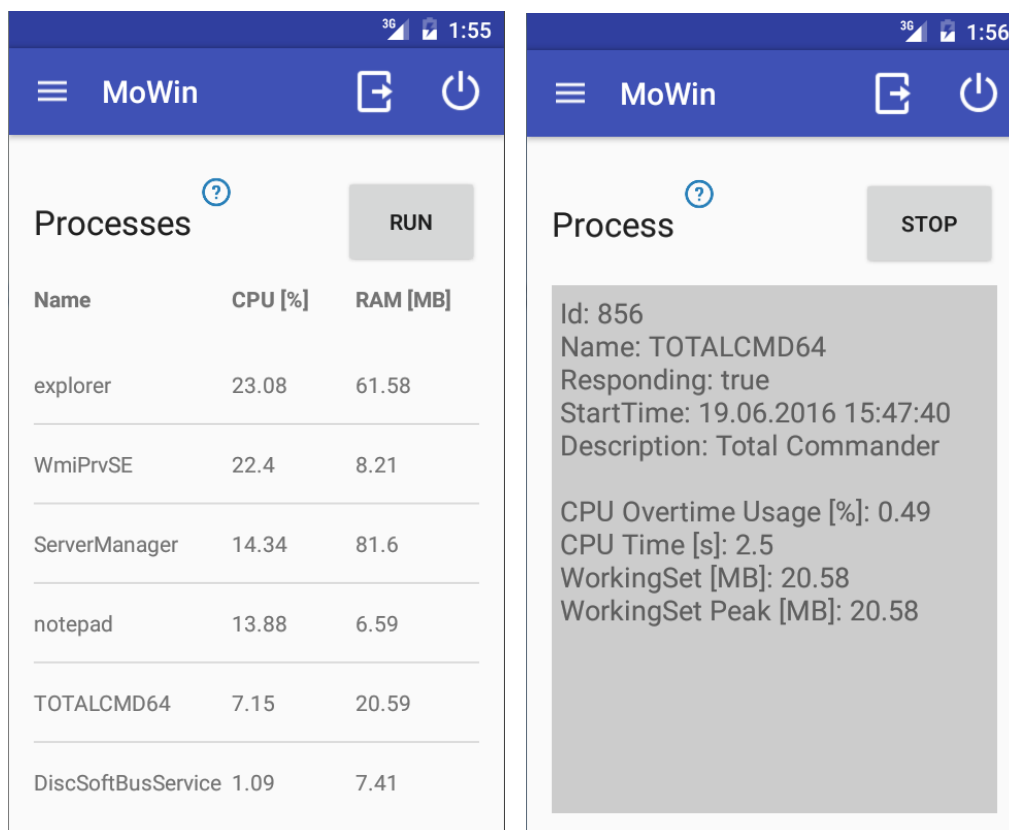


Obrázek D.1.4: Vytváření nového uživatele s ukázkou nevalidního vstupu



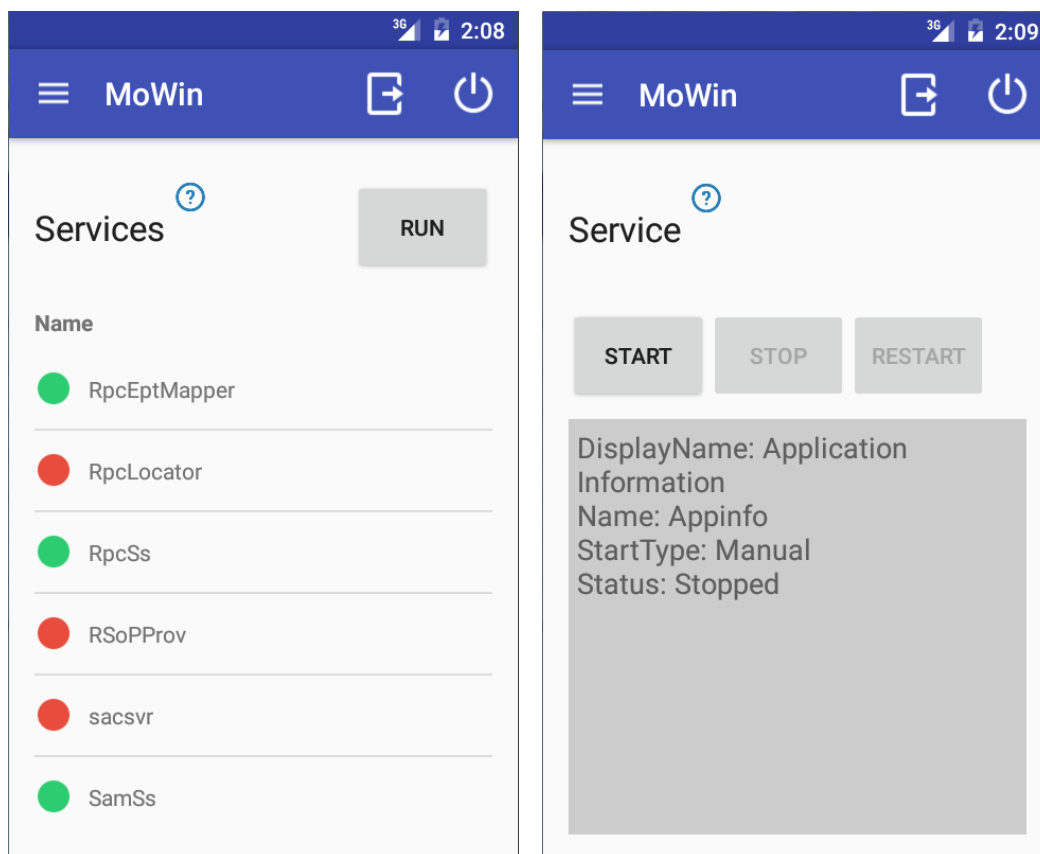
Obrázek D.1.5: Činnosti pro zobrazení aktivních a neaktivních uživatelů a činnost pro zobrazení účtů a stanic ke kterým se po dobu 30 dnů nikdo nepřihlásil

Další činností viz. Obrázek D.1.6. je zobrazení seznamu aktuálně běžících procesů s názvem procesu, spotřebou procesorového času od startu procesu a aktuální spotřebou paměti. Po kliknutí na položku seznamu je zobrazen fragment s dalšími informacemi o procesu. V této části je také možné příslušný proces zastavit pomocí tlačítka *Stop*.



Obrázek D.1.6: Zobrazení seznamu běžících procesů a podrobné informace o procesu s možností zastavení procesu

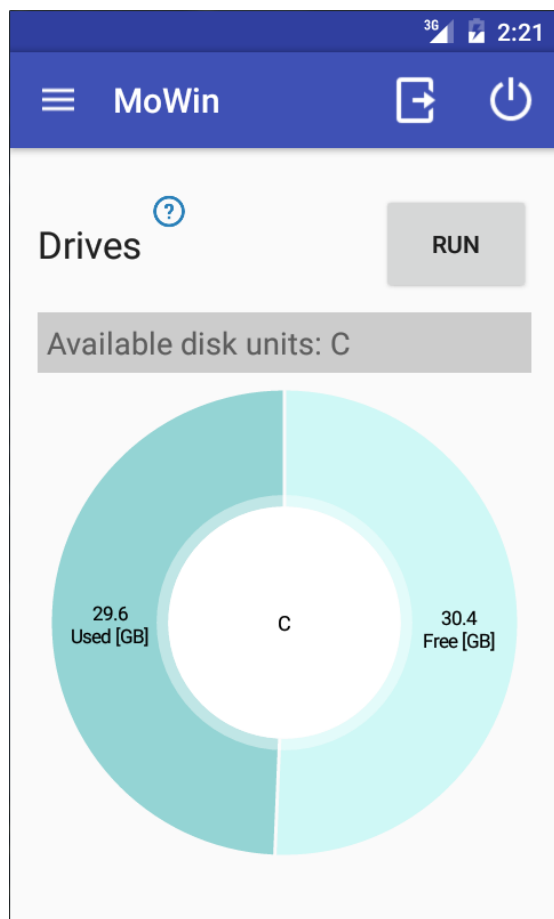
Obrázek D.1.7 zobrazuje fragment se seznamem dostupných služeb v systému. V seznamu je zobrazen název služby a indikace zda služba běží (zelené kolečko) nebo ne (červené kolečko). Po kliknutí na položku v seznamu se otevře fragment s bližšími informacemi o službě a uživatel má možnost službu zastavit, spustit nebo restartovat. Tlačítka *Start*, *Stop* a *Restart* se zpřístupňují podle aktuálního stavu služby.



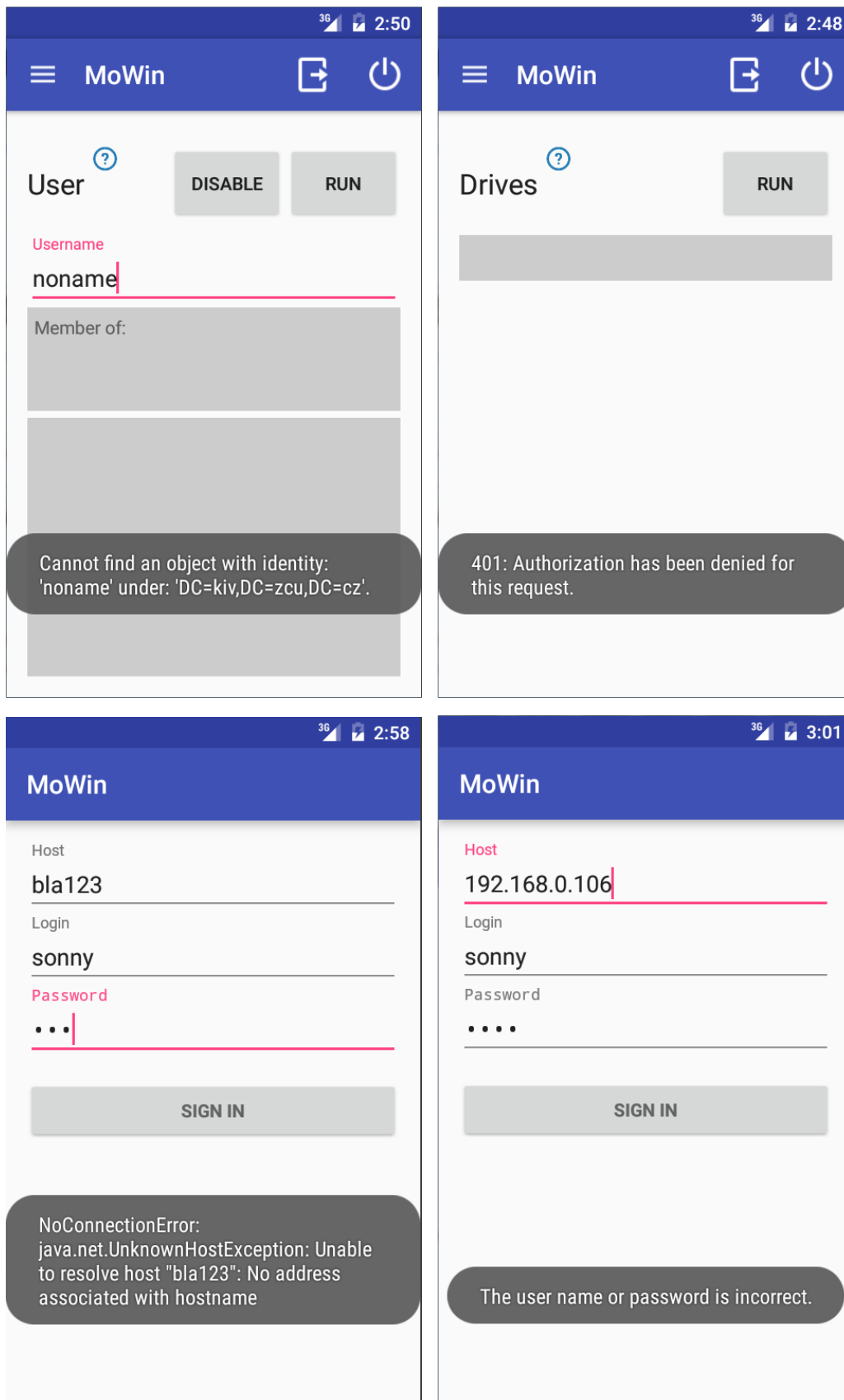
Obrázek D.1.7: Činnost zobrazující seznam dostupných služeb a další činnosti pro zobrazení detailů o službě s možností spuštění, zastavení a restartování

Poslední činnost správy, kterou je možné v aplikaci používat, je zobrazení kapacity úložišť. Fragment, který zobrazuje Obrázek D.1.8 ukazuje graf využití místa disku. Zkoumaný server má pouze jedno úložiště, pokud by bylo jednotek více, vykreslí se pro každé úložiště samostatný graf.

Obrázek D.1.9 pak ukazuje chybové výstupy, které se zobrazují jako vyskakovací zpráva tzv. *Toast*. Ten zobrazuje chyby, které se mohou vyskytnou přímo v aplikaci (např. neznámá adresa serveru) pak ty, které vrací webová aplikace (např. spuštění činnosti po vypršení autorizačního tokenu) a také chyby, které vrací PowerShell (např. nenalezení uživatele v Active Directory).



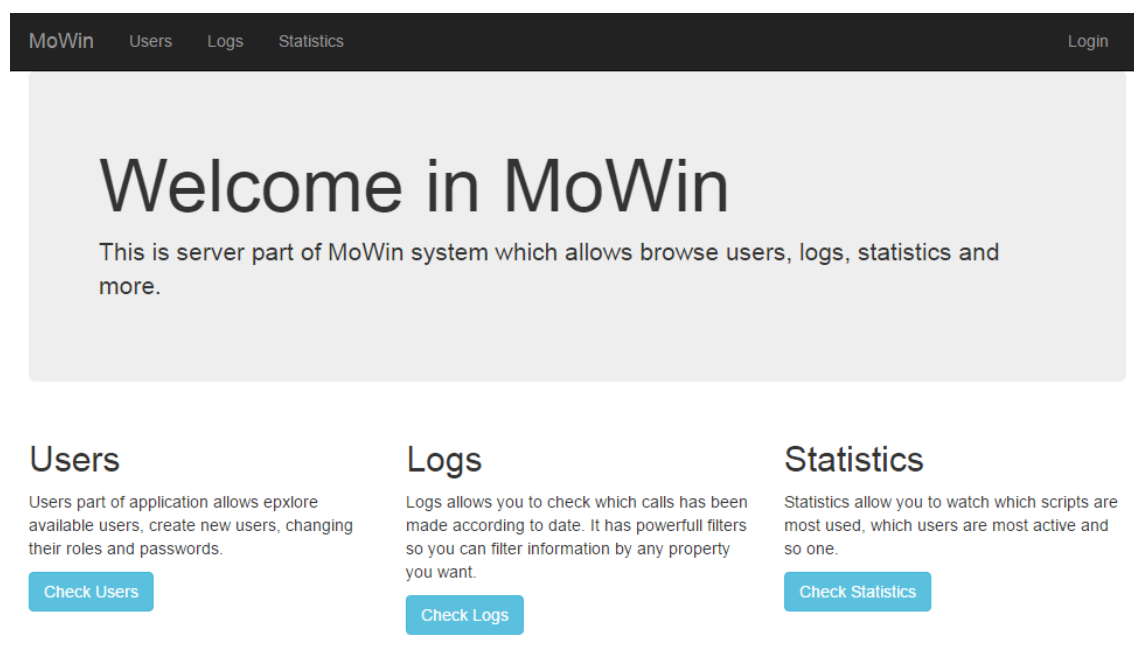
Obrázek D.1.8: Fragment zobrazující kapacitu úložiště



Obrázek D.1.9: Ukázky několika chybových stavů

D.2 Webová aplikace

Po otevření webové aplikace na příslušné adrese, se zobrazí úvodní stránka, která slouží jako rozcestník viz. Obrázek D.2.1. V horní části je umístěno menu, které je dostupné ze všech stránek aplikace. Obsahuje odkazy na jednotlivé části správy a odkaz na stránku s přihlášením (pro přihlášené tlačítko pro odhlášení). Při pokusu o vstup do jednotlivých částí je nepřihlášený uživatel přesměrován na stránku pro přihlášení. Po úspěšném přihlášení je přesměrován na úvodní stránku a již může přistupovat k dalším částem.



Obrázek D.2.1: Úvodní obrazovka webového rozhraní

D.2.1 Administrace uživatelů a správa rolí

Administrace uživatelů je zobrazena na obrázku D.2.2. V tabulce jsou zobrazeni všichni uživatelé systému, s jejich uživatelským jménem, emailem (pokud byl zadán při vytváření účtu) a s výpisem rolí, které daný účet má přiděleny. Kliknutím na uživatelské jméno se administrace přesune do části s výpisem informací o uživateli. Kliknutím na *Edit Roles* do části pro editaci rolí. V tabulce je možné řadit podle jakéhokoliv sloupce, stačí kliknout na hlavičku sloupce. Dále je možné filtrovat záznamy napsáním požadované hodnoty nebo její části do pole *Search* (filtruje se okamžitě). Pod tabulkou je umístěno tlačítko, které přesměruje na stránku s formulářem pro vytvoření nového uživatele.

MoWin Users Logs Statistics Logged as libor Logout

Users

Show entries Search:

Username	Email	Roles	
admin		admin	Edit Roles
boby	bob@adi.cz	admin	Edit Roles
franko	frankie@mowin.net	admin	Edit Roles
johny		admin	Edit Roles
Libor	libor@mowin.net	supervisor,admin	Edit Roles
sonny	sonnyman@vega.org	admin	Edit Roles

Showing 1 to 6 of 6 entries Previous Next

[Create User](#)

Obrázek D.2.2: Ukázka z administrace uživatelů

Po kliknutí na uživatele boby se otevře stránka s detailem uživatele. Na této stránce je možnost změnit heslo uživatele (pouze pokud není v roli supervizora – to neplatí pro aktuálně přihlášeného uživatele). Obrázek D.2.3 ukazuje stránku s detailem uživatele po změně hesla. Na stránce je zobrazena hláška o provedené operaci.

MoWin Users Logs Statistics Logged as libor Logout

User boby

Id: 057d9805-2c82-4bd3-b341-5d78e1325232

Email: bob@adi.cz

Roles: admin

Set new password

User password has been changed ×

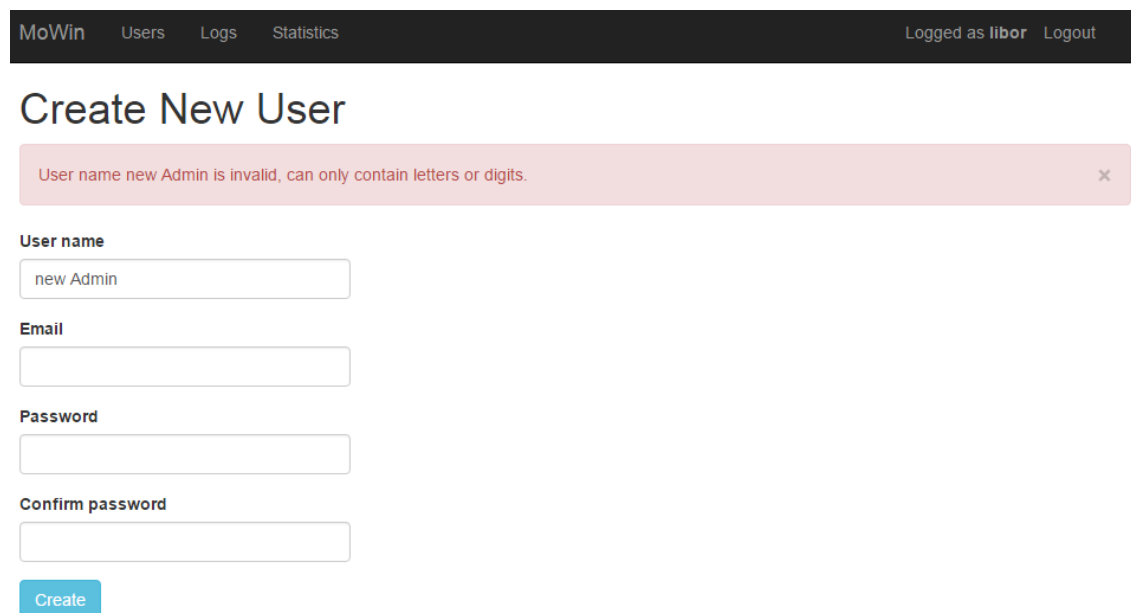
Password

Confirm password

[Reset password](#)

Obrázek D.2.3: Ukázka stránky s detailem uživatele po změně jeho hesla

Kliknutím na tlačítko *Create User* se otevře stránka s formulářem pro vytvoření nového uživatele. Do formuláře se zadává povinně uživatelské jméno a dvakrát heslo pro kontrolu. Volitelně pak email. Uživatelské jméno může obsahovat pouze alfanumerické znaky a heslo musí mít alespoň 6 znaků. Obrázek D.2.4 zobrazuje pro ukázkou chybu validace při vytváření nového uživatele.



MoWin Users Logs Statistics Logged as libor Logout

Create New User

User name new Admin is invalid, can only contain letters or digits. ✕

User name

Email

Password

Confirm password

Create

Obrázek D.2.4: Formulář pro vytvoření nového uživatele

Kliknutí na odkaz *Edit Roles* v seznamu uživatelů se administrace přesune na stránku se správou rolí viz. Obrázek D.2.5. Pod nadpisem stránky je v modrém boxu vidět ve kterých rolích je uživatel členem. Na stránce jsou dva formuláře, horní je pro přidávání uživatele do role. Pokud chceme například přidat uživatele *boby* do role *supervisor*, do políčka *UserName* vyplníme *boby*, ze seznamu *Role* vybereme *supervisor* a klikneme na tlačítko *Add to role*. Pro odebrání funguje stejný postup pouze v dolním formuláři. Uživatelské jméno. Políčko *UserName* se předvyplňuje dle jména, kterým přistupujeme ze seznamu uživatelů. Pokud je uživatel již v roli a má být přidán nebo naopak není a má být odebrán, tak je uživatel formou chybového výpisu informován o průběhu akce.

Manage User Roles

boby is member of: [admin](#)

Add User to Role

UserName

Role

Add to role

Delete A User from a Role

UserName

Role

Delete from role

Obrázek D.2.5: Stránka pro správu rolí

D.2.2 Logování

Tato část slouží k zobrazení provedených API volání. Uživatelské rozhraní obsahuje kalendář viz. Obrázek D.2.6, ve kterém se vybere datum, pro který chceme zobrazit jednotlivé logy.

Logs

June 2016						
Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Api calls from **19-06-2016**

Obrázek D.2.6: Kalendář pro výběr data logování

Po výběru je asynchronně obnoven seznam příslušných logů, který je zobrazen v tabulce viz. Obrázek D.2.7. Tyto záznamy je možné řadit podle jakéhokoliv sloupce a také filtrovat podle jakýchkoli dat, podobně jako je tomu v sekci pro administraci uživatelů.

Api calls from 19-06-2016

Show entries Search:

Time	User	Controller	Action	Script	Params	Response status code
13:42:07		account	getcrt			200
14:02:01		account	getcrt			200
14:52:01	sonny	pscripts	runscriptpost	getUser	User=libor	200
14:52:17	sonny	pscripts	runscriptpost	getUserMembership	Name=libor	200
14:57:04	sonny	pscripts	runscriptpost	createUser	Login=nefa&Password=Neffy12345&Name=Adam+Neff	200
14:57:05	sonny	pscripts	runscriptpost	getUser	User=nefa	200
15:26:08		pscripts	runscript	getUsers		401
15:26:08		pscripts	runscript	getUsers		401
15:26:29	sonny	pscripts	runscript	getUsers		200
15:28:03	sonny	pscripts	runscript	getInactive		200

Showing 1 to 10 of 42 entries Previous 2 3 4 5 Next

Obrázek D.2.7: Tabulka zobrazující Api volání ve vybraný den

Po kliknutí na některý ze záznamů v tabulce jsou pod tabulkou zobrazeny další detaily o příslušném volání, ze kterých je možné vyčíst informace jako který uživatel volání prováděl, návratový kód, odpověď serveru a další viz. Obrázek D.2.8.

Call details

Time : 14:52:01
Controller : pscripts
Action : runscriptpost
Script : getUser
Content Type : application/x-www-form-urlencoded; charset=UTF-8
Uri : https://192.168.0.106/api/pscripts/runscriptpost/getUser
Request Method : POST
Request Ip : 192.168.0.104
Params : User=libor
User : sonny

Response Time : 14:52:16
Response Status : 200
Response Content Type : application/json
Response Content Body : "

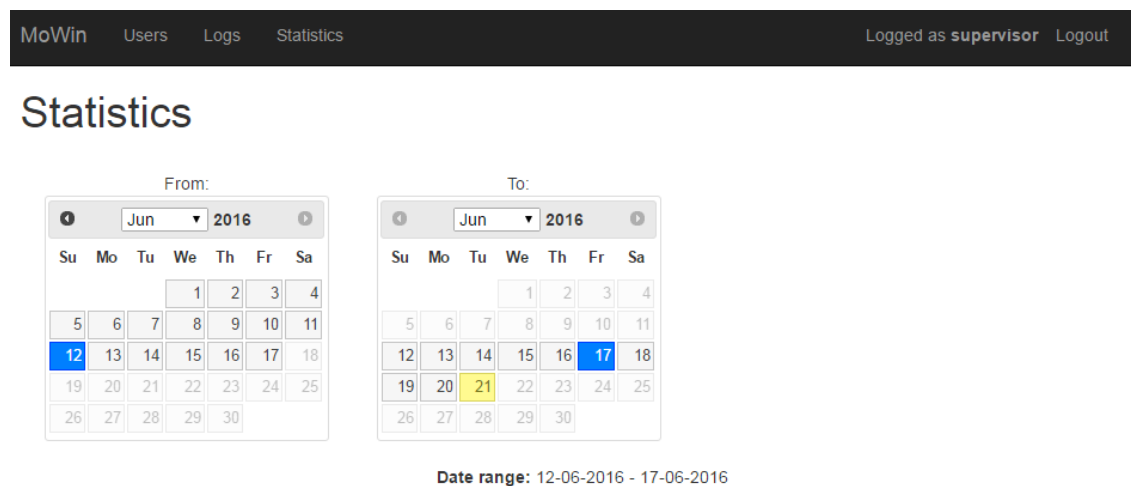
```
{
  "GivenName": null,
  "Surname": null,
  "Name": "Libor",
  "DisplayName": null,
  "SamAccountName": "Libor",
  "Enabled": true,
  "Created": "02.11.2015 14:54:15",
  "LastLogon": "03.11.2015 18:41:35",
  "AccountExpirationDate": null,
  "PasswordLastSet": "02.11.2015 14:50:52",
  "EmailAddress": null
}
```

"

Obrázek D.2.8: Detaily konkrétního Api volání

D.2.3 Statistika

Na základě provedených Api volání jsou v této sekci zobrazeny rozličné statistiky ve formě grafů. Uživatelské rozhraní obsahuje dva kalendáře viz. Obrázek D.2.9, které slouží pro označení časového rozsahu logů, pro který se mají statistiky zobrazovat.



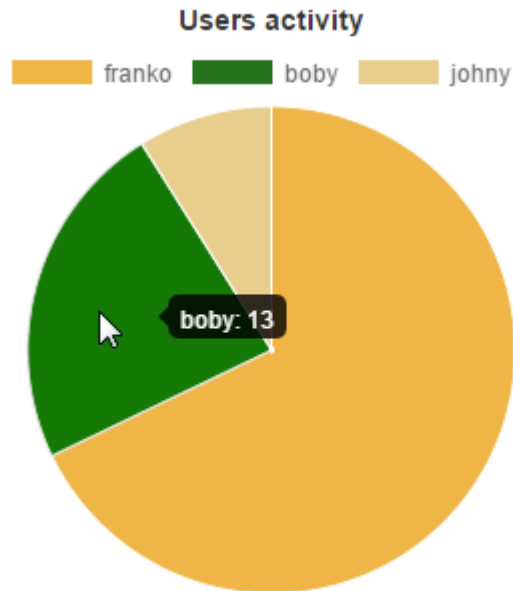
Obrázek D.2.9: Výběr rozsahu logů

Po změně data jsou změny ihned promítnuty do zobrazení příslušných statistik. Najetím myši na sloupec nebo část koláčového grafu se zobrazí hodnota dané statistiky. Obrázek D.2.10 zobrazuje statistiky využívání jednotlivých činností správy ve formě sloupcového grafu, Obrázek D.2.11 informace o aktivitě uživatelů (počet volání, které za dané období provedly), Obrázek D.2.12 počty statusů odpovědi od serveru,

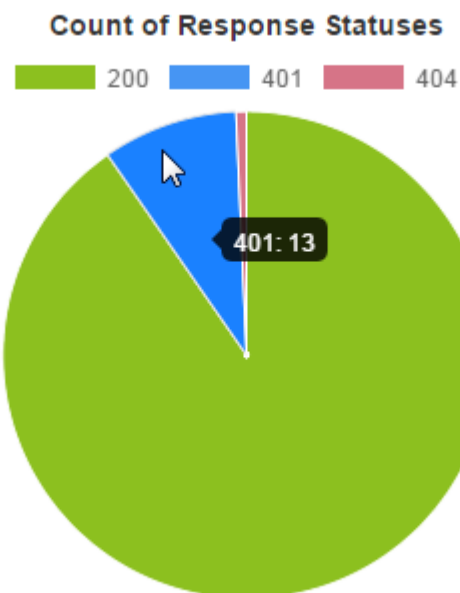


Obrázek D.2.10: Využívání jednotlivých činností správy

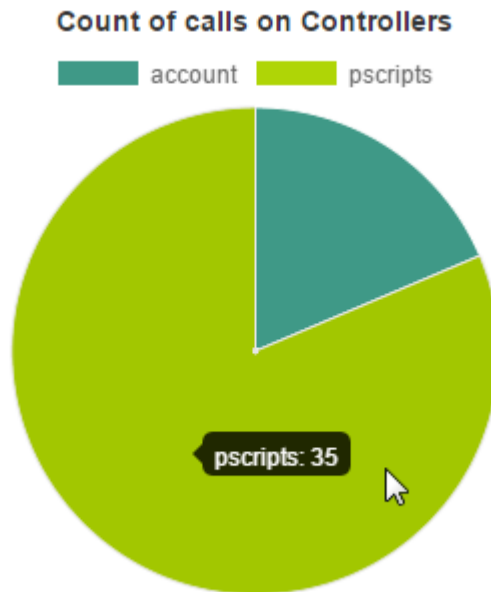
Obrázek D.2.13 využívání jednotlivých Api kontrolerů a Obrázek D.2.14 využití jednotlivých akcí, které kontrolery poskytují.



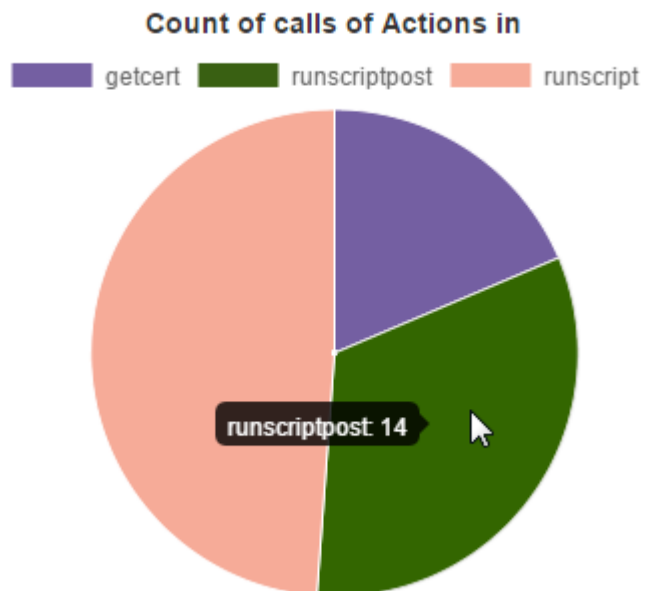
Obrázek D.2.11: Počet volání podle uživatelů



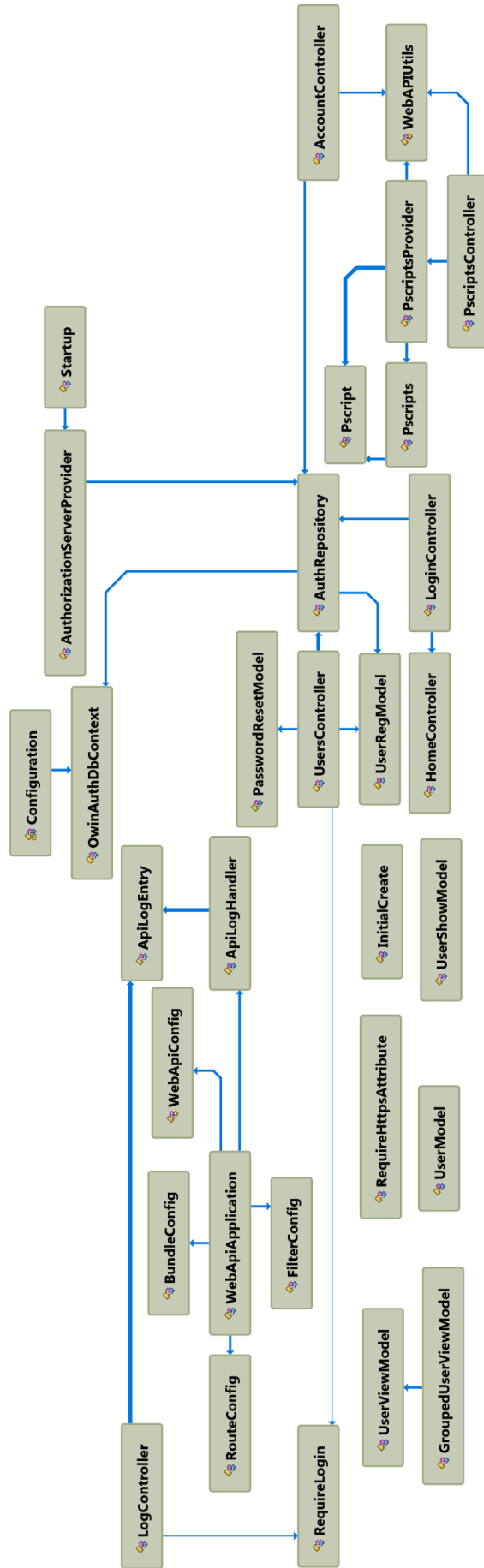
Obrázek D.2.12: Počet odpovědí od serveru (200 - OK, 401 - Unauthorized, 404 - Not Found)



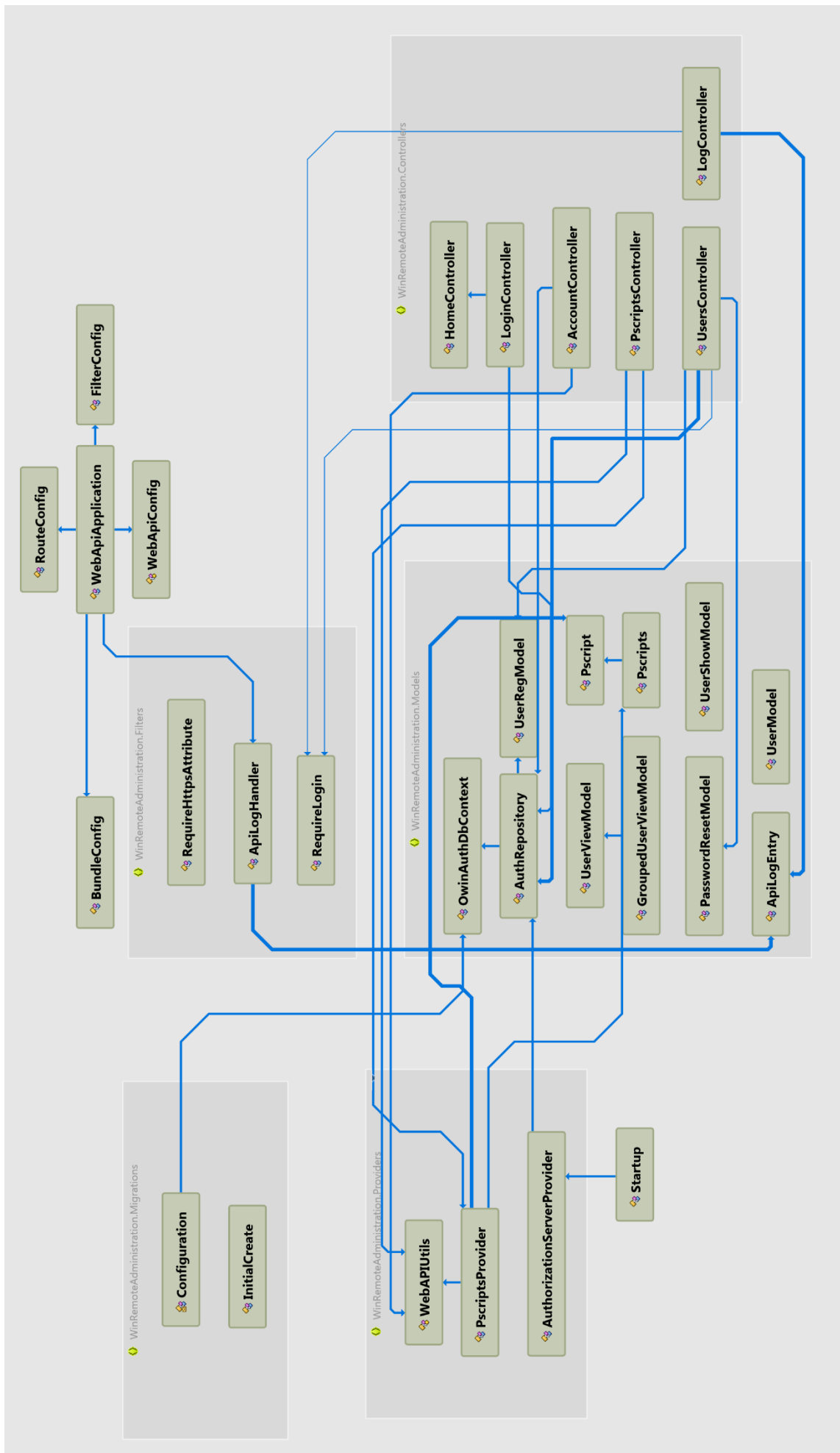
Obrázek D.2.13: Počet volání na jednotlivé kontrolery



Obrázek D.2.14: Využití akcí kontrolerů



Obrázek.E.1: Diagram závislosti tříd



Obrázek E.2: Diagram závislosti tříd a namespaces