

Agent pro síťový monitorovací systém serverových řešení

Bc. Ondřej Zálešák

Diplomová práce
2017



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Ondřej Zálešák**
Osobní číslo: **A15224**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **kombinovaná**

Téma práce: **Agent pro síťový monitorovací systém serverových řešení**
Téma anglicky: **An Agent for a Server-Solution Network Monitoring System**

Zásady pro vypracování:

1. Seznamte se s problematikou monitorovacích systémů.
2. Vypracujte rešerši existujících řešení.
3. Proveďte analýzu požadavků a uživatelských cílů na zvolené řešení.
4. Navrhněte vhodné řešení aplikace.
5. Realizujte navrženou aplikaci.
6. Popište nasazení do produkčního prostředí.
7. Proveďte vyhodnocení přínosu nového řešení.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně / Matthew MacDonald, Adam Freeman, Mario Szpuszta; překlad Jan Pokorný**
2. **KRAVAL, Ilja. Postupky pro EFEM s použitím UML se zaměřením na CASE nástroj ENTERPRISE ARCHITECT 3.6: USE CASE MODELLING. Valašské Klobouky: Objects Consulting, 2003, 46 s.**
3. **Mistrovství v SQL Server 2012: kompletní průvodce databázového experta / L'uboslav Lacko; překlad Martin Herodek**
4. **C# a .NET 2.0 profesionálně / Andrew Troelsen; překlad Jan Pokorný**
5. **Příklady modelů analýzy a návrhu aplikace v UML / Alena Buchalceová, Ivan Stanovská**
6. **Microsoft Visual C# 2008: krok za krokem / John Sharp; překlad Lukáš Krejčí, Jaroslav Černý**

Vedoucí diplomové práce:

Ing. Petr Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

3. února 2017

Termín odevzdání diplomové práce:

16. května 2017

Ve Zlíně dne 3. února 2017



doc. Mgr. Milan Adámek, Ph.D.
děkan



prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 16.5.2017



podpis diplomanta

*** naskenované Prohlášení str. 2***

ABSTRAKT

Diplomová práce se zabývá agentem pro monitorovací systém serverových řešení. Jedná se nástroj pro dohled a správu existujících instalací enterprise systému v architektuře klient-server. V práci je zastoupené řešení pro klientskou část, které slouží k vykonávání testů na vzdálených serverech a odeslání výsledků na centrální serverovou část. Agent je navržen a implementován v prostředí .net Framework. Práce také obsahuje výzkum, který zastřešuje porovnání stávajících řešení a popis jejich silných a slabých stránek.

Klíčová slova: .NET Framework, klient, server, signalR, SQLite, C#, služba, zásuvný modul

ABSTRACT

This diploma thesis deals with an agent for the monitoring system of server solutions. It is a tool for supervising and managing existing enterprise system installations in the client-server architecture. The work is represented by a client part solution, which is used to perform tests on remote servers and send the results to the central server part. The agent is designed and implemented in the .NET Framework. The work also includes research that covers the comparison of existing solutions and a description of their strengths and weaknesses.

Keywords: .NET Framework, client, server, signalR, SQLite, C#, service, plugin

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu mé diplomové práce Ing. Petru Šilhavému, Ph.D. za odborné vedení, konstruktivní připomínky a návrhy na zlepšení práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD.....	11
I.TEORETICKÁ ČÁST	13
1 MICROSOFT VISUAL STUDIO.....	14
1.1 ARCHITEKTURA.....	15
1.2 VLASTNOSTI	16
1.2.1 Editor kódu.....	16
1.2.2 Debugger	17
1.2.3 Designer	18
1.3 PODPOROVANÉ PRODUKTY	19
1.3.1 Microsoft Visual C++	19
1.3.2 Microsoft Visual C#.....	20
1.3.3 Microsoft Visual Basic.....	20
1.3.4 Microsoft Visual Web Developer	20
1.3.5 Team Foundation Server	20
1.4 EDICE	21
1.4.1 Community.....	21
1.4.2 Professional	21
1.4.3 Enterprise	21
1.4.4 Test Professional	21
1.4.5 Express	22
1.5 HISTORIE.....	22
1.5.1 97.....	22
1.5.2 6.0 (1998)	23
1.5.3 .NET (2002)	23
1.5.4 .NET (2003)	24
1.5.5 2005.....	25
1.5.6 2008.....	26
1.5.7 2010.....	27
1.5.8 Ultimate 2010.....	27
1.5.9 2012.....	28
1.5.10 2013.....	28
1.5.11 2015.....	28
1.5.12 2017.....	29
2 JAVASCRIPT OBJECT NOTATION (JSON).....	30
2.1 HISTORIE.....	30
2.2 DATOVÉ TYPY	30
2.2.1 Číslo	30
2.2.2 Řetězec	31
2.2.3 Boolean	31
2.2.4 Pole.....	31
2.2.5 Objekt.....	31
2.2.6 Null.....	31

2.3	PŘÍKLAD.....	31
2.4	POUŽITÍ JSON V JAVASCRIPTU	32
2.5	NEPODPOROVANÉ NATIVNÍ DATOVÉ TYPY	33
2.6	SCHÉMA JSON.....	33
2.7	BEZPEČNOSTNÍ RIZIKA.....	33
2.8	SROVNÁNÍ S JINÝMI FORMÁTY	34
2.8.1	YAML	34
2.8.2	XML	34
3	SQLITE	35
3.1	DESIGN	35
3.2	PŘÍKLAD APLIKACE KÓDU NA ZÁKLADĚ SQLITE.....	36
3.3	HISTORIE SQLITE	36
3.4	FUNKCE.....	37
4	SIGNALR.....	38
4.1	POPIS	38
4.2	PŘENOS DAT A ZÁLOHA	38
4.2.1	HTML 5 druhy přenosu	38
4.2.2	Comet přenos dat.....	39
4.2.3	Metoda výběru přenosu.....	39
5	INTERNET OF THINGS (IOT)	41
6	ENTITY FRAMEWORK.....	42
6.1	PŘEHLED	42
7	REŠERŠE EXISTUJÍCÍCH ŘEŠENÍ.....	43
7.1	SYSTEM CENTER OPERATIONS MANAGER	43
7.1.1	Výhody.....	44
7.1.2	Nevýhody	45
7.1.3	Cena.....	45
7.2	SOLARWINDS NPM	45
7.2.1	Výhody.....	46
7.2.2	Nevýhody	46
7.2.3	Cena.....	46
7.3	APPDYNAMICS	46
7.3.1	Výhody.....	46
7.3.2	Nevýhody	46
7.3.3	Cena.....	47
7.4	DYNATRACE APM.....	47
7.4.1	Výhody.....	47
7.4.2	Nevýhody	47
7.4.3	Cena.....	47

7.5	SCIENCELOGIC	48
7.5.1	Výhody	48
7.5.2	Nevýhody	48
7.5.3	Cena.....	48
7.6	VYHODNOCENÍ PRŮZKUMU	49
II.PRAKTICKÁ ČÁST		50
8	MOTIVACE PRO VZDÁLENÉ MONITOROVÁNÍ A SPRÁVU SKUPINY ZAŘÍZENÍ	51
9	ANALÝZA POŽADAVKŮ NA MONITOROVACÍ SYSTÉM.....	54
9.1	FUNKČNÍ POŽADAVKY	54
9.2	NEFUNKČNÍ POŽADAVKY	56
9.2.1	Seznam nefunkčních požadavků	57
9.3	CENA	58
10	APLIKACE MONITORING AGENT	59
10.1	BĚH PROGRAMU NA POZADÍ.....	62
10.2	PLUGINY (ZÁSUVNÉ MODULY)	62
10.2.1	Šablona pluginu.....	63
10.2.2	Seznam pluginů	64
10.3	AGENT SERVICE.....	71
10.4	DATABÁZE SQLITE.....	72
10.5	DIAGRAMY TŘÍD.....	74
10.6	MAPA KÓDU	76
10.7	LOGOVÁNÍ.....	80
10.8	NASAZENÍ DO REÁLNÉHO PROVOZU	83
10.9	OHODNOCENÍ NÁKLADŮ IMPLEMENTACE	84
10.10	PLÁNOVANÝ ROZVOJ.....	85
10.10.1	Možnost vzdáleně ovládat služby běžící na klientském stroji	85
10.10.2	Možnost stahovat nové verze pluginů a aplikace přes FTP	86
10.10.3	Připojení systému pro evidenci hlášení	87
10.10.4	Integrace e-mailového klienta	88
10.10.5	Rozšíření možnost návratových hodnot pluginů	89
10.10.6	Příprava aplikace pro multiplatformní prostředí	89
11	ZÁVĚR.....	91

ÚVOD

Rychlý a dynamický rozvoj technologií, informačních a komunikačních technologií zvláště, nám dovoluje využívat stále většího množství prostředků, výpočetního výkonu a možností zpracování větších objemů dat, než bylo před několika desítkami let vůbec myslitelné. Procesory, operační paměti, úložiště i samotná infrastruktura procházejí nezastavitelným procesem minimalizace, zrychlování a následně i procesem zlevňování jak z pohledu pořizovacích nákladů, tak i nákladů provozních. Technologie, jejichž použití bylo ještě před pár lety striktně vyhrazeno pro výzkumná střediska a univerzity disponující obrovskými rozpočty se dnes relativně nenápadně, ale o to masivněji dostávají do běžného života téměř každého člověka.

Velmi často zmiňovaným termínem je dnes pojem „Internet věcí“, znamená to mimo jiné i to, že velké množství zařízení je schopno komunikace a vzájemné interakce s jinými zařízeními v rámci sítě internet. Je možné ovládat celou svou domácnost za použití chytrého telefonu připojeného k internetu a online sledovat aktuální údaje široké škály domácích spotřebičů, nebo technického zabezpečení budovy od pračky přes tepelné čerpadlo až po zabezpečovací systém.

S narůstajícími možnostmi a výkonem jednotlivých prvků a s tím zákonitě spjatým nárůstem objemu dat každého systému narůstají také požadavky na stabilitu, spolehlivost a dostupnost všech prvků daného systému. Existují autonomní systémy, které mají možnost kontroly svého běhu a bez použití externích aplikací zasílají reporty do nadřazeného systému, který je vyhodnocuje a na základě předvolených parametrů může dojít k informování osoby zodpovědné za běh daného podsystemu. U tohoto typu systémů je třeba s použitím vlastního monitoringu počítat už při jejich návrhu a samotný monitorovací systém může ve výsledku dosahovat stejné, nebo vyšší komplexnosti, než samotný monitorovaný systém.

Pro prostředí a na něm běžících systémech, které neobsahují integrovanou možnost monitoringu a reportování běhu a chyb je třeba využít software třetích stran, který tuto práci provede. Existuje velké množství řešení, které mají různé parametry, kvalitu i cenu a není snadné se zorientovat v tom, které konkrétní řešení je pro danou implementaci to nejvhodnější.

Tato práce se zabývá myšlenkou a samotným řešením části monitorovacího systému, kterým je monitorovací agent, jenž běží na klientském prostředí a sbírá data, která následně zasílá serveru. Než došlo k samotnému návrhu, proběhla rešerše existujících řešení a prověření

jejich silných a slabých stránek a samozřejmě také zhodnocení cenových a licenčních podmínek. Ve výsledku bylo rozhodnuto vyvinout vlastní produkt, který bude plně transparentní a jeho funkcionalita bude snadno rozšiřitelná za pomoci zásuvných modulů, neboli tzv. pluginů.

I. TEORETICKÁ ČÁST

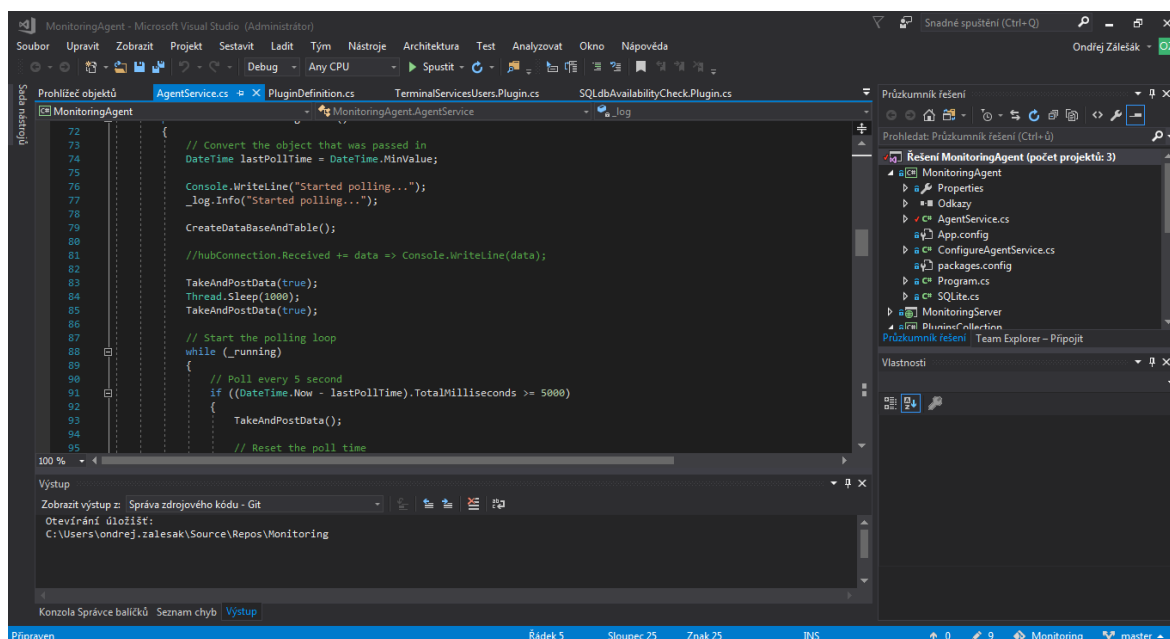
1 MICROSOFT VISUAL STUDIO

Microsoft Visual Studio (Obr. 1) je integrované vývojové prostředí (IDE) od firmy Microsoft. Je využíváno k vývoji počítačových programů převážně pro platformu Microsoft Windows, stejně tak pro vývoj webových aplikací, webových služeb a mobilních aplikací. Visual Studio využívá vývojové platformy firmy Microsoft jako například Windows API, Windows Forms, Windows Presentation Foundation, Windows Store a Microsoft Silverlight. Je schopno produkovat jak nativní, tak managed kód [16].

Visual Studio obsahuje editor kódu, který podporuje technologii IntelliSense stejně tak jako možnost refaktoringu kódu. Integrovaný debugger pracuje na úrovni zdrojového kódu i na úrovni stroje. Ostatní zabudované prvky obsahují profiler kódu, designer formulářů a designér databázového schématu. Akceptuje pluginy, které mohou rozšířit funkcionalitu téměř na každé úrovni a přidat novou sadu nástrojů jako například editory a vizuální designéry pro doménově specifické jazyky, nebo sady nástrojů pro ostatní aspekty životního cyklu vývoje software [16].

Visual Studio podporuje různé programovací jazyky a umožňuje editoru kódu a debuggeru podporovat téměř jakýkoliv programovací jazyk. Vestavěné jazyky obsahují C, C++ a C++/CLI (skrže Visual C++), VB.NET, C#, F# a TypeScript. Obsahuje dále podporu jazyků, jako jsou například Python, Ruby, Node.js a M, které je možné doinstalovat zvlášť. Dále podporuje XML/XSLT, HTML/XHTML, JavaScript a CSS. Java a J# byly podporovány v minulosti [16].

Microsoft poskytuje bezplatnou verzi Visual Studia, tzv. Community edici, která podporuje pluginy a je k dispozici bez jakýchkoliv poplatků [16].



Obr. 1. Prostředí Visual Studio 2017

1.1 Architektura

Visual Studio nepodporuje standardně všechny programovací jazyky [17], řešení, nebo nástroje. Namísto toho dovoluje přidávat funkcionalitu kódovanou jako VSPackage. Pokud je tento balík instalován, funkcionalita je dostupná jako služba. IDE poskytuje tři druhy služeb. SVsSolution, které poskytuje schopnost vyjmenovat projekty a řešení, SVsUIShell, které poskytuje práci s okny a funkcionalitu uživatelského rozhraní (UI) a SVSShell, které pracuje s registrací VSPackage. Navíc IDE je rovněž zodpovědné za koordinaci a zprostředkování komunikace mezi službami. Všechny editory, designéři, typy projektů a další nástroje jsou implementovány jako VSPackage. Visual Studio využívá COM pro přístup k VSPackage. Visual Studio SDK rovněž obsahuje tzv. „Managed Package Framework“ (MPF), který je sadou řízených wrapperů okolo COM rozhraní, které dovolují balíčkům, aby mohly být napsány v libovolném jazyku, který je v souladu s CLI. Služby mohou být využity pro tvorbu jiných balíčků, které přidají funkcionalitu do IDE Visual Studia [16].

Podpora programovacích jazyků je přidána za pomoci využití specifického VSPackage, který se jmenuje „Language Service“ [16]. Language service definuje rozličné rozhraní, které může implementace VSPackage implementovat a podpořit různé funkcionality. Funkcionalita, které mohou být tímto způsobem přidány, obsahují funkce: syntax coloring, statement completion, brace matching, parameter information tooltips, member lists a error

markers po kompilaci na pozadí. Pokud je interface implementován, funkcionality bude dostupná pro daný jazyk. Language services jsou implementovány na úrovni jazyků. Implementace mohou znovu využít kód z parseru, nebo compileru pro svůj jazyk. Language services mohou být implementovány buď v nativním, nebo managed kódu. Pro nativní kód mohou být použity i nativní COM rozhraní Babel Framework (součást Visual Studio SDK). Pro managed kód, MPF obsahuje wrappery pro psaní managed jazykových služeb. [17]

Visual Studio neobsahuje žádnou zabudovanou podporu kontroly zdrojového kódu, ale definuje dva alternativní způsoby, jak integrovat kontrolní systémy s IDE [16]. VSPackage pro kontrolu zdrojového kódu může poskytnout jeho vlastní upravené rozhraní. Oproti tomu, plugin pro kontrolu zdrojového kódu, který používá technologii MSSCCI (Microsoft Source Code Control Interface) poskytuje sadu funkcí, které jsou použity pro implementaci rozličné funkcionality pro kontrolu zdrojového kódu ve standardním rozhraní Visual Studia. MSSCCI bylo nejprve využito pro integraci Visual SourceSafe s Visual Studií 6.0 ale později bylo otevřeno skrze Visual Studio SDK. Visual Studio .NET 2002 využívalo MSSCCI 1.1 a Visual Studio .NET 2003 využívalo MSSCCI 1.2. Visual Studio 2005, 2008 a 2010 používalo MSSCCI verze 1.3, které přidalo podporu pro přejmenování a odstranění propagace a také asynchronní otevírání [17].

Visual Studio podporuje běh více instancí prostředí (každé z nich s vlastní sadou VSPackage) [18]. Instance využívají různé registrované úly, aby uložily svou konfiguraci a oddělily ji pomocí AppID (Application ID). Instance jsou spuštěny AppID specifickým .exe, které určuje AppId, nastavuje hlavní úl a spouští IDE. VSPackage registrované pro jedno AppId jsou integrovány s ostatními VSPackagemi. Rozdílné produktové edice Visual Studia jsou tvořeny za pomoci různých AppId. Visual Studio edice Express jsou instalovány s jejich vlastním AppId, ale edice Standard, Professional a Team Suite sdílí stejné AppId. V návaznosti na to může být instalována Express edice bok po boku ostatním edicím, toto však neplatí pro ostatní edice, které by provedly update dané instalace. Systém AppId je nahrazen komponentou Visual Studio Shell ve Visual Studiu 2008 [16].

1.2 Vlastnosti

1.2.1 Editor kódu

Stejně tak jako i jiné IDE, Visual Studio obsahuje editor kódu, který podporuje zvýrazňování syntaxe a doplňování kódu za pomoci systému IntelliSense pro proměnné, funkce, metody,

smyčky a LINQ dotazy [26]. IntelliSense je podporováno pro vestavěné jazyky, stejně tak pro XML, CSS a JavaScript. Samo doplňovací návrhy se objevují ve formě nabídky s textem, který se objevuje nad oknem editoru kódu v blízkosti kurzoru. Ve Visual Studio 2008 a novějších může být dočasně částečně průhledné, aby bylo možné vidět kód, který zakrývá. Editor kódu je použit pro všechny podporované jazyky [16].

Editor kódu Visual Studia také podporuje nastavení záložek v kódu pro rychlejší navigaci. Jiné navigační pomůcky obsahují skrývání bloků kódu a inkrementální vyhledávání vedle standardního textového hledání a regex hledání. Editor kódu také obsahuje schránku na více položek a seznam tasků [17]. Editor kódu podporuje útržky kódu, které jsou uloženými šablonami pro daný kód a mohou být vloženy do kódu a upraveny pro daný projekt, na kterém je právě pracováno. Nástroj pro správu útržků kódu je taktéž vestavěn. Tyto nástroje jsou zobrazeny jako plovoucí okna, které se mohou automaticky skrývat, nebo dokovat na straně obrazovky. Editor kódu ve Visual Studiu mimo jiné podporuje refaktoring kódu, který obsahuje parametr pro přezazení, přejmenování proměnných a metod, rozšíření rozhraní, nebo členů tříd [16].

Visual Studio obsahuje kompilaci na pozadí (také nazvanou jako „inkrementální kompilace“). V průběhu psaní kódu, Visual Studio kompiluje kód na pozadí tak, aby mohl poskytnout zpětnou vazbu ohledně chyb v syntaxi a kompilačních chyb, které jsou následně označeny červeným vlnkovitým podtržením. Varování jsou označena zeleným podtržením. Kompilace na pozadí negeneruje spustitelný kód v případě, že potřebuje jiný kompilátor než ten, který je použit pro generování spustitelného kódu. Kompilace na pozadí byla uvedena v Microsoft Visual Basicu, ale teď je rozšířena do všech jazyků [26].

1.2.2 Debugger

Visual Studio obsahuje funkci debuggeru, která pracuje na úrovni zdrojového kódu i na úrovni stroje. Pracuje zároveň s managed kódem i nativním kódem a může být použita pro debuggování aplikací napsaných v jakémkoliv jazyku podporovaným Visual Studií. Navíc se může také přiřadit k běžícím procesům a monitorovat a debugovat tyto procesy. Pokud je zdrojový kód běžících procesů k dispozici, zobrazuje kód, jako by právě běžel. Pokud není zdrojový kód k dispozici, může zobrazit i rozložení kódu. Debugger Visual Studia může také tvořit výpisy z paměti, které je poté možné použít pro debuggování. Více vláknové programy jsou podporovány. Debugger může být nastaven tak, aby byl spuštěn i když aplikace běžící mimo prostředí Visual Studia bude neočekávaně ukončena [27].

Debugger dovoluje nastavení break pointů (které zajišťují, že spuštěná aplikace může být dočasně zastavena v kterémkoliv bodě) a sledování (které monitorují hodnoty proměnných tak, jak probíhá vykonání programu). Breakpointy mohou být podmíněné, což znamená, že jsou dovolány jen, když je splněna podmínka. Kód může spustit jednu řádku kódu najednou, může také vstupovat do funkcí, aby provedl debugování uvnitř samotné funkce. Debugger podporuje funkci „Uprav a pokračuj“, což znamená, že kód může být upravován ve stejné chvíli, kdy je debugován. V průběhu debugování, pokud se najede ukazatelem myši na jakoukoliv proměnnou, je její aktuální hodnota zobrazena v popisku nástrojů (tooltip), kde může být také upravena, pokud je to požadováno. Během psaní kódu, debugger Visual Studio dovoluje, aby byly některé funkce odvolány ručně z okna „Bezprostředního nástroje“ [28].

1.2.3 Designer

Visual Studio obsahuje hostitele vizuálních designérů, který pomáhá při vývoji aplikací. Tyto nástroje obsahují [29]:

- ***Windows Forms Designer***

Windows Forms Designer je používán k vytvoření grafického uživatelského rozhraní (GUI) aplikací využívajících Windows Forms. Rozložení může být kontrolováno za pomoci rozmístění ovládacích prvků do jiných kontejnerů, nebo pomocí jejich uzamknutí na stranu formuláře. Zobrazená data, jak například textové pole, pole seznamu, nebo mřížka může být přiřazena k datovým zdrojům, jako například databáze, nebo dotazy. Uživatelské rozhraní (UI) je svázáno s kódem, které používá programovací model řízený událostmi. Designer generuje buď C#, nebo VB.NET kód pro aplikaci [29].

- ***WPF Designer***

WPF Designer, kódovým názvem „Cider“, byl představen ve Visual Studiu 2008. Stejně tak jako Windows Forms Designer podporuje metodu „drag and drop“. Je používána pro tvorbu uživatelských rozhraní, které cílí na Windows Presentation Foundation. Podporuje veškerou funkcionalitu WPF včetně vázání dat a automatické správy zobrazení. Generuje XAML kód pro uživatelské rozhraní. Generovaný XAML soubor je kompatibilní s Microsoft Expression designem, designově orientovaným produktem. XAML kód je spřažen s kódem, který běží na pozadí modelu [29].

- ***Web designer/development***

Visual Studio obsahuje editor webových stránek, který nabízí tvorbu stránek na základě „drag and drop“ metody. Je využíván pro tvorbu aplikací v technologii ASP.NET a podporuje HTML, CSS a JavaScript. Využívá kód na pozadí modelu, aby jej spojila s ASP.NET kódem. Od Visual Studia 2008 dále, správa zobrazení použita webovým designérem je sdílena s Microsoft Expression Web. ASP.NET také podporuje technologii MVC jako oddělený zdroj a ASP.NET Dynamic Data [29].

- ***Designer tříd (Class designer)***

Designer tříd je používán k tvorbě a úpravě tříd (včetně jejich členů a jejich přístupu) za použití UML modelování. Designer tříd může generovat obrysy kódu C# a VB.NET pro třídy a metody. Může také generovat diagram tříd z ručně psaných tříd [29].

- ***Designer mapování (Mapping designer)***

Od Visual Studia 2008 dále, je designer mapování používá LINQ, aby navrhl mapování mezi databázovými schématy a třídami, které zapouzdřují data. Nové řešení podle ORM přístupu, ADO.NET Entity Framework nahrazuje a vylepšuje starou technologii [29].

1.3 Podporované produkty

1.3.1 Microsoft Visual C++

Microsoft Visual C++ je implementací kompilátoru C a C++ od firmy Microsoft. Může kompilovat buď v módu C, nebo C++. Pro C následuje ICO C standard z roku 1990 s částmi specifikace C99 vedle knihoven specifických pro MS. Pro C++ následuje ANSI C++ specifikaci vedle několika C++11 funkcí. Podporuje také C++/CLI specifikaci pro psaní managed kódu stejně tak jako mód smíšeného kódu (mix nativního a managed kódu). Microsoft určuje Visual C++ pro vývoj v nativním kódu, nebo v kódu, který obsahuje jak nativní, tak managed komponenty. Visual C++ podporuje COM stejně tak jako MFC knihovnu. Pro vývoj MFC poskytuje soubor průvodců pro tvorbu a úpravu MFC boiler plate kódu a tvorbu aplikací s grafickým uživatelských rozhraním, které používají MFC. Visual C++ může také použít Visual Studio Forms Designer pro grafickou tvorbu uživatelského rozhraní (UI). Visual C++ může být použito s Windows API. Také podporuje použití vnitřních funkcí, jež

jsou funkcemi rozpoznány samotným kompilátorem a nejsou implementovány jako knihovna. Vnitřní funkce jsou použity k vystavení SSE instrukční sadě moderních CPU. Visual C++ také obsahuje specifikaci OpenMP [30].

1.3.2 Microsoft Visual C#

Microsoft Visual C#, jež je implementací jazyka C# firmou Microsoft, se zaměřuje na .NET Framework zároveň s jazykovými službami, které dovolují Visual Studiu IDE podporovat projekty psané v C#. Zatímco jazykové služby jsou součástí Visual Studia, kompilátor je k dispozici samostatně jako součást .NET Framework. Kompilátory Visual C# 2008, 2010 a 2012 podporují verze 3.0, 4.0 a 5.0 jazykových specifikací C#. Visual C# podporuje Visual Studio designer tříd, designer formulářů a designér dat [31].

1.3.3 Microsoft Visual Basic

Microsoft Visual Basic je implementace jazyky VB.NET od firmy Microsoft společně s nástroji a službami. Byla představena s Visual Studiem .NET v roce 2002. Microsoft určil Visual Basic jako jazyk pro Rychlý vývoj aplikací. Visual Basic může být použit jako pro konzolové aplikace, tak pro aplikace s grafickým uživatelským rozhraním (GUI). Tak jako Visual C#, Visual Basic také podporuje Visual Studio designer tříd, formulářů a dat. Stejně jako u C# je i VB.NET kompilátor dostupný jako součást .NET Framework, ale jazykové služby, které dovolují, aby byly VB.NET projekty vyvíjeny ve Visual Studiu jsou dostupné druhotně [32].

1.3.4 Microsoft Visual Web Developer

Microsoft Visual Web Developer je používán k tvorbě webových stránek, webových aplikací a webových služeb, které využívají ASP.NET. Jak C#, tak VB.NET může být použito. Visual Web Developer může použít Visual Studio Web Designer pro grafický návrh zobrazení stránky [33].

1.3.5 Team Foundation Server

Team Foundation Server je určen pro souběžný vývoj softwarových projektů a chová se jako serverový backend, který poskytuje kontrolu zdrojových kódů, sběr dat, reporting a kontrolu změn v projektu. Obsahuje také Team Explorer, klientský nástroj pro TFS služby, který je integrován uvnitř Visual Studio Team Systém [34].

1.4 Edice

Microsoft Visual Studio vychází v následujících edicích:

1.4.1 Community

Edice Community byla představena 12. listopadu 2014, jako nová volná verze, která je funkčně podobná Visual Studiu Professional. Před tímto datem jediná volná edice Visual Studia byla funkčně omezená varianta Express. Na rozdíl od verze Express, Visual Studio Community podporuje více jazyků a poskytuje podporu pro rozšíření. Visual Studio Community je orientováno na individuální vývojáře a malé týmy [35].

1.4.2 Professional

Od verze Visual Studio 2010, edice Professional je základní komerční edice Visual Studia. Poskytuje IDE pro všechny podporované programovací jazyky. Podpora MSDN je dostupná jako MSDN Essentials. Podporuje editaci XML a XSLT a může tvořit balíčky pro nasazení, které využívají technologii ClickOnce a MSI. Obsahuje nástroje jako například Server Explorer a také integraci s Microsoft SQL Serverem. Vývoj pro Windows Mobile byl zahrnut ve verzi Visual Studio 2005 Standard, avšak od verze 2008 je dostupný jen ve verzi Professional a vyšších. Vývoj pro Windows Phone 7 byl přidán do všech edic Visual Studia od verze 2010. Vývoj pro Windows Mobile není nadále podporován a byl nahrazen Windows Phone 7 [36].

1.4.3 Enterprise

Navíc k funkcionalitě obsažené v edici Professional, edice Enterprise poskytuje novou sadu nástrojů pro vývoj software, databázový vývoj, spolupráci, metriky, architekturu, testování a reportování [36].

1.4.4 Test Professional

Edice Test Professional byla představena s Visual Studií 2010. Zaměřuje se na vyhrazenou roli testera. Obsahuje podporu pro správu testovacích prostředí, schopnost spustit a reportovat testy a spojení na Team Foundation Server. Neobsahuje podporu pro vývoj, nebo tvorbu testů [37].

1.4.5 Express

Visual Studio Express je funkčně omezená verze Visual Studia pro studenty a nadšence, která byla poprvé přestavena s verzí 2005. Původně sestávala z několika edicí, kdy každá byla zaměřena na jeden programovací jazyk. Visual Studio Express 2005, 2008 a 2010 sestávaly z následujících edicí, které mohly být instalovány současně [35;36]:

- Visual Basic Express
- Visual C++ Express
- Visual C# Express
- Visual J# Express
- Visual Web Developer Express
- Visual Studio Express for Windows Phone (pouze 2010) [37]

Visual Studio 2012, 2013 a 2015 sestávalo z edicí, které byly určeny pro různé platformy:

- **Express for Web:** Zaměření na vývoj webových aplikací
- **Express for Windows:** Zaměření na vývoj aplikací pro univerzální Windows Platformu
- **Express for Desktop:** Zaměřuje se na vývoj tradičních aplikací pro Windows za užití Windows API
- **Team Foundation Server Express:** Poskytuje správu zdrojového kódu a správu životního cyklu aplikace
- **Express for Windows Phone** (pouze 2012): zaměřuje se na vývoj software pro Windows Phone 7.5 a 8.0 [37].

1.5 Historie

1.5.1 97

Microsoft vydává první verzi Visual Studia (kódový název Boston podle názvu stejnojmenného města, odtud začíná řada pojmenování spojených s místy), v roce 1997, kdy poprvé dává dohromady spoustu svých vlastních programovacích nástrojů. Visual Studio 97 vyšlo ve dvou edicích: Visual Studio Professional a Visual Studio Enterprise, edice Professional měla tři CD, Enterprise edice čtyři. Obsahovalo Visual J++ 1.1 pro programování v jazyku Java a představilo Visual InterDev pro tvorbu dynamicky generovaných webových stránek

za využití Active Server Pages. Bylo zde samostatné CD, které obsahovalo knihovnu Microsoft Developer Network [38].

Visual Studio 97 bylo prvním pokusem Microsoftu o použití stejného vývojového prostředí pro různé jazyky. Visual J++, InderDev a knihovna MSDN byly všechny používaly stejné prostředí zvané Developer Studio [38].

Visual Studio bylo také prodáváno jako balení s oddělenými IDE, které mohly být použity pro Visual C++, Visual Basic a Visual FoxPro [38].

1.5.2 6.0 (1998)

Nadcházející verze 6.0 (kódový název Aspen, podle lyžařského střediska v Coloradu), byla vydána v červnu 1998 a je to poslední verze, která běží na platformě Windows 9x. Všechny verze obsažených jazyků byly označeny jako 6.0, včetně Visual J++, jehož předchozí verze byla označena jako 1.1. Verze 6.0 byla základním prostředím pro další čtyři vydání, které poskytla programátorům integrovanou stejně vypadající platformu. Toto vedlo Microsoft k přechodu vývoj na platformě nezávislý .NET Framework [39].

Visual Studio 6.0 bylo poslední verzí, která obsahovala Visual J++, které Microsoft odstranil jako část dohody s firmou Sun Microsystems, která vyžadovala, aby Microsoft Internet Explorer neposkytoval podporu pro Java Virtual Machine [39].

Visual Studio 6.0 vyšlo ve dvou edicích: Professional a Enterprise. Edice Enterprise obsahovala extra obsah, který nebyl k dispozici v edici Professional jako například [39]:

- Application Performance Explorer
- Automation Manager
- Microsoft Visual Modeler
- RemAuto Connection Manager
- Visual Studio Analyzer [39]

1.5.3 .NET (2002)

Microsoft vydal Visual Studio .NET (VS.NET), s kódovým označením Rainier (podle Washingtonské hory Mount Rainier) v únoru 2002. Největší změna byla představení vývojového prostředí pro managed kód, které využívalo .NET Framework. Programy, pro jejichž vývoj je využit .NET nejsou kompilovány do strojového kódu, tak jako například C++, ale

namísto toho se jedná o formát nazvaný Microsoft Intermediate Language (MSIL), nebo Common Intermediate Language (CIL). Pokud je CIL aplikace spuštěna, je kompilována za běhu do vhodného strojového kódu pro platformu, na které právě běží, tudíž je kód snadněji přenositelný mezi platformami. Programy kompilované do CIL mohou být spuštěny pouze na platformách, které mají implementaci Common Language Infrastructure. Je možné spustit CIL programy na Linuxu, Mac OS X za použití ne-Microsoftích NET implementací jako například Mono a DotGNU [40].

Toto byla první verze Visual Studia, která vyžadovala Windows platformu na základě technologie NT. Tento požadavek byl kontrolován během instalace [40].

Visual Studio .NET 2002 bylo dodáváno ve čtyřech edicích: Academic, Professional, Enterprise Developer a Enterprise Architect. Microsoft představil C# (C-sharp), nový programovací jazyk, který se zaměřuje na .NET. Také představil následníka Visual J++, nazvaný Visual J#. Visual J# programy používají syntaxi jazyka Java. Nicméně oproti Visual J++ programům, Visual J# programy mohou využívat pouze .NET Framework, nikoliv Java Virtual Machine, kterou využívají všechny ostatní Java nástroje [40].

Visual Basic se výrazně změnil, aby byl v souladu s novým Framework a novou verzí nazvanou Visual Basic .NET. Microsoft také přidal rozšíření pro C++ nazvané Managed Extensions for C++, takže .NET programy mohou být tvořeny v C++ [40].

Visual Studio .NET může být využito pro tvorbu aplikací pro Windows (za použití Windows Forms, součástí .NET Framework), Web (za použití ASP.NET a Webových služeb) a přenosných zařízení (za použití .NET Compact Framework) [40].

Interní číslo verze Visual Studia .NET 2002 je 7.0. V březnu 2005 Byl vydán Service Pack 1 pro VS.NET 2002 [40].

1.5.4 .NET (2003)

V dubnu 2003 představil Microsoft drobný upgrade pro Visual Studio nazvaný Visual Studio .NET 2003, s kódovým označením Everett (podle stejnojmenného města). Obsahuje upgrade na .NET Framework 1.1 a je prvním vydáním, které podporuje vývoj programů na mobilních zařízeních za použití ASP.NET, nebo .NET compact Framework. Standardy kompilátoru Visual C++ byly vylepšeny. Sada nástrojů Visual C++ 2003 je verzí stejného kompilátoru C++ dodávaného s Visual Studií .NET 2003 bez IDE, které Microsoft poskytl zdarma

k dispozici. Od roku 2010 už není k dispozici a byla nahrazena Express edicí. Interní číslo verze Visual Studio .NET 2003 je verze 7.1 [41].

Visual Studio .NET 2003 bylo dodáváno ve čtyřech edicích: Academic, Professional, Enterprise Developer a Enterprise Architect. Visual Studio .NET 2003 Enterprise Architect obsahuje implementaci modelovacích technologií Microsoft Visio 2002, obsahující nástroje pro UML reprezentaci aplikační architektury [41].

Service Pack 1 byl vydán 13. září 2006 [41].

1.5.5 2005

Visual Studio 2005 s kódovým názvem Whidbey (odkazující se na ostrov Whidbey) byl vydán online v říjnu 2005 a na pulty obchodů se dostal o pár týdnů později. Microsoft odstranil „.NET“ přídomek z Visual Studia 2005, ale to se i nadále primárně zaměřuje na .NET Framework, který byl upgradován na verzi 2.0. Je to poslední verze k dispozici pro Windows 2000 a poslední verze, která se zaměřuje na Windows 98, Windows Me a Windows NT 4.0 pro C++ aplikace [42].

Visual Studio 2005 má interní číslo 8.0 a bylo upgradováno, aby podporovalo všechny nové funkce, které byly představeny v .NET Framework 2.0. Funkce IntelliSense byla upgradována pro nové projektové typy a pro podporu ASP.NET webových služeb. VS 2005 také obsahuje lokální webový server oddělený od IIS, který může hostit ASP.NET aplikace během vývoje a testování. Také podporuje všechny SQL Server 2005 databáze. Databázové designéry byly upgradovány pro podporu ADO.NET 2.0, které je obsaženo v .NET Framework 2.0. Visual Studio také přidalo rozsáhlou podporu pro 64-bitovou architekturu. Zatímco samo hostitelské vývojové prostředí je dostupné pouze jako 32-bitové, Visual Studio 2005 podporuje kompilaci pro x86-64 (AMD64 a Intel 64) stejně tak jako IA-64 Itanium [42].

Microsoft oznámil, že Visual Studio Tools pro Aplikace bude následníkem Visual Basic pro aplikace (VBA) a VSA (Visual Studio for Applications). VSTA 1.0 byl vydán společně s Office 2007. Aplikace Office 2007 pokračují v integraci s VBA, kromě InfoPath 2006, který je integrován s VSTA [42].

1.5.6 2008

Visual Studio 2008 a Visual Studio Team System 2008 s kódovým označením Orcas (s odkazem na ostrov Orcas), byly vydány pro odběratele MSDN 19. listopadu 2007 zároveň s .NET Framework 3.5. Zdrojový kód Visual Studio 2007 IDE je dostupný pod sdílenou licenci s některými Microsoft partnery a ISV. Microsoft vydal Service Pack 1 pro Visual Studio 2008 11. srpna 2008. Interní číslo verze Visual Studia 2008 je verze 9.0 zatímco verze formátu souborů je 10.0. Visual Studio 2008 je poslední verze, které podporuje vývoj aplikací C++ pro Windows 2000 [42].

Visual Studio 2008 se zaměřuje na vývoj pro Windows Vista, Office 2007 a Webové aplikace. Pro vizuální design je k dispozici nový designér pro Windows Presentation Foundation a nový editor HTML/CSS ovlivněný Microsoft Expression Webem. J# není k dispozici. Visual Studio 2008 vyžaduje .NET Framework 3.5 a jako výchozí konfiguruje kompilovaný zdrojový kód tak, aby běžel pod tímto frameworkem, ale také podporuje možnost, aby si programátor vybral vývoj i pro jiné frameworky, jako například .NET 2.0, 3.0, 3.5, Silverlight CoreCLR, nebo .NET Compact. Visual Studio 2008 také obsahuje nové nástroje pro analýzu kódu, jako například nový nástroj Code Metrics. Pro Visual C++ Visual Studio přidává novou verzi Microsoft Foundation Classes (MFC 9.0), které přidávají podporu pro vizuální styly a ovládání UI, které přinesly Windows Vista [42].

Visual Studio 2008 obsahuje integrovaný designér založený na XAML, designér pracovních procesů, LINQ a SQL designér, XSLT debugger, podporu JavaScript Intellisense a podporu JavaScript debuggingu. Obsahuje také více vláknový build engine (MSBuils), který kompiluje zdroje ikon ve formátu PNG. Designér pro XML schéma byl vydán samostatně nějakou dobu po vydání Visual Studia 2008 [42].

Visual Studio Debugger obsahuje funkce, které cílí na jednodušší debugging více vláknových aplikací. V debugging módu je okno vláken, které obsahuje seznam všech vláken. Vlákna mohou být přímo pojmenovávána a označována pro snazší identifikaci z okna samotného. Navíc v okně kódu, vedle indikace lokace právě prováděné instrukce v aktuálním vlákně mohou být tyto instrukce zvýrazněny. Debugger Visual Studia podporuje integrované debuggování základních knihoven tříd (BCL) založených na .NET 3.5 Framework, které mohou dynamicky stahovat zdrojové kódy BCL a použít je pro debuggování zdrojového kódu. Od roku 2010 je k dispozici širší nabídka BCL zdrojů s větší podporou plánovanou do budoucna [42].

1.5.7 2010

12. dubna 2010 vydal Microsoft Visual Studio 2010 s kódovým označením Dev10 a .NET Framework 4 [42].

IDE Visual Studio 2010 bylo re designováno což, dle Microsoftu vyčistilo organizaci uživatelského rozhraní. Nové IDE lépe podporuje okna s více dokumenty a plovoucí okna nástrojů a k tomu podporuje použití více monitorů. Jádro IDE bylo přepsáno za použití Windows Presentation Foundation (WPF), zatímco interní kód byl re designován za použití Managed Extensibility Framework (MEF) který nabízí širší rozšiřitelnost IDE, než mohly nabídnout předchozí verze. Nová multi-paradigm ML varianta F# formulářů je také součástí Visual Studio 2010 [42].

Visual Studio 2010 přichází s .NET Framework 4 a podporuje vývoj aplikací zaměřených na Windows 7. Podporuje databáze IBM DB2 a Oracle vedle standardní podpory Microsoft SQL Serveru. Má integrovanou pro vývoj aplikací Microsoft Silverlight včetně interaktivního designéru. Visual Studio 2010 nabízí několik nástrojů, které usnadňují paralelní programování [42].

Editor kódu Visual Studio 2010 nyní označuje reference, pokud je vybrán symbol, také poskytuje možnost rychlého hledání, která prohledává inkrementálně všechny symboly v projektech v C++, C# a VB.NET. Rychlé hledání podporuje kontrolu shody podřetězců a „camelCase“ hledání. Ve verzi Visual Studio 2010 byla chyba, která znemožňovala použití IntelliSense v projektech čistého jazyka C [42].

Visual Studio 2010 už nepodporuje vývoj pro platformu Windows Mobile před verzí Windows Phone 7 [42].

1.5.8 Ultimate 2010

Visual Studio Ultimate 2010 nahrazuje Visual Studio 2008 Team Suite. Obsahuje nové modelovací nástroje, jako například Architecture Explorer, který graficky zobrazí projekty a třídy a vztahy mezi nimi. Podporuje UML aktivity diagramy, diagramy komponent, diagramy tříd, sekvenční diagramy a use case diagramy. Visual Studio Ultimate 2010 také obsahuje analýzu dopadů testování, které poskytuje nápovědu k tomu, jaké test casey jsou ovlivněny modifikací zdrojového kódu, aniž by musel samotný test case proběhnout. Toto zvyšuje rychlost testování tím, že se vyhýbá testům, které nejsou nezbytné [42].

Visual Studio Ultimate 2010 také obsahuje historický debugger pro managed kód, nazvaný IntelliTrace. Na rozdíl od tradičního debuggeru, který zaznamenává pouze aktivní zásobník, IntelliTrace zaznamenává všechny události, jako například volání před funkcí, parametry metod, události a výjimky [42].

1.5.9 2012

Finální verze Visual Studia 2012 byla oznámena 1. srpna 2012 a oficiální událost vydání byla 12. září 2012 [43].

Na rozdíl od předchozích verzí, Visual Studio 2012 nemůže nahrávat a spouštět makra a editor maker byl odstraněn [43].

Nové funkce obsahují podporu WinRT a C++/CX a C+ AMP (Programování GPGPU) sémantické zbarvování [43].

16. září 2011, bylo vydáno kompletní Developer Preview Visual Studia 11 na webových stránkách Microsoftu. Developer Preview Visual Studia 11 požaduje Windows 7, Windows Server 2008 R2, Windows 8, nebo pozdější operační systémy. Verze Microsoft Foundation Class Library (MFC) a C runtime (CRT) obsaženy v tomto vydání nemůže produkovat software, který je kompatibilní s Windows XP, nebo Windows Server 2003 s výjimkou použití nativního multi-targetingu a použití nejnovějších knihoven, kompilátorů a hlaviček [43].

Zdrojový kód Visual Studia 2012 sestává z přibližně 50 milionů řádků kódu [43].

1.5.10 2013

Preview pro Visual Studio 2013 bylo oznámeno v roce 2013 a k dispozici bylo 26. června téhož roku. Visual Studio 2013 RC (Release Candidate) bylo dáno k dispozici vývojářům prostřednictvím MSDN 9. září 2013 [44].

Finální release Visual Studia 2013 byl k dispozici pro stažení 17. října 2013 společně s .NET 4.5.1. Visual Studio 2013 bylo oficiálně vydáno 13. listopadu 2013 [44].

1.5.11 2015

Původně označováno jako Visual Studio 14, první Community Technology Preview (CTP) bylo vydáno 3. června 2014 a Release Candidate byl vydán 29. dubna 2015 [45].

1.5.12 2017

Původně označováno jako Visual Studio 15, bylo vydáno 7. března 2017. První preview bylo vydáno 30. března 2016. 5. dubna 2017, Visual Studio 2016 15.1 bylo vydáno a byla přidána podpora, která cílí na .NET Framework 4.7 [46].

Visual Studio 2017 nabízí vlastnosti jako podporu EditorConfig, NGen podporu, sady nástrojů .NET Core a Docker toolset a Xamarin 4.3. Také obsahuje XAML editor, vylepšené IntelliSense, živé unit testování, rozšíření pro debuggování a lepší IDE [46].

2 JAVASCRIPT OBJECT NOTATION (JSON)

JavaScript Object Notation neboli JSON je otevřený standard, který používá lidsky čitelný text pro přenos datových objektů, které se skládají z párů atributů a hodnot. Je to velmi obvyklý datový formát používaný pro asynchronní komunikaci mezi prohlížečem a serverem, kdy je používán jako náhrada XML v některých na AJAX technologii založených systémech [2;3]

JSON je jazykově nezávislý datový formát. Je odvozen od JavaScriptu, ale mnoho současných programovacích jazyků obsahují kód pro generování a parsování datového formátu JSON. Oficiální typ média na internetu pro JSON je „application/json“. Soubory JSON používají koncovku „.json“ [3].

Douglas Crockford původně specifikoval formát JSON v raných 2000. Dva standardy RFC 7159 a ECMA-404 byly definovány v roce 2013. Standard ECMA popisuje pouze povolenou syntaxi, zatímco RFC pokrývá i některé bezpečnostní otázky [4].

2.1 Historie

JSON vznikl kvůli potřebě stavového, v reálném čase běžícím komunikačního protokolu, který by byl schopen používat pluginy prohlížeče jako například Flash, nebo Java applets.

Douglas Crockford prvně specifikoval a popularizoval formát JSON v roce 2001 [5].

2.2 Datové typy

Základní datové typy formátu JSON jsou:

2.2.1 Číslo

Znaménkem odlišené desetinné číslo, které může obsahovat část zlomku a může obsahovat exponenciální notaci E, ale nemůže obsahovat nečíselné hodnoty, jako jsou například NaN. Formát nerozlišuje mezi typem integer a číslem s plovoucí desetinnou čárkou. JavaScript používá formát plovoucí desetinné čárky s dvojitou přesností pro všechny jeho číselné hodnoty, ale ostatní jazyky implementující JSON mohou používat formát čísel jiným způsobem [7].

2.2.2 Řetězec

Sekvence žádného, nebo více znaků ve formátování Unicode. Řetězce jsou odděleny dvojitými uvozovkami a podporují escape syntaxi za použití zpětného lomítka [7].

2.2.3 Boolean

Hodnota nabývající hodnot „true“, nebo „false“ [7].

2.2.4 Pole

Seřazený seznam žádné, nebo více hodnot, kdy každá z nich může být libovolného typu. Pole používají hranaté závorky a jejich prvky jsou odděleny čárkami [7].

2.2.5 Objekt

Neseřazená sbírka párů jméno/hodnota, kdy jména (někdy zvané klíče) jsou řetězce [7]. Pokud objekty jsou určeny pro reprezentaci svázaných polí, je doporučeno, nicméně ne požadováno, aby byl každý klíč unikátní v rámci objektu. Objekty jsou odděleny složenými závkami a používají čárky pro oddělení každého páru, zatímco dvojtečka odděluje klíč od jeho hodnoty [6].

2.2.6 Null

Prázdná hodnota, používá slovo „null“.

Je možné používat omezený prázdný prostor, za který jsou považovány pouze čtyři znaky a to mezera, horizontální tabelátor, line feed (LF) a carriage return (CR). JSON nepodporuje žádnou syntaxi pro komentáře.

Dřívější verze JSON (například uvedená ve standardu RFC 4627) požaduje, aby platný dokument JSON obsahoval pouze objekty, nebo typy polí, které mohou obsahovat jiné typy uvnitř [6].

2.3 Příklad

Následující příklad zobrazuje možnou reprezentaci pomocí formátu JSON, která popisuje osobu.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Obr. 2. Příklad použití formátu JSON [8]

2.4 Použití JSON v JavaScriptu

Vzhledem k tomu, že JSON byl odvozen od JavaScriptu a jeho syntaxe je z velké části podmnožinou jazyka, je možné použít JavaScript „eval()“ funkci k parsování dat JSON. Kvůli problému s parsováním ukončovacích znaků řádků ve formátování Unicode potřebuje funkce eval() provést nahrazení řetězců [9].

Tento postup je nebezpečný, pokud je řetězec nedůvěryhodný, podpora formátu JSON by měla být použita jak pro čtení, tak pro zápis souboru JSON [9].

Korektně implementovaný parser formátu JSON akceptuje pouze validní soubory typu JSON a chrání před vykonáním potenciálně škodlivého kódu [9].

Od roku 2010 mají webové prohlížeče jako například Firefox a Internet Explorer integrovanou podporu pro parsování formátu JSON. Nativní podpora je více efektivní a bezpečná než metoda „eval()“, nativní podpora JSON je zahrnuta v Edici 5 ECMAScriptu [9].

2.5 Nepodporované nativní datové typy

Syntaxe JavaScriptu definuje několik nativních datových typů, které nejsou zahrnuty ve standardu JSON. Mezi tyto datové typy patří: Map, Set, Date, Error, Regular Expression, Function, Promise a „nedefinováno“. Tyto datové typy JavaScriptu musí být reprezentovány jiným datovým formátem s programy na obou stranách, které jsou synchronizovány v otázce jak konvertovat mezi danými typy. Od roku 2011 existují některé standardy, například konverze data do řetězce, ale žádný z nich není univerzálně rozpoznáván. Ostatní jazyky mohou mít rozdílnou sadu nativních typů, které musí být opatrně serializovány, aby si poradily s daným typem konverze [7;10].

2.6 Schéma JSON

Schéma JSON specifikuje formát založený na JSON, který definuje strukturu dat pro validaci, dokumentaci a kontrolu interakce [11]. Poskytuje kontrakt pro JSON data, které jsou požadovány danou aplikací, a popisuje, jak mohou být data upravována [12].

Schéma JSON je založena na konceptech XML schématu (XSD), ale je založen na formátu JSON [11]. Stejně jako v XSD šabloně mohou být použity stejné nástroje pro serializaci a deserializaci [12]. Existuje několik validátorů, které jsou k dispozici pro různé programovací jazyky, každý z nich s různou úrovní shody [13].

2.7 Bezpečnostní rizika

JSON je určen k použití jako formát serializace dat [9]. Nicméně jeho design jako nestriktní podmnožina JavaScriptového skriptovacího jazyka představuje několik bezpečnostních obav. Tyto obavy soustřeďují střed použití JavaScriptového interpreteru pro vykonání JSON textu dynamicky jako zabudovaný JavaScript. Toto vystavuje program chybným, nebo škodlivým skriptům. Jedná se o vážný problém, pokud se jedná o práci s daty získanými z internetu. Tato jednoduchá, ale riskantní technika zneužívá kompatibility JSON s JavaScriptovou „eval()“ funkcí [11].

2.8 Srovnání s jinými formáty

JSON je vyzdvihován jako nízko režijní alternativy k formátu XML, oba tyto formáty mají širokou podporu pro tvorbu, čtení a dekodování v reálných situacích, kde jsou běžně používány. Na rozdíl od XML, příklady mohou obsahovat OGDŁ, YAML, nebo CSV. Mimo tyto i Google Protocol Buffery mohou plnit tuto roli, nicméně se nejedná o jazyk určený pro výměnu dat [14].

2.8.1 YAML

YAML verze 1.2 je nadřazenou sadou k JSON, předchozí verze nebyly „striktně kompatibilní“. Jako příklad je možné uvést uvozování lomítka (/) za pomoci zpětného lomítka (\), které je validní ve formátu JSON, ale už ne v YAML. I přesto dokáže mnoho YAML parserů nativně parsovat výstup z mnoha JSON enkoderů [15].

2.8.2 XML

XML byl používán k popisu strukturovaných dat a serializaci objektů. Rozličné protokoly založené na XML reprezentují stejné druhy datových struktur jako JSON pro stejný druh výměny dat. Data mohou být kódována do XML různými způsoby. Nejvíce expanzivní metoda je využití dvojic tagů, která ústí ve větší soubor, než například srovnatelný soubor JSON. Pokud jsou data uložena v attributech v krátké tag formě, je uzavření tagu nahrazeno formátem „/>“, reprezentace je často shodně velká jako JSON, nebo o něco větší. Pokud jsou data komprimována za použití například metody gzip, je jen malý rozdíl, protože komprese je účinná v případech, kdy se opakuje daný vzor v souboru [11].

XML má také koncept schématu. Toto dovoluje použít silné typování, uživatelský definované typy, předdefinované tagy a formální strukturu, které dovoluje formální validaci XML proudu v přenosné formě [11].

XML na rozdíl od formátu JSON podporuje komentáře [11].

3 SQLITE

SQLite je databázový systém pro správu relačních databází obsažený v knihovně programovacího jazyk C. Oproti jiným systémům pro správu databáze neběží SQLite na principu klient-server, namísto toho je vestavěn přímo do programu [47].

SQLite je v souladu s technologií ACID (Atomicity = atomicita, Consistency = konzistence, Isolation = izolace, Durability = trvanlivost), což je sada vlastností databázových transakcí a implementuje většinu standardů SQL za použití dynamické a slabě typované SQL syntaxe která nezaručuje doménovou integritu [47].

SQLite je populární volba v případě použití vestavěného databázového software pro lokální/klientské uložení v aplikačním software jako jsou například webové prohlížeče. Je to pravděpodobně nejčastěji používaný databázový engine, který je dnes používán mnoha prohlížeči, operačními systémy a vestavěnými systémy, jako jsou například mobilní telefony a jiné. SQLite obsahuje vazby na mnoho programovacích jazyků [47].

3.1 Design

Oproti systémům pro správu databáze na úrovni klient-server, engine SQLite neobsahuje žádný samostatný proces, pomocí kterého by aplikační komunikoval. Namísto toho je knihovna SQLite spojena s aplikací a díky tomu se stává součástí aplikačního programu. Knihovna může být také volána dynamicky [47]. Program aplikace používá funkcionalitu SQLite skrze jednoduché volání funkcí, které snižují zpoždění v přístupu do databáze. Volání funkcí skrze samostatný proces jsou více efektivní než komunikace mezi procesy. SQLite uchovává celou databázi včetně definic, tabulek, indexů a samotných dat v rámci jednoho souboru, který je možné používat skrze platformy na hostitelském stroji. Implementuje tento jednoduchý princip pomocí zamčení celého databázového souboru během zápisu. Operace čtení v rámci SQLite může být provedena více úlohami paralelně, nicméně čtení může být prováděno pouze sekvenčně [48].

Díky návrhu neobsahujícím server, aplikace běžící na základě SQLite nepotřebují tolik konfigurace jako databáze běžící v konfiguraci klient-server. SQLite je nazýván „bez konfigurační“ kvůli tomu, že nepotřebuje žádnou konfiguraci služeb, nebo kontrolu přístupových hesel a práv. Ověření přístupu je řízeno na základě oprávnění souborového systému, který jí dává samotné databázi [48].

Jedním z dalších důsledků architektury bez serveru je to, že může být třeba, aby více procesů mělo možnost zapisovat do databázového souboru. V databázích založených na serveru [49]

3.2 Příklad aplikace kódu na základě SQLite

```
#include <stdio.h>
#include "sqlite3.h"
int main(void)
{
    sqlite3* db = 0;
    sqlite3_stmt* stmt = 0;
    int retcode;
    retcode = sqlite3_open("MyDB", &db); /* Open a database named
    MyDB */
    if (retcode != SQLITE_OK) {
        sqlite3_close(db);
        fprintf(stderr, "Could not open MyDB\n");
        return retcode;
    }
    retcode = sqlite3_prepare(db, "select SID from Students order
    by SID",
        -1, &stmt, 0);
    if (retcode != SQLITE_OK) {
        sqlite3_close(db);
        fprintf(stderr, "Could not execute SELECT\n");
        return retcode;
    }
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        int i = sqlite3_column_int(stmt, 0);
        printf("SID = %d\n", i);
    }
    sqlite3_finalize(stmt);
    sqlite3_close(db);
    return SQLITE_OK;
} [49]
```

3.3 Historie SQLite

Dwayne Richard Hipp (narozen 9. dubna 1961) navrhl SQLite v roce 2000, když pracoval pro firmu General Dynamics na kontraktu pro námořnictvo Spojených Států Amerických. Hipp navrhoval software pro použití na zařízeních pro protiraketovou ochranu, které původně používaly databáze HP-UX a IBM Informix. SQLite začal původně jako rozšíření Tcl [50].

Cíle při návrhu SQLite byly dovolit programu, aby mohl pracovat s databází, aniž by bylo třeba instalovat systém pro správu databáze a aniž by bylo třeba databázového správce. Hipp založil syntaxi a sémantiku na PostgreSQL 6.5. V srpnu 2000 vyšla první verze SQLite 1.0

s úložištěm založeným na základech gdbm (Správce databáze GNU). SQLite verze 2.0 nahradila gdbm modifikovatelnou implementací B-stromu a přidala schopnost pracovat s transakcemi. SQLite 3.0 přidala národní nastavení a další hlavní vylepšení [48;49].

V roce 2011 Hipp oznámil, že plánuje přidat UnQL rozhraní k SQLite databázím a vyvinout Uniaté, vestavitelnou dokumentově orientovanou databázi [48].

3.4 Funkce

SQLite implementuje většinu standardu SQL-92, ale chybí mu některé funkce. Například částečně poskytuje trigger, ale neumí zapisovat do view (přestože poskytuje `INSTEAD OF` trigger, které nabízejí tuto funkcionalitu). Přestože poskytuje použití komplexních dotazů, stále má omezenou funkci `ALTER TABLE`, protože nedokáže upravovat, nebo mazat sloupce [49].

SQLite používá neobvyklý systém typů pro DBMS kompatibilní s SQL. Namísto přiřazení typu ke sloupci, tak jako ve většině SQL databázových systémů, typy jsou přiřazeny jednotlivým hodnotám. V pravidlech jazyka je dynamicky typována. Navíc je slabě typován v některých způsobech stejně, jako je i Perl. Jeden z nich je vkládání řetězce do pole integeru (přestože se SQLite nejprve pokusí konvertovat řetězec na celé číslo, pokud je preferovanou hodnotou pole integer). Toto dodává sloupcům na pružnosti, obzvlášť v případě, že jsou navázány na dynamicky typovaný skriptovací jazyk. Tato technika však není přenositelná na jiné SQL produkty [49].

Více počítačových procesů, nebo vláken může přistoupit ke stejné databázi konkurenčně. Více přístupů pro čtení může být obslouženo paralelně. Přístup k zápisu může být obsloužen pouze v případě, že nikdo jiný v danou chvíli nepřistupuje k souboru, jinak přístup pro zápis skončí na chybovém kódu. Tento případ konkurenčního přístupu by se změnil, pokud by se jednalo o práci s dočasnými tabulkami. Toto omezení bylo odstraněno ve verzi 3.7, pokud je zapnuta funkce WAL (write-ahead logging) a dovoluje konkurenční čtení i zápis [49].

SQLite verze 3.7.4 přidává modul s funkcí FTS4 (Fulltextové vyhledávání), který rozšiřuje možnosti předchozí verze modulu FTS3. FT4 dovoluje uživateli provádět fulltextové hledání nad dokumenty podobně, jako to dělá například engine webových stránek. Verze 3.8.2 přidává možnost tvorby tabulek bez rowid, které může poskytnout prostor pro vylepšení výkonu. SQLite s plnou podporou Unicode je volitelné [49].

4 SIGNALR

SignalR je software pracující na serverové části navržený pro psaní škálovatelných webových aplikací pracujících v internetu, převážně webové servery. Programy jsou psány v jazyku .NET za použití událostmi řízeným asynchronním vstupem a výstupem pro poskytnutí maximální škálovatelnosti za minimálních režijních nákladů [1].

4.1 Popis

SignalR je ASP.NET knihovna pro ASP.NET vývojáře, která jejich aplikacím dovoluje přidat funkcionalitu pracující v reálném čase. Funkcionalita pracujících v reálném čase je schopnost poslat kód ze serveru na klienty dle libosti a v reálném čase [51].

SignalR využívá možnosti několika druhů přenosu, automaticky vybírá nejlepší dostupný přenos, který poskytuje klient a server. Využívá WebSocket a HTML5 API, které dovoluje dvousměrnou komunikaci mezi prohlížečem a serverem. SignalR použije WebSockets pokud jsou dostupné a elegantně využije jiných technik a technologií v případě, že dostupné nejsou, zatímco kód aplikace zůstává nezměněn [51].

SignalR také poskytuje jednoduché, vysokoúrovňové API pro práci s RPC mezi serverem a klientem v rámci ASP.NET aplikace, stejně tak, jak přidání užitečných nástrojů pro správu spojení, jako například události pro spojení a odpojení, grupování spojení a autentifikace [51].

4.2 Přenos dat a záloha

SignalR je určitou abstrakcí přes určité druhy transportů, které jsou třeba pro komunikaci mezi klientem a serverem v reálném čase. WebSocket vyžaduje použití Windows Server 2012, nebo Windows 8 na serveru s .NET Framework 4.5, v opačném případě využije jiný druh přenosu dat [51].

4.2.1 HTML 5 druhy přenosu

Tyto druhy přenosu dat závisejí na podpoře HTML 5. Pokud prohlížeč klienta nepodporuje standard HTML 5, bude využitý starší typ přenosu [51].

- **WebSocket** (Pokud jak server, tak prohlížeč podporují Websocket). WebSocket je jediný druh přenosu, který poskytuje skutečně trvalé, obousměrné spojení mezi klientem a serverem. Nicméně WebSocket má také nejpřísnější požadavky. Je plně podporován jen v posledních verzích Microsoft Internet Explorer, Google Chrome a Mozilla Firefox a má jen částečnou implementaci v prohlížečích jako jsou Opera a Safari [51].
- **Server Sent Events**, také známé jako EventSource (pokud prohlížeč podporuje Server Sent Events, což jsou v podstatě všechny prohlížeče kromě Internet Exploreru.) [51].

4.2.2 Comet přenos dat

Následující druh přenosu dat je založený na webovém aplikačním modelu Comet, ve kterém prohlížeč, nebo klient spravuje dlouhodobě držený HTTP dotaz, který může použít server k zaslání dat na klienta, aniž by ho klient specificky vyžadoval [51].

- **Forever Frame** (pouze pro Internet Explorer). Forever Frame tvoří skrytý IFrame, který dovoluje nekompletní dotaz na cílový bod na serveru. Server potom kontinuálně zasílá skript klientovi a ten je ihned vykonán. Ve skutečnosti se jedná o jednosměrné spojení v reálném čase ze serveru na klienta. Spojení z klienta na server používá oddělené spojení a stejně jako standardní HTML dotaz, je nové spojení vytvořeno pro každý kus dat, které musí být zaslány [51].
- **Ajax long polling**. Long polling netvoří trvalé spojení, ale namísto toho kontaktuje server s dotazem, který zůstane otevřený tak dlouho, dokud server neodpoví, čímž se ukončí spojení a ihned je vyžádáno nové spojení. Tento způsob může obsahovat určité zpoždění v době, kdy se resetuje spojení [51].

4.2.3 Metoda výběru přenosu

Následující seznam popisuje, jakým způsobem SignalR rozhoduje, který druh přenosu použije [52].

1. Pokud je prohlížeč Internet Explorer 8, nebo starší, využívá se Long Polling [52].
2. Pokud je nakonfigurován JSONP (to znamená, že parametr „jsonp“ je nastaven na hodnotu „true“, když je navázáno spojení), využívá se Long Polling [52].

3. Pokud je použito spojení napříč doménami (což znamená, že cíl vlání SingalR není stejný, jako doména na hostitelově stránce), využívá se WebSocket, pokud jsou splněna následující kritéria [52]:
 - Klient podporuje CORS (Cross-Origin Resource Sharing).
 - Klient podporuje WebSocket
 - Server podporuje WebSocket

Pokud není vyhověno jakémukoliv z těchto kritérií, využívá se Long Polling [52].
4. Pokud není konfigurován JSONP a spojení není napříč doménami, využívá se WebSocket, pokud je podporován oběma stranami [52].
5. Pokud buď klient, nebo server nepodporuje WebSocket, používá se Sever Sent Events [52].
6. Pokud Server Sent Events není k dispozici, pokouší se SignalR o použití Forever Frame [52].
7. Pokud komunikace přes Forever Frame selže, využívá se Long Polling [52].

5 INTERNET OF THINGS (IOT)

Internet věcí (IoT) je síťová komunikace mezi fyzickými zařízeními, vozidly, budovami a ostatními objekty, které jsou zabudovány do elektroniky, software, senzorů, pohonů a síťovou konektivitou, která zajišťuje, že tyto objekty mohou sbírat a vyměňovat si data [19]. V roce 2013 „Global Standards Initiation Internet of Things“ (IoT-GSI) definoval IoT jako „Infrastrukturu informační společnosti“ [20;21]. IoT dovoluje objektům, aby mohly být vzdáleně ovládány, nebo sledovány skrz existující síťovou infrastrukturu a vytvářet příležitosti pro přímější integraci fyzického světa do počítačem řízených systémů, což vede ke zvýšení efektivity, přesnosti a ekonomických výhod současně se snížením lidských zásahů. [22;23;24] Když je IoT rozšířen o senzory a pohony, technologie se stane instancí obecnější třídy kyber-fyzických systémů, které také zahrnují technologie, jakými jsou například chytré mřížky, virtuální elektrárny, chytré domácnosti, inteligentní přeprava a chytrá města. Každá věc je unikátně identifikovatelná skrze její vestavěný počítačový systém, ale je schopna spolupracovat v rámci existující internetové infrastruktury. Experti odhadují, že IoT bude koncem roku 2020 sestávat ze zhruba 30 miliard objektů [25].

6 ENTITY FRAMEWORK

Entity Framework je objektově-relační open source framework pro ADO.NET. Byl součástí .NET Framework, ale od verze 6 je oddělen [53].

6.1 Přehled

Entity Framework je sada technologií používaných v ADO.NET, které podporují vývoj datově orientovaných aplikací. Architekti a vývojáři datově orientovaných aplikací se často potýkají s potřebou dosažení dvou velmi rozdílných cílů. Musí modelovat entity, vztahy a logiku obchodních problémů, ale musejí také pracovat s datovými enginy, které se používají k uložení a nalezení dat. Data mohou být roztažena po více úložných systémech, každý se svým vlastním protokolem, nebo dokonce vlastním systémem ukládání dat a je tak velmi složité napsat efektivní kód, který dokáže tento druh aplikací spravovat [53].

Entity Framework dává vývojářům možnost pracovat s daty ve formě doménově specifických objektů a vlastností, jako například zákazníci a zákaznické adresy, aniž by se museli starat o to, jak konkrétně fungují databázové tabulky a pole, kde jsou data uložena. Díky Entity Frameworku mohou vývojáři pracovat na vyšší úrovni abstrakce při práci s daty a mohou vytvářet a spravovat datově orientované aplikace pomocí mnohem efektivnějšího a minimalističtějšího kódu [53].

7 REŠERŠE EXISTUJÍCÍCH ŘEŠENÍ

Jako základ pro hledání toho nejlepšího možného řešení byla navržena rešerše již existujících řešení. Díky tomu bude možné porovnat jejich nabízenou funkcionalitu, zjistit klady i zápory daného řešení a zjistit náklad na provoz toho či onoho produktu. Jak se v průzkumu ukázalo, na trhu je velké množství produktů, které by mohly pokrýt část, nebo i všechny výše uvedené požadavky, ne vždy ale plně vyhovovaly jak z funkčního, tak cenového hlediska.

Pro další rešerši jsem vybral následující produkty:

- System Center Operations Manager
- SolarWinds NPM
- AppDynamics
- DynaTrace
- Hewlett Packard Enterprise

7.1 System Center Operations Manager

System Center Operations Manager (SCOM) je komponenta Microsoft System Center 2012. Tento software pomáhá monitorovat služby, zařízení a operace pro mnoho počítačů z jedné konzole. Používá jednotný interface, který zobrazuje stav, zdraví a informace o výkonu počítačových systémů. Dále poskytuje možnost hlášení generovaných na základě dostupnosti, výkonu, konfigurace, nebo bezpečnostní situace. Pracuje s Microsoft Windows Server, nebo hostitelích na bázi Unixu [54].

Základní myšlenkou tohoto řešení je umístit část software, agenta, na počítač, který má být monitorován. Agent sleduje určité zdroje na tomto stroji včetně Windows Event Logu, pro určité události, nebo hlášení generované aplikacemi, které běží na sledovaném stroji. Při výskytu a detekci hlášení agent přepošle hlášení centrálnímu SCOM serveru. Tento SCOM server udržuje databázi, která obsahuje historii hlášení. SCOM server dále provede filtrování hlášení, které přicházejí. Pravidlo může odvolat notifikaci lidskému uživateli, jako například e-mail, nebo jiný druh zprávy, generovat ticket po síti, nebo spustit jiný proces, který má za účel napravit příčinu problému v co nejkratším časovém horizontu [54].

SCOM používá pojem „management pack“, kterým se odkazuje na sadu pravidel, které jsou specifické pro monitorovanou aplikaci. Zatímco Microsoft a jiní dodavatele software dodávají tyto balíčky pro jejich produkty, SCOM poskytuje také možnost uživatelského nastavení balíčků. Zatímco administrátorská role je potřeba pro instalaci agentů, konfiguraci monitorovaných strojů a tvorbu balíčku pro správu, práva pro zobrazení hlášení mohou být přiřazeny libovolnému uživatelskému účtu [54].

Více SCOM serverů může být agregováno dohromady, aby monitorovaly více sítí skrze logické Windows doménové a fyzické sítě. V předchozí verzi Operations Manageru, byla webová služba určena k tomu, aby spojila více odděleně spravovaných skupin do centrální lokace. Od verze 2007 webová služba není nadále používána. Namísto toho je využíváno přímé TCP spojení, které pro tuto komunikaci používá port 5723 [54].

Operations Manager 2007 obsahuje rozšiřitelné rozhraní v příkazové řádce nazvané „Command Shell“, které je upravená instance Windows PowerShell, které poskytuje interaktivní a na skriptech založený přístup k datům a operacím SCOM. Stejně tak jako Windows PowerShell se jedná o objektově orientovaný model programování a používá verzi 2.0 Microsoft .NET Framework. Obsahuje také sadu příkazů a funkcionality dostupnou v PowerShell, která poskytuje administrátorům možnost automatizovat jejich práci s Operations Managerem [54].

SCOM může být rozšířen za pomoci importu „management packů“ (MP), které definují to, jak SCOM monitoruje systémy. Ve výchozím stavu SCOM monitoruje pouze základní služby, které se vztahují k operačnímu systému, ale nové MP mohou být importovány a obsahovat kontrolu například SQL serverů, SharePointu, Apache, Tomcat, VMware a SUSE Linuxu [54].

7.1.1 Výhody

- Centralizovaný reporting hlášení, toto je výhodné pro enterprise řešení, které mohou využívat celé týmy.
- Management Packy mohou být instalovány pro ostatní produkty, jako například SQL Server a také SW třetích stran.
- Je možné nastavit práh pro hlášení, takže aplikace zašle varování předtím, než odešle kritickou hlášku, je zde tedy čas na řešení.
- Hlášení mohou být zasílány skrze e-mail, nebo přes SMS.

- SCOM ve spojení s SA Vision Live Maps usnadňuje tvorbu vizuálního „dashboardu“ z uživatelského prostředí. Je možné vytvářet hierarchické mapy, reprezentovat celé prostředí na geografickém měřítku.
- Pokud má uživatel aplikaci, která zasílá zprávy do event logu, je snadné vytvořit monitory a pravidla pro specifickou chybu, která uživatele zajímá a zaslat chybu jako e-mail
- SCOM je zároveň jak agent tak může běžet i bez agenta, takže je zde možnost získat lepší možnosti monitoringu instalací agenta.

7.1.2 Nevýhody

- Zatěžování výkonu systému
- Složitá implementace síťového monitoringu
- Dlouhá doba, než Microsoft vyřeší servisní hlášení
- Není možné importovat vlastní definice kontrol

7.1.3 Cena

- Typ licence: 2 letý pronájem licencovaný na jádro
- Cena: **3.607 USD / 2 roky** [55]

7.2 SolarWinds NPM

Monitorovací systém pro kontrolu výkonu sítě (NPM) poskytuje kontrolu chyb sítě a správu výkonu, které je možné škálovat a rozšiřovat dle potřeb monitoringu. Uživatelé mohou vidět dostupnost sítě v reálném čase a procházet statistiky a historii provozu na routerech, switchích, firewallech, serverech a jiných zařízeních za pomoci přístupu přes webové rozhraní [56].

Uživatelé SolarWinds mohou získat plnou instalaci ve stejný den, kdy si software stáhnou. Implementace nepotřebuje tým konzultantů, jak tomu často bývá u podobných produktů. Systém je možné nasadit a zprovoznit během jedné hodiny, po níž NPM poskytne rychlý náhled na „zdraví“ síťových zařízení a serverů na síti, díky čemuž zajistí, že uživatelé jsou schopni monitorovat vše v reálném čase [56].

7.2.1 Výhody

- Vysoká flexibilita systému
- Práce se šablonami
- Možnost snadného nastavení chybových hlášení
- Reportovací systém je součástí

7.2.2 Nevýhody

- Pouze kontrola síťového provozu
- Díky obrovské flexibilitě nastává problém s množstvím možností pro nastavení
- Náročné použití pro nezkušené uživatele

7.2.3 Cena

- Typ licence: 30-denní testovací licence zdarma, poté je nutné licenci zakoupit.
- Cena: **2.795 USD** [57].

7.3 AppDynamics

AppDynamics je aplikace, které poskytuje přístup v reálném čase ke všem částem provozu organizace, díky čemuž je schopna některé problémy s předstihem předvídat a automaticky je vyřešit ještě předtím, než nastanou. Aplikační zpravodajství poskytuje uživatelům náhled na informace o výkonu aplikací, způsobu práce uživatelů a také obchodní stránce jejich softwarových aplikací [58].

7.3.1 Výhody

- Kontrola běhu kódu a odhalení výkonových problémů
- Cílené výsledky pro všechny možné problémy
- Detailní informace o průběhu kódu
- Jednoduchá vizualizace

7.3.2 Nevýhody

- Složitě prvotní nastavení
- Nestabilita systému

- Nedostatečné možnosti reportingu
- Nepřehledná vyhledávací funkce

7.3.3 Cena

Typ licence: 15-denní testovací verze, poté je třeba zakoupit licenci.

Cena: **300 USD** [59]

7.4 DynaTrace APM

Řešení pro správu výkonu aplikací, jejich monitoring a řízení. Hlavním cílem je poskytnout uživateli co nejlepší možnosti pro kontrolu běhu aplikací. Proto jsou uživatelské interakce a obchodní transakce monitorovány a analyzovány až na úroveň kódu aplikace tak, aby systém mohl poskytnout co nejlepší náhled na běh aplikace. Moderní systém pro kontrolu běhu aplikací, který dokáže odhalit případné problémy dříve, než mohou nastat [60].

7.4.1 Výhody

- Šetří čas díky tomu, že dokáže odhalit pravděpodobnou příčinu problému
- Rychlá kontrola běhu bez vlivu na výkon systému
- Kvalitně provedený dashboard
- Interaktivní rozhraní, které zobrazuje vizuální reporty pro návratnost investice

7.4.2 Nevýhody

- Velké množství možností je zobrazeno nepřehledně
- Neintuitivní ovládání
- Množství funkcí, které jsou u konkurence standardem, není obsaženo a je nutné za ně připlatit

7.4.3 Cena

- Typ licence: On-Premise, nebo SaaS
- Cena: **10.000 USD / rok** [60]

7.5 ScienceLogic

Systém ScienceLogic dovoluje definovat, jaká data se shromažďují, jak syntetizovat a vyhodnocovat jakoukoli kombinaci dat pro generování inteligentních událostí, jaké akce spouštějí, když se tyto události vyskytnou, a jak prezentovat data a události v dashboardu a sestavách specifických pro danou roli [61].

- Automaticky sbírá a sleduje konfiguraci, aktiva (licence, sériová čísla atd.), Stav a metriky výkonu prvků a služeb
- Shromažďujte data z libovolného prvku nebo služby pomocí různých technik sběru dat
- Používá kombinaci synchronních a asynchronních metod sběru
- Automaticky vybere nejlepší metody a metriky pro shromažďování běžných klíčových slov KPI, jako je procesor, paměť, latence sítě a kapacita úložiště, a to pomocí jakéhokoli podporovaného protokolu nebo metody sběru
- Automaticky odvodí metriky ze shromážděných dat, aby zajistil konzistentní smysl mezi dodavateli a produkty. Například, jelikož jiní dodavatelé shromažďují a ukládají metriku jinak, standardní zásady normalizují tyto nesrovnalosti ve vašem zájmu; Pravidla pro normalizaci typu "point-and-click" mohou také využít podporu politik pro nové dodavatele a produkty.
- Snadno nastaví zásady sběru dat mimo dosah, které splňují specifické potřeby, například zapnutí a vypnutí kolekce pro určité metriky a přidejte další vlastní nebo odvozené metriky [61]

7.5.1 Výhody

- Webově založená aplikace, není třeba instalace klienta
- Používá SNMP pro komunikaci se systémy a sběr dat
- Rychlý a robustní

7.5.2 Nevýhody

- Chyby v SSL komunikaci
- V případě ukládání snapshotů vytěžuje velmi procesor počítače.

7.5.3 Cena

Typ licence: Jednorázová, bez časového omezení

Cena: **25.000 USD** [61]

7.6 Vyhodnocení průzkumu

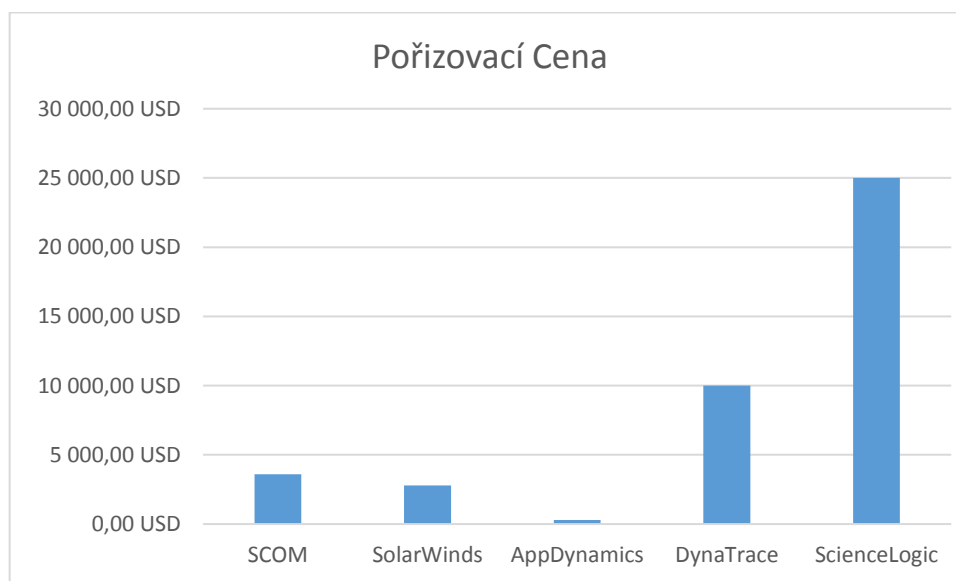
Z průzkumu je patrné, že existuje velké množství řešení, které je schopné pokrýt požadovanou problematiku, nicméně každý ze systémů je určitým způsobem specifický a zaměřuje se více či méně na určitou specializaci co se monitoringu týče.

Jak po stránce funkční, tak po stránce cenové jsou obrovské rozdíly napříč monitorovacími systémy. Pokud bychom měli volit po stránce funkční, nejlépe vypadaly aplikace **SCOM** od firmy Microsoft a **Science Logic** od stejnojmenné firmy, které poskytovaly dostatečné množství funkcionality.

Nejnižší pořizovací cenu má aplikace **AppDynamics**, která je řádově levnější, než konkurence (Tab. 1, Obr. 3).

Pořadové číslo	Název	Typ licence	Cena
1	SCOM	2 letý pronájem	3 607,00 USD
2	SolarWinds	Časově neomezená	2 795,00 USD
3	AppDynamics	Časově neomezená	300,00 USD
4	DynaTrace	1 roční pronájem	10 000,00 USD
5	ScienceLogic	Časově neomezená	25 000,00 USD

Tab. 1. Porovnání cen a licencování monitorovacích systémů



Obr. 3. Porovnání cen monitorovacích systémů na trhu

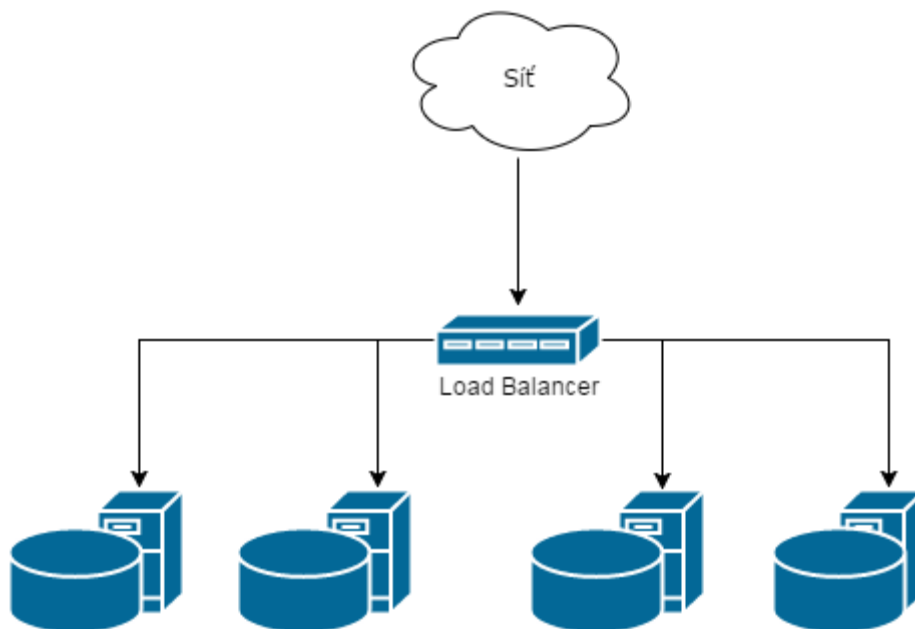
II. PRAKTICKÁ ČÁST

8 MOTIVACE PRO VZDÁLENÉ MONITOROVÁNÍ A SPRÁVU SKUPINY ZAŘÍZENÍ

V moderní době v oboru IT stále vzrůstají nároky na spolehlivost a dostupnost informačního systému. Mít možnost pracovat s informačním systémem kontinuálně a bez výpadků se považuje za naprostou samozřejmost, nicméně nikdo není schopen na 100% zaručit perfektní provoz, protože ten je ovlivněn celou řadou faktorů jako například:

- dostupnost sítě
- dodávka elektrické energie
- spolehlivost hardware
- funkcionality software
- živelné katastrofy
- lidský faktor

Samozřejmě existují řešení, která zvyšují spolehlivost systému a to buď formou vybudování vhodného zabezpečení v rámci infrastruktury, SW clusteringu, SW, nebo HW load-balancingu (Obr. 4), zálohování dat, zdvojení, či ztrojení elektrických sítí, či internetových linek.



Obr. 4. Příklad Load Balancingu ve schematicém zobrazení

Každá z uvedených metod vyžaduje určitý náklad na vybudování a taky na provoz a ne vždy je majitel informačního systému ochoten, nebo schopen tyto náklady pokrýt, přestože si uvědomuje rizika daná možným selháním systému. Toto je třeba vždy velmi pečlivě promyslet

a na základě toho, o jak důležitou součást systému se jedná, je třeba provést i odpovídající zálohu.

V závislosti na systému, je nutné provést i členění potenciálních závad, chyb, nebo selhání, která mohou nastat. Většina publikací uvádí 4 hlavní úrovně „závažnosti“ (anglicky „severity“):

1. **Critical (Mission / Business Critical):** Hlavní systém, nebo jiný životně důležitý systém (systémy) je nedostupný a není možnost, jak problém ihned obejít. V případě Mission critical se může jednat například o selhání navigačního systému letadla. Jedná se tedy o situaci, která přímo ohrožuje úspěch mise, je zde velmi reálná možnost ztráty na životech, vážného zranění, nebo velké finanční škody. Ve světě ERP systémů, kterým se tato práce zabývá, se jedná v drtivé většině případů o problémy typu Business critical, tedy o ztrátu finanční.
2. **Major:** Hlavní funkcionality systému je vážně narušena. Systém může dočasně pracovat v omezeném režimu, avšak dlouhodobá produktivita může být nepříznivě narušena. Pokud by problém nebyl odstraněn, dosažení hlavního cíle je ohroženo.
3. **Minor:** Částečně omezená funkcionality systému. Některé komponenty jsou ovlivněny, ale systém z valné části pracuje dál. Pokud nedojde k odstranění problému, ani z dlouhodobého hlediska nehrozí ovlivnění produktivity.
4. **Cosmetic:** Problémy, či nedostatky kosmetického rázu. Nedochozí k žádnému ovlivnění funkcionality, systém pracuje dále. Může se jednat o drobné nedostatky v UI aplikace, případně chyby v dokumentaci [65].

Toto jsou ve zkratce hlavní skupiny problémů, které mohou v systému nastat, pro účel této práce budeme pracovat s prvními třemi skupinami.

Firma XY, nabízí ERP, pokladní a skladový systém pro maloobchodní zákazníky. Jedná se o několik desítek zákazníků z velké části v ČR a SK, určitá část je zahraniční. Zákazníci provozují maloobchodní prodejny s rozsahem mezi jednotkami až do stovek prodejen. Spojujícím faktorem těchto zákazníků je využití stejného ERP systému.

Vzhledem k tomu, že firma XY poskytuje i podporu a v některých případech i infrastrukturu pro provoz systému, vyvstal požadavek na možnost centrálního monitoringu a řízení těchto systémů tak, aby bylo možno kontrolovat prostředky s následujícími požadavky:

- Kontrola místa na disku včetně prahových hodnot
- Kontrola vytížení procesoru včetně prahových hodnot

- Kontrola vytížení paměti RAM včetně prahových hodnot
- Kontrola stavu předem definovaných služeb včetně možnosti je vzdáleně řídit
- Kontrola databáze ERP aplikace na základě předdefinovaných skriptů
- Vše formou pluginů do aplikace s možností rozšíření

Jak se ukázalo, existuje obrovské množství již hotových řešení, které nabízejí různou úroveň funkcionality a samozřejmě mají různou pořizovací cenu.

9 ANALÝZA POŽADAVKŮ NA MONITOROVACÍ SYSTÉM

Pro definici jasného rámce požadavků byl stanoven tým zástupců sestávající z členů následujících oddělení:

- technická podpora (Technical Support)
- vývoj (Development)
- kvalita (Quality Assurance)
- business analýza (Business Analyst)
- obchod (Sales)

Každý člen týmu přidal část požadavků v závislosti na svých pracovních zkušenostech tak, aby výsledný produkt co nejvíce napomohl k usnadnění jejich práce a pokud možno pokryl již známé, často se opakující problémy a situace.

Docházelo k pravidelným schůzkám a poradám, které vnášely na řešení velmi cenný nadhled skrze

9.1 Funkční požadavky

Funkční požadavky na SW definují funkci systému, nebo jeho komponent. Funkce je popsána jako sada vstupů, chování a výstupů.

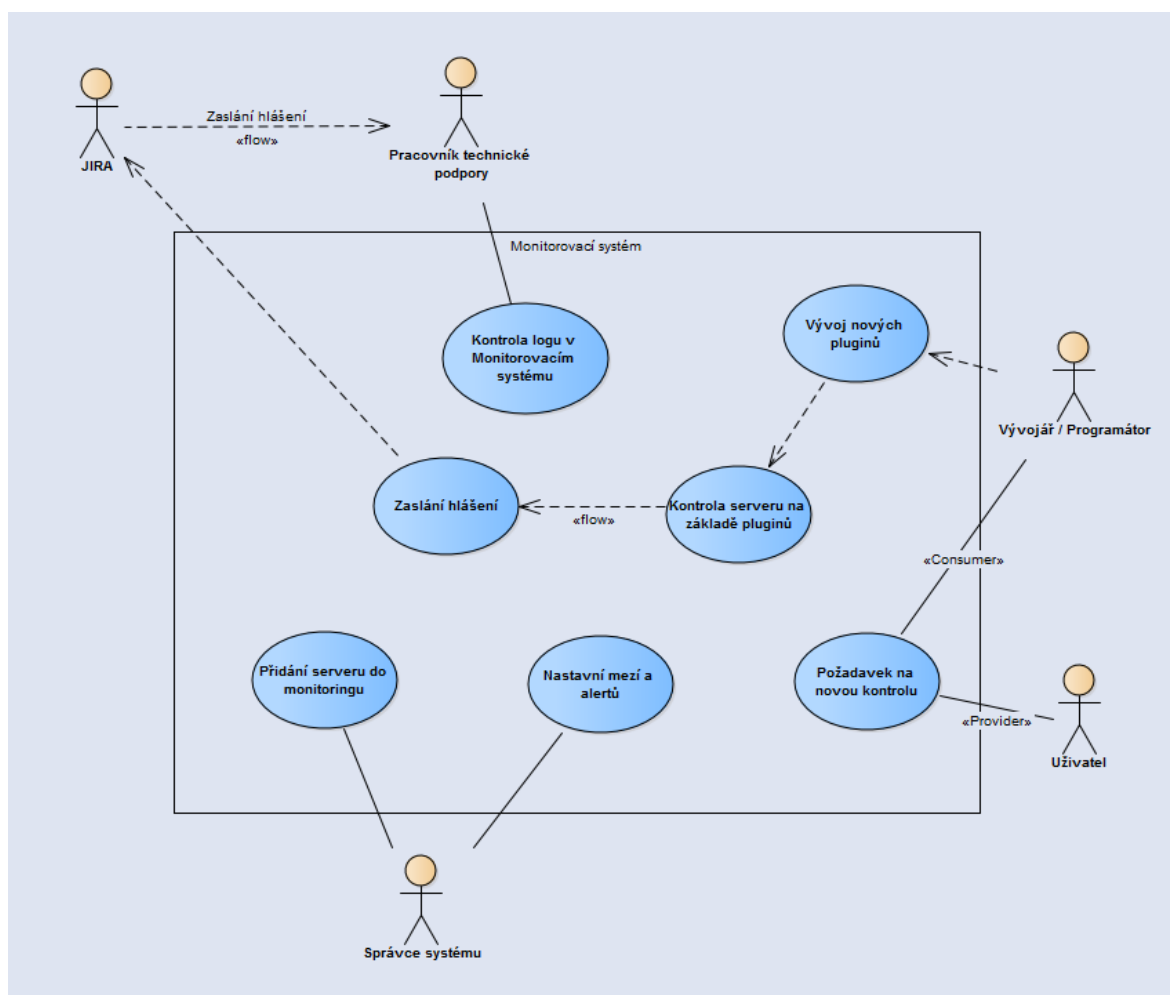
Funkční požadavky mohou být výpočtu, technické detaily, práce s daty a jejich zpracování a jiná specifická funkcionalita, která definuje, co by měl systém dokázat. Požadavky na chování, které popisují všechny případy, kdy systém používá funkční požadavky, je zaznamenán v tzv. „Případu užití“ (anglicky Use case). Funkční požadavky jsou podpořeny nefunkčními požadavky, které přinášejí omezení ze strany designu, nebo implementace. Obecně vzato funkční požadavky specifikují výsledky systému. Toto je dáno do kontrastu s nefunkčními požadavky, které specifikují celkovou charakteristiku, jakou jsou náklady a spolehlivost. Funkční požadavky řídí aplikační architekturu systému, zatímco nefunkční požadavky řídí technickou architekturu systému.

V některých případech analytik požadavků vytvoří případy užití poté, co zjistí a ověří sadu funkčních požadavků. Hierarchie funkčních požadavků je: Požadavek uživatele/zúčastněné strany -> vlastnost -> případ užití -> business pravidlo. Každý případ užití popisuje scénář chování skrze jeden, nebo více funkčních požadavků.

Byly vydefinovány tyto základní požadavky (Obr. 5):

- Standardní kontrola služeb (služba ERP systému a služba IIS)
- Informace z performance counterů
- Indexace SQL databáze
- Kontrola Task Scheduleru
- Kontrola průběhu SQL Jobů
- Nastavení mezí a alertů
- Kontrola dostupnosti SQL serveru
- Kontrola korektního nastavení databáze pro běh ERP systému

Na základě vydefinovaných požadavků byl vytvořen diagram use case pro fungování aplikace:



Obr. 5. Diagram Use Case pro monitorovací systém

Představa je taková, že bude vyvinuta základních pluginů, která bude provádět základní kontroly běhu systému. Systém bude nastaven tak, aby mohl reportovat hlášení do systému

JIRA, který firma XY používá pro evidenci hlášení. Hlášení bude předáno na pracovníka technické podpory, který bude proškolen pro práci s monitorovacím systémem a provede kontrolu logu. Na základě poskytnutých informací provede odstranění problému. O nastavení mezi hlášení a typů hlášení se bude starat osoba zodpovědná za daný server, v našem případě jsme jej nazvali „Správce systému“. Toto nastavení může být individuální vždy dle daného serveru. Pokud bude třeba z jakýchkoliv důvodů rozšířit možnosti monitoringu, primární impulz je očekáván od osoby, která je ve schématu nazvána jako „Uživatel“. Toto může být konzultant, který má na starosti daného zákazníka, může to být zákazník sám, případně se může jednat o vývojáře, pracovníka oddělení QA, nebo jiný zdroj, důležitý je jeho zájem o správné fungování prostředí a každé jeho hlášení by mělo být schváleno představenstvem, které rozhodne o tom, zda daný požadavek dává smysl.

9.2 Nefunkční požadavky

Při definici požadavků jsou vedle funkčních požadavků velmi důležité tzv. „nefunkční požadavky“. Jedná se o specifický druh požadavku na systém, který určuje takové podmínky, které mohou zhodnotit samotný provoz systému bez ohledu na jeho vlastní funkcionalitu, nebo samotné chování. Tyto požadavky jsou vypracovány v tzv. „systémovém návrhu“. Plán pro implementaci nefunkčních požadavků je do detailu rozepsán v systémové architektuře, protože se obvykle jedná o požadavky významné pro architekturu.

Obecně řečeno, funkční požadavky definují, co by měl systém dělat, kdežto nefunkční požadavky definují, jak by to měl systém dělat. Funkční požadavky jsou často definovány ve formě „systém by měl dělat <požadavek>“, jedná se o individuální akci části systému ve smyslu matematické funkce, popisu vstupu, výstupu, procesu a funkčního modelu. Oproti tomu nefunkční požadavky jsou ve formě „systém by měl být <požadavek>“, ve smyslu celkové vlastnosti systému jako celku, nebo jako jeho aspekt, nikoliv jako specifická funkce.

Nefunkční požadavky jsou často nazvány „atributy kvality“ systému. Jiným názvem se tyto požadavky dají nazvat jako „vlastnosti“, „cíle vlastností“, „vlastnosti servisních požadavků“, „omezení“, „ne-behaviorální požadavky“. Neformálně jsou často nazývány anglickým slovem „ilities“, na základě atributů jako je stabilita a přenositelnost. Vlastnosti, které patří mezi nefunkční požadavky, mohou být rozděleny do dvou hlavních kategorií:

1. Vlastnosti provedení, jako je například zabezpečení a použitelnost, které jsou pozorovatelné v době běhu

2. Vývojové vlastnosti, jako je například testovatelnost, udržitelnost, rozšiřitelnost a škálovatelnost, které jsou vytvořeny ve statické struktuře softwarového systému.

9.2.1 Seznam nefunkčních požadavků

Zde je uveden seznam výsledných nefunkčních požadavků. Jsou seřazeny v pořadí dle přiřazené důležitosti:

- ***Dostupnost***

Jako nejdůležitější požadavek pro systém monitoringu byla zvolena dostupnost. Pod tímto požadavkem si představujeme stabilní komunikaci klientské se serverovou částí i v případě nepříznivých podmínek daných například vytížením serveru, nebo linky. Dostupnost bude zajištěna metodou Active / Passive, kdy systém poběží pouze na jednom stroji a bude připraven v záloze k převzetí aktivity v okamžiku, kdy na druhém serveru nastane výpadek.

- ***Modifikovatelnost***

Požadavek na modifikovatelnost je založený na představě možnosti upravovat kód systému, případně za pomoci pluginů měnit jeho běžné chování. Jako sekundární výstup tohoto požadavku je požadavek na open source kód, který by vývojové oddělení mohlo modifikovat na základě měnících se požadavků.

- ***Stabilita***

Stabilita systému je klíčová pro kontinuální monitorování a správu systémů. Pokud by docházelo k častým výpadkům, nebo pádům software, nejen, že by neplnil svůj úkol, ale přidal by další starosti týmu zodpovídající za SW podporu.

- ***Rozšiřitelnost***

Základní požadavek pro potřeby flexibilního používání SW je možnost jeho rozšiřitelnosti. Předpokládá se, že po nasazení monitorovacího systému dojde k rozšiřování jeho základní funkcionality v závislosti na potřebách interních i na potřebách konkrétního zákazníka. Každá implementace systému musí být schopna dynamicky obsloužit různé funkční požadavky.

- ***Omezení zdrojů***

Servery, na kterých je plánováno nasazení monitorovacího systému, jsou velmi vytíženy samotným provozem ERP systému. Není možné očekávat, že budou přidány nové prostředky jen pro provoz monitorovacího systému, proto je velmi kritické, aby byla aplikace schopna fungovat za co nejnižšího využití systémových prostředků a nezatěžovala procesor, operační paměť, disky, nebo síťový provoz.

- ***Nasazení (Deployment)***

Snadné nasazení systému na klientské stroje je dalším z klíčových požadavků. Momentálně se jedná o desítky až stovky serverů a proto je pohodlná implementace důležitým faktorem pro rozhodování.

- ***Čas odezvy***

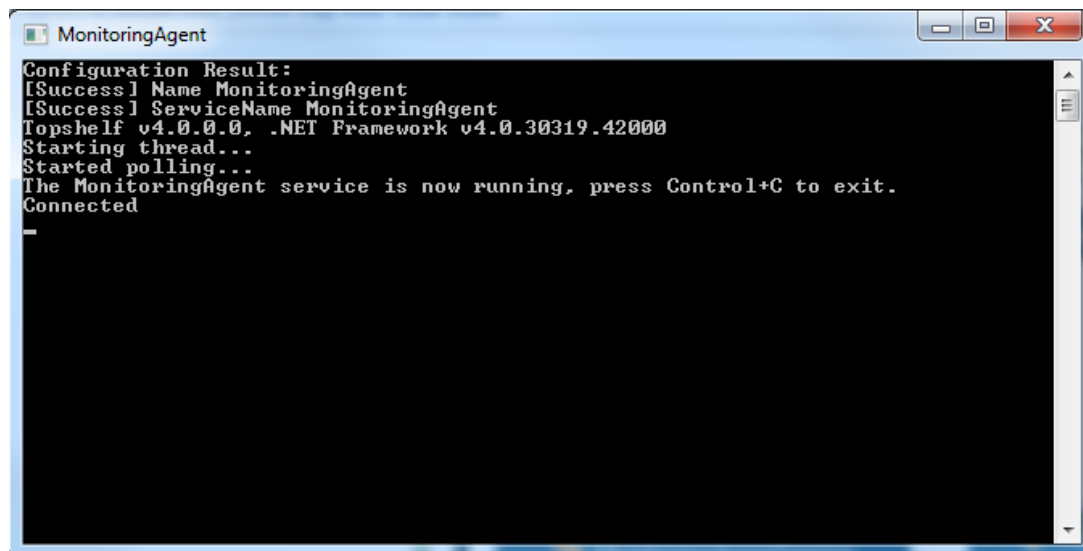
V rámci funkčních požadavků bylo definováno, že se očekává, že systém bude schopen vzdáleně kontrolovat a ovládat běh služeb, které jsou kritické pro ERP systém. Pro tento požadavek je důležité, aby odezva systému byla v rámci rozumných mezí. Není možné akceptovat systém, jehož reakční doba znemožní kvalitní kontrolu zdrojů a ovládání běhu služeb.

9.3 Cena

Velmi důležitým, často tím nejdůležitějším, faktorem při rozhodování o pořízení nového systému je samozřejmě cena. Technicky se jedná o nefunkční požadavek, nicméně mu byla přiložena taková důležitost, že jsem se rozhodl věnovat mu samostatnou podkapitolu. V rámci projektu bylo diskutováno, jaké jsou očekávání a možnosti na funkční a nefunkční parametry systému, nicméně se ukázalo, že cena je rozhodujícím faktorem. Bohužel není možné, aby firma XY byla schopna hradit licenční poplatky za SW, který bude monitorovat běh jejich ERP systémů napříč všemi zákazníky. Na druhou stranu není reálné očekávat, že zákazníci sami budou ochotni hradit poplatky navíc, ať už jednorázové, nebo na měsíční bázi za to, že bude prováděna kontrola běhu jejich systému. Výsledkem diskuze je tedy požadavek na průzkum aktuálně nabízených systémů s nulovou pořizovací cenou, případně vývoj vlastního SW.

10 APLIKACE MONITORING AGENT

Aplikace Monitoring agent je koncipována jako monitorovací a reportovací aplikace (Obr. 6) běžící na pozadí systému Windows Server / Windows (desktop) jako služba, která v pravidelných intervalech zasílá data serveru. Tato aplikace je vyvinuta za použití vývojového prostředí Microsoft Visual Studio 2015 s pozdější migrací do prostředí Microsoft Visual Studio 2017.

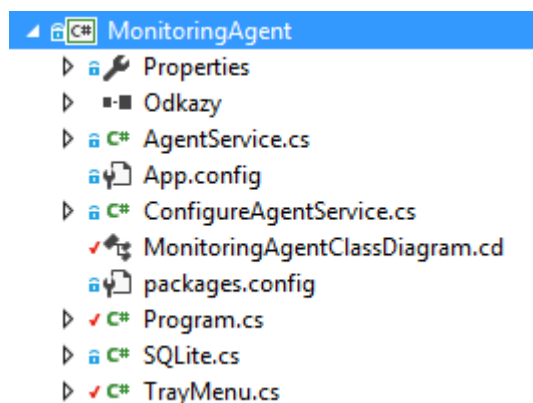


Obr. 6. Ukázka běhu konzolové aplikace MonitoringAgent

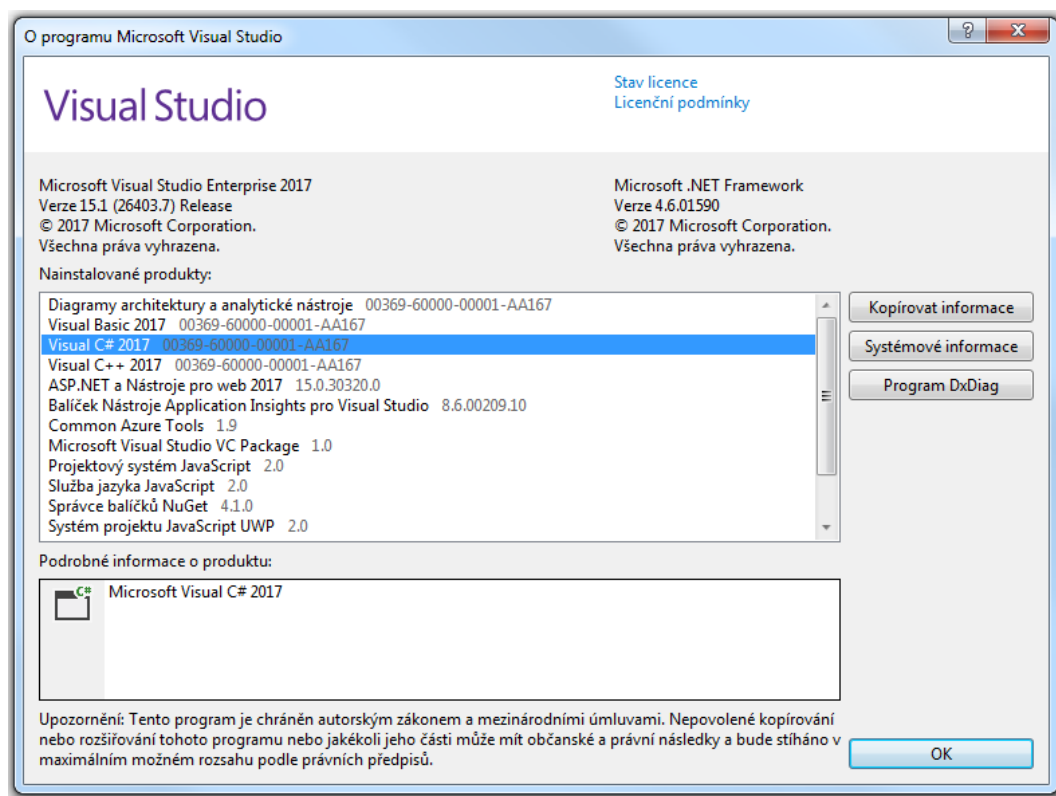
Programovacím jazykem je Visual C# verze 2017 (Obr. 8) za využití Microsoft .NET Framework 4.6. z roku 2017. Kritériem pro volbu tohoto programovacího jazyka byla jak jeho kvalitní a dostupná dokumentace, tak i to, že je hlavním programovacím jazykem pro vývoj software ve firmě XY.

Řešení v rámci této práce obsahuje klienta, jež se napojuje na server, který je zadáním jiné diplomové práce.

Visual Studio a jeho menu (Obr. 7) dovoluje velmi pohodlnou práci s projekty.



Obr. 7. Zobrazení stromu projektu v IDE
VS



Obr. 8. Ukázka verze použitého jazyka C#

Aplikace běží se skrytým aplikačním oknem a v je zobrazena pomocí ikony v liště nástrojů (Obr. 9)

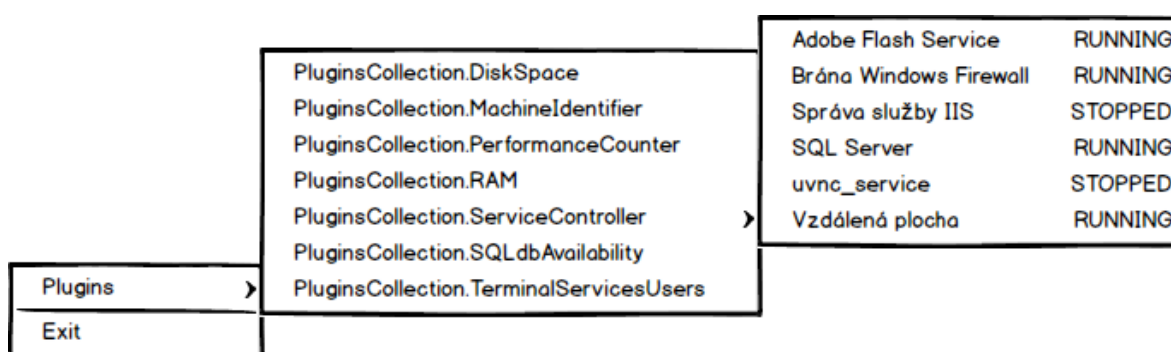


Obr. 9. Ukázka aplikace běžící v liště

Po kliknutí pravým tlačítkem se zobrazí menu (Obr. 10), které má volbu ukončení aplikace a zobrazení menu se seznamem pluginů. Jako plánované rozšíření aplikace je dynamické menu se seznamem všech pluginů (Obr. 11), které jsou aktuálně načteny včetně aktuálně změřených hodnot:



Obr. 10. Menu aplikace



Obr. 11. Vizualizace kompletního menu včetně výstupů z testů

Je třeba zdůraznit, že aplikace je klientem a její hlavní činností je běh s co nejnižší zátěží na stroji a reporting dat na server, zobrazovací funkce je čistě nadstavbová a slouží pro ad hoc přístup v případě, že je uživatel připojený na daném stroji a chce z jednoho místa kontrolovat a využití systémových prostředků a dalších atributů.

Menu aplikace je implementováno ve třídě „**TrayMenu**“ a běží na samostatném vlákne, což aplikaci usnadňuje běh a nezatěžuje vykonávání jejich procesů.

Ukázka kódu:

```
public static void Notify()
{
    Thread notifyThread = new Thread(
        delegate ()
        {
            SetMenu();

            notificationIcon = new NotifyIcon()
            {
                Icon = new Icon("MonitoringAgent.ico"), //Properties.Re-
sources.Services,
                ContextMenu = menu,
                Text = "MonitoringAgent",
            };
        }
    );
    notifyThread.Start();
}
```

```
        notificationIcon.Visible = true;
        Application.Run();
    }
};
```

10.1 Běh programu na pozadí

Pro běh programu na pozadí (okno aplikace je skryté) je vytvořen níže uvedený kontroler, který za pomoci pomoci volání uvedených metod skryje, nebo zobrazí konzolové okno:

- **ShowWindow(handle, SW_HIDE)** = skryje okno aplikace
- **ShowWindow(handle, SW_SHOW)** = zobrazí okno aplikace

```
using System.Runtime.InteropServices;
```

```
/// <summary>
/// Get Console Windows controls
/// </summary>
/// <returns></returns>
[DllImport("kernel32.dll")]
static extern IntPtr GetConsoleWindow();

/// <summary>
/// Windows visibility controls
/// </summary>
/// <param name="hWnd"></param>
/// <param name="nCmdShow"></param>
/// <returns></returns>
[DllImport("user32.dll")]
static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

/// <summary>
/// Methods to HIDE or SHOW application window
/// </summary>
const int SW_HIDE = 0;
const int SW_SHOW = 5;
```

10.2 Pluginy (zásuvné moduly)

Monitorovací agent pracuje na základě funkcionality, kterou obsahuje plugin, respektive kolekce pluginů. Systém si pluginy dynamicky načte (kontroluje příslušnou složku a načte a spustí pouze ty, které obsahují všechny náležitosti) a vykoná jejich funkcionalitu, která může být jakýmkoliv kódem dle pluginové šablony, jenž obsahuje název, unikátní identifikátor a jako návratovou hodnotu vrací výsledek testů.

Plugin nedokáže jako část software pracovat samostatně, ale pouze jako součást programu.

10.2.1 Šablona pluginu

Pluginy běží ve jmenném prostoru „**PluginsCollection**“, kdy každý plugin dědí ze třídy „**IPlugin**“ (Obr. 18), která obsahuje vlastnosti:

- **Name** = název pluginu
- **UID** = unikátní identifikátor pluginu, jeho využití je pro rozlišení jak různých pluginů, tak i jejich verzí
- **Type** = typ návratové hodnoty pluginu, prozatím jsou vytvořeny hodnoty typu tabulka (table), graf (graph) a neznámý (unknown)
- **Output** = návratová hodnota metody

```
/// <summary>
/// Implementation of plugin interface with Name, UID, Type and Output properties
/// </summary>
public interface IPlugin
{
    string Name { get; }
    Guid UID { get; }
    PluginType Type { get; }
    PluginOutputCollection Output();
}
```

Samotná šablona pluginu obsahuje konstruktor, který vytvoří výstupní kolekci pluginu, jež obsahuje výstupní hodnoty a odvolá metodu **PluginOutputCollection**, která odvolá požadovanou funkcionalitu a vrátí výsledek. Ten může být ve formě tabulky, grafu, nebo jiné, prozatím nespecifikované formě.

```
using System;

// Plugin Collection default namespace
namespace PluginsCollection
{
    /// <summary>
    /// RAM Plugin class used for getting information
    /// </summary>
    public class PluginName : IPlugin
    {
        PluginOutputCollection _pluginOutputs;

        /// <summary>
        /// Plugin unique identifier for system recognition
        /// </summary>
        public Guid UID
        {
            get
            {
                return new Guid("00000000-0000-0000-0000-000000000000");
            }
        }

        /// <summary>
```

```

    /// Plugin specific name
    /// </summary>
    public string Name
    {
        get
        {
            return "Plugin text name";
        }
    }

    /// <summary>
    /// Plugin type (Graph, Table, Unknown)
    /// </summary>
    public PluginType Type
    {
        get
        {
            return PluginType.Table;
        }
    }

    /// <summary>
    /// Plugin constructor
    /// </summary>
    public PluginName()
    {
        _pluginOutputs = new PluginOutputCollection();
        _pluginOutputs.PluginUID = UID;
        _pluginOutputs.PluginName = Name;
    }

    /// <summary>
    /// Plugin specific functionality
    /// </summary>
    /// <returns>
    /// Plugin specific output
    /// </returns>

    public PluginOutputCollection Output()
    {
        return _pluginOutputs;
    }
}

```

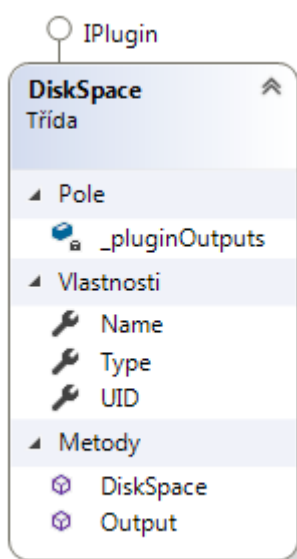
10.2.2 Seznam pluginů

Pro bližší informaci o prozatím vyvinutých pluginech slouží následující popis, ve kterém je uveden název, funkcionality a použité knihovny a techniky vývoje pro danou knihovnu. Tento seznam není konečný a je možné jej libovolně rozšiřovat, pokud bude dodržena šablona pluginu uvedena výše v této kapitole. Všechny uvedené pluginy používají návratový typ „table“, tedy tabulka, ale do budoucna je počítáno s rozšířením návratových typů dle požadavků, které v průběhu používání aplikace nastanou.

Třída **DiskSpace** (Obr. 12) pracuje s místem na disku a vrací tyto hodnoty, třída **MachineIdentifier** (Obr. 13) pracuje s unikátními identifikátory stroje a díky tomu má server jasný

přehled o tom, s jakými zařízeními pracuje. **PerformanceCounter** (Obr. 14) pracuje se systémovými událostmi a procesy, díky tomu dokáže kontrolovat vytížení stroje. **RAM** (Obr. 15) pracuje s hodnotami vytížení a dostupnosti paměti RAM. **SQLdbAvailability** (Obr. 16) slouží ke kontrole dostupnosti databáze. **TerminalServicesUsers** (Obr. 17) zobrazuje aktuálně spojené terminálové uživatele. Díky tomu je možné mít přehled o vytížení těmito uživateli.

- **DiskSpace**



Obr. 12. *DiskSpace*

Třída:

DiskSpace

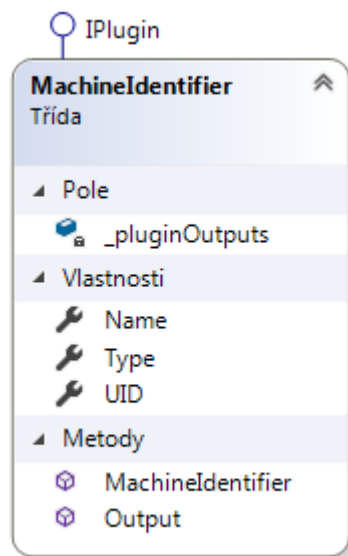
Použité jmenné prostory:

System = Základní jmenný prostor využívaný k použití základních datových typů

System.Collections.Generic = Jmenný prostor, který obsahuje definice kolekcí, v tomto případě je využíván pro typ `List<>`.

System.IO = Jmenný prostor, který je využitý k práci s informacemi o diskových jednotkách systému

- **MachineIdentifier**



Obr. 13. *MachineIdentifier*

Třída:

MachineIdentifier

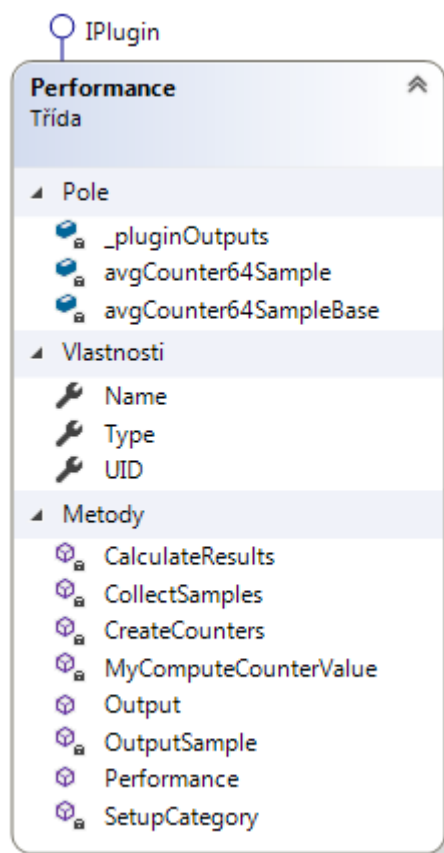
Použité jmenné prostory:

System = Základní jmenný prostor využívaný k použití základních datových typů

System.Collections.Generic = Jmenný prostor, který obsahuje definice kolekcí, v tomto případě je využíván pro typ `List<>`.

System.Management = Jmenný prostor, který poskytuje přístup k sadě pro správu událostí systému, zařízení a aplikacím registrovaným v infrastruktuře Windows Management Instrumentation (WMI).

- PerformanceCounter



Obr. 14. Třída Performance

Třída:

Performance

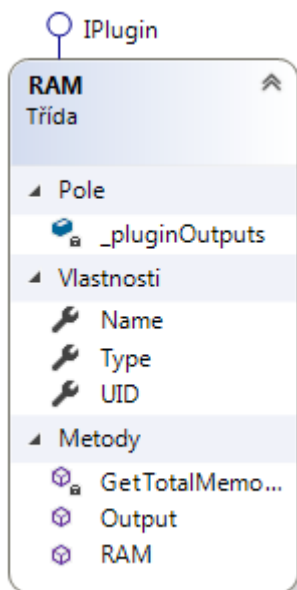
Použité jmenné prostory:

System = Základní jmenný prostor využívaný k použití základních datových typů

System.Collections.Generic = Jmenný prostor, který obsahuje definice kolekcí, v tomto případě je využíván pro typ List<>.

System.Diagnostics= Jmenný prostor, který obsahuje třídy, které podporují interakce se systémovými procesy, protokoly událostí a čítači výkonu. Podřízené obory názvů potom obsahují typy zajišťující interakci s nástroji pro analýzu kódu a další jako například podporu kontraktů.

- RAM



Obr. 15. Třída RAM

Třída:

RAM

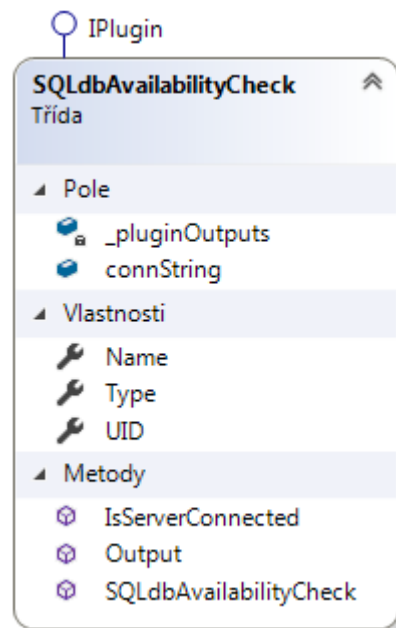
Použité jmenné prostory:

System = Základní jmenný prostor využívaný k použití základních datových typů

Microsoft.VisualBasic.Devices = Jmenný prostor, který obsahuje typy, které podporují „My“ objekty, které jsou dostupné v jazyce Visual Basic.

System.Collections.Generic = Jmenný prostor, který obsahuje definice kolekcí, v tomto případě je využíván pro typ List<>.

- **SQLdbAvailability**



Obr. 16. *SQLdbAvailability-Check*

Třída:

SQLdbAvailabilityCheck

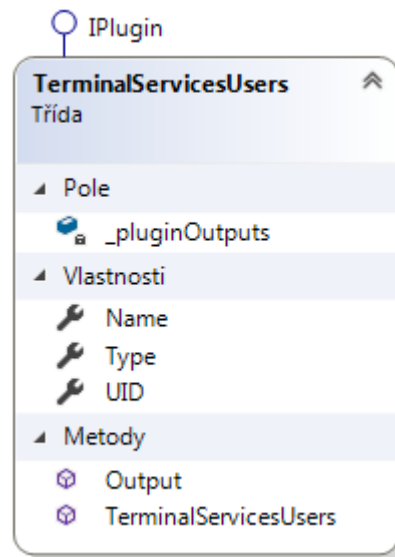
Použité jmenné prostory:

System = Základní jmenný prostor využívaný k použití základních datových typů

System.Data.SqlClient= Jmenný prostor, který definuje práci rozhraní .NET Framework Data Provider se SQL Serverem

System.Collections.Generic = Jmenný prostor, který obsahuje definice kolekcí, v tomto případě je využíván pro typ `List<>`.

- **TerminalServicesUsers**



Obr. 17. *TerminalService-
sUsers*

Třída:

TerminalServicesUsers

Použité jmenné prostory:

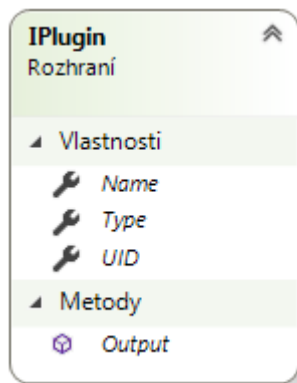
System = Základní jmenný prostor využívaný k použití základních datových typů

System.Security.Principal = Jmenný prostor, který definuje objekt zabezpečení, který představuje kontext zabezpečení, v němž je spuštěn kód.

System.Collections.Generic = Jmenný prostor, který obsahuje definice kolekcí, v tomto případě je využíván pro typ `List<>`.

Cassia = Knihovna pro správu připojených terminálových uživatelů.

- Rozhraní IPlugin



Obr. 18. IPlugin

10.3 Agent Service

Agent je navržen tak, aby byl schopen běžet jako služba na pozadí. Tato vlastnost zjednoduší jeho práci a bude možné zajistit to, že aplikace bude spuštěna ihned po spuštění serveru. Existuje více možností, jak spustit aplikaci jako službu, při návrhu aplikace bylo zvažovány tyto možnosti:

- Napsat aplikaci nativně jako **Windows Service**
- Napsat aplikaci jako konzolovou aplikaci a použít knihovnu **TopShelf**.

Nakonec kvůli lepší možnosti debuggování a instalaci služby byla zvolena druhá varianta, tedy využití knihovny **TopShelf**.

Implementace TopShelf v rámci projektu:

```
using Topshelf;

// Run application as a Windows service
return (int)HostFactory.Run(x =>
{
    x.Service<AgentService>(s =>
    {
        s.ConstructUsing(() => new AgentService());
        s.WhenStarted(service => service.Start());
        s.WhenStopped(service => service.Stop());
    });
});
```

Jak je z uvedeného kódu zřejmé, byl zde využit mechanismus „**HostFactory**“, který dovo-
luje tvorbu instancí hostitele služeb dynamicky na požádání. Toto je velmi užitečné v pří-
padě, že je třeba implementovat obslužné rutiny událostí pro spuštění a ukončování služby,

v případě použití nativní Windows Service aplikace by bylo jak celkové ladění v rámci IDE Visual Studio, tak i následná práce s aplikací složitější, což se projevilo již v brzkých fázích testování a vývoje.

10.4 Databáze SQLite

V případě, že není dostupné spojení na server, aplikace ukládá data určená pro odeslání na server do SQLite databáze, kde jsou uložena až do té doby, než je navázáno spojení. Tomuto procesu říkáme „cachování“, data jsou zde pouze dočasně a to do chvíle, než je opět navázána komunikace.

Díky tomu je možné, aby klientská aplikace fungovala po dobu výpadku spojení na server samostatně a data si uchovávala lokálně.

Pro přidání jmenného prostoru, který dovoluje práci s databází SQLite je použit tento způsob inicializace:

```
using System.Data.SQLite;
```

Pro práci s SQLite databází je použita třída SQLiteDB (Obr. 19), která obsahuje následující metody:

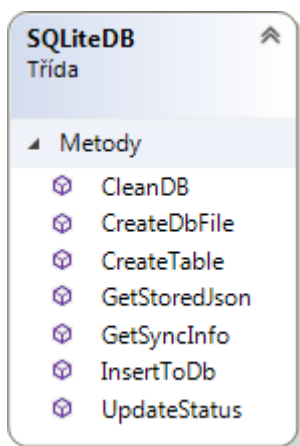
- **CleanDB** = Metoda pro vyčištění (promazání) databáze, v aktuálním řešení není implementována a bude řešena až jako další rozvojové práce.
- **CreateDbFile** = Metoda, která tvoří databázový soubor na základě podmínky, která ověří, jestli už databázový soubor neexistuje. Pokud není soubor nalezen, metoda jej vytvoří.
- **CreateTable** = Metoda, která tvoří databázovou tabulku v existujícím databázovém souboru. Používá velmi jednoduchý princip:

```
string sql = "CREATE TABLE IF NOT EXISTS [MonitoringAgentCache] ([Id] INTEGER  
PRIMARY KEY, [Datetime] DATETIME, [Result] TEXT, [Sync] BOOL)";
```

- **GetStoredJson** = Metoda, která vyčte z databáze data uložená ve formátu JSON a předá je jako návratovou hodnotu.
- **GetSyncInfo** = Metoda, která vrací informaci ohledně stavu synchronizace dat.
- **InsertToDb** = Metoda pro vložení dat do databázové tabulky. Vstupní parametry jsou název databázového souboru a výsledek, používá tento princip:

```
string insertQuery = String.Format("INSERT INTO MonitoringAgentCache (Datetime,  
Result, Sync) VALUES (DATETIME('NOW'), '{0}', '{1}'))", result, false);
```



- **UpdateStatus** = Metoda, která slouží pro úpravu stavu synchronizace na stav „Synchronizováno“. Má 2 vstupní parametry, kterými jsou název souboru a ID požadovaného řádku v tabulce.



Obr. 19. Metody ve třídě SQLiteDB

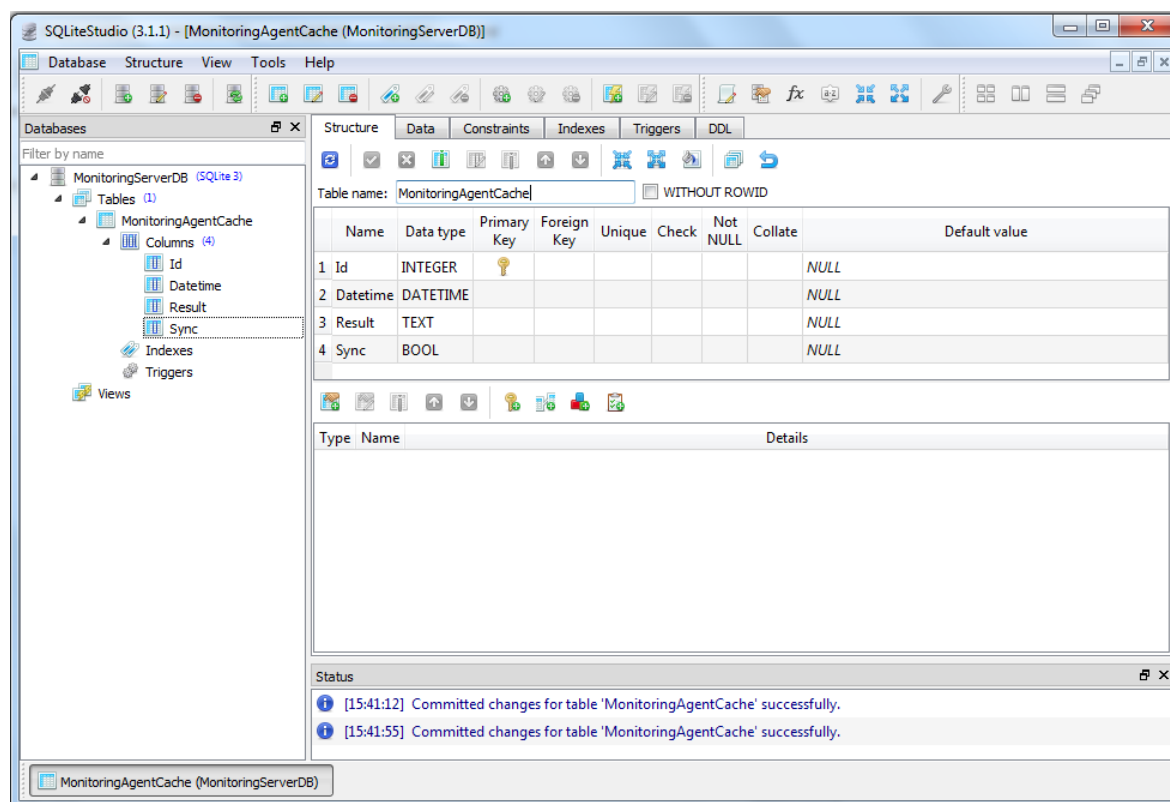
Pro tvorbu tabulky **MonitoringAgentCache** s možností cachování uložených dat (Obr. 20) volá aplikace následující SQL script:

```
CREATE TABLE MonitoringAgentCache (
    Id INTEGER PRIMARY KEY,
    Datetime DATETIME,
    Result TEXT,
    Sync BOOL
);
```

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	Id	INTEGER							NULL
2	Datetime	DATETIME							NULL
3	Result	TEXT							NULL
4	Sync	BOOL							NULL

Obr. 20. Tabulka MonitoringAgentCache v SQLite db

Pro editaci SQL scriptů byla využita aplikace SQLite Studio 3.1.1 (Obr. 21)



Obr. 21. Prostředí SQLiteStudio 3.1.1

10.5 Diagramy tříd

IDE Visual Studio 2017 poskytuje možnost pro generování diagramu tříd a zobrazení vazeb mezi třídami.

Diagram tříd je jedním typem z UML diagramů a zobrazuje statický pohled na strukturu systému.

Pro aplikaci MonitoringAgent je tento diagram zobrazen níže (Obr. 22).

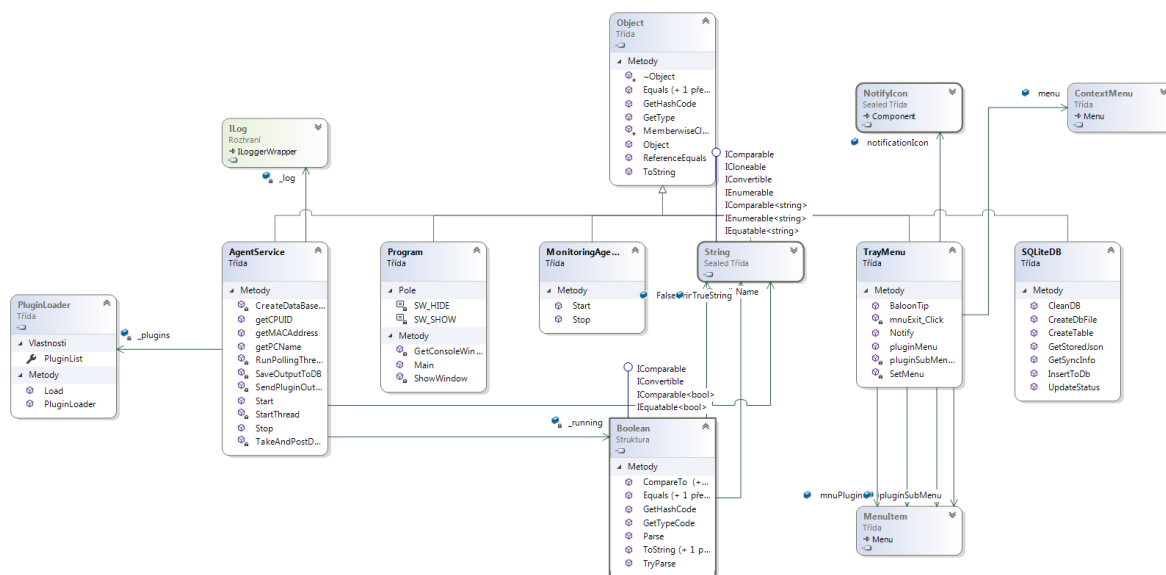
Celkové zjednodušené schéma bez vazeb (Obr. 23) dává jednoduchou představu o tom, a jakými třídami pracujeme.

Řešení obsahuje tyto třídy:

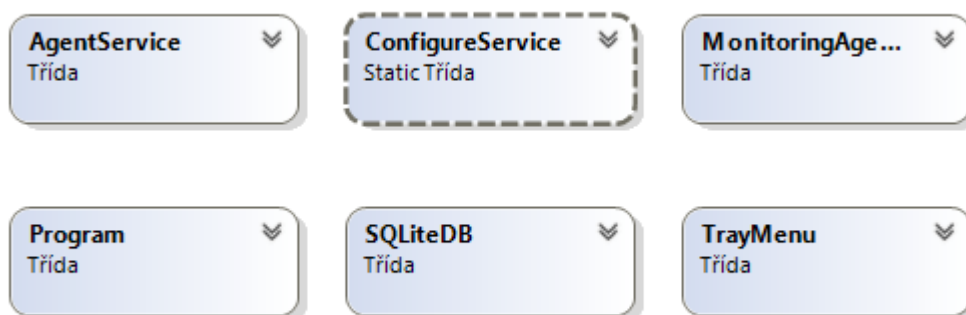
- **AgentService** = Hlavní třída, která obsluhuje většinu běhu programu
- **Program** = Třída, které provede inicializaci aplikace a poté předá řízení programu třídě AgentService
- **SQLiteDB** = Třída sloužící k práci s databází SQLite
- **TrayMenu** = Menu aplikace, které je obslouženo samostatným vláknem.

- **MonitoringAgent** = Třída, která spouští a ukončuje službu aplikace

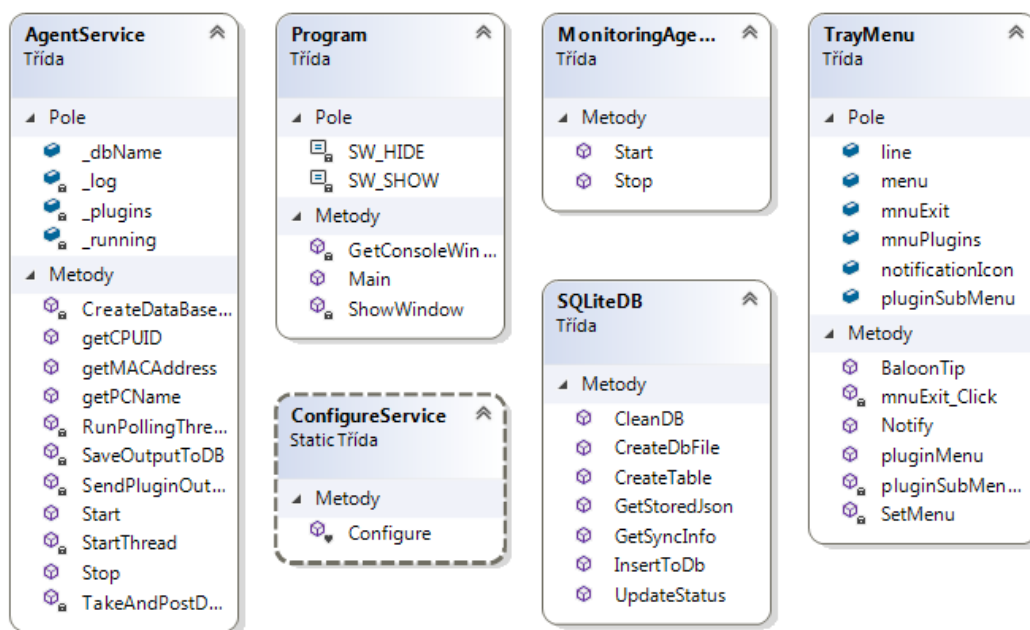
Schéma tříd může po „rozkliknutí“ zobrazovat také všechna využitá pole (Obr. 24).



Obr. 22. Diagram tříd včetně vzájemných vazeb



Obr. 23. Třídy aplikace ve zjednodušeném zobrazení



Obr. 24. Schéma tříd včetně metod a polí

10.6 Mapa kódu

Zde je zobrazena mapa kódu monitorovacího agenta vygenerovaná v IDE Visual Studio, které má možnost tuto mapu nejen generovat, ale také s ní dynamicky pracovat.

Na rozdíl od diagramu tříd se nejedná o statický, ale dynamický prvek, který nám pomáhá s laděním běhu aplikace a graficky znázorňuje samotný průběh.

Jsou zde uvedeny vazby, které existují v rámci běhu programu mezi jednotlivými jmennými prostory. V případě startu aplikace využijeme snadné rozhraní, kde jsou zobrazeny všechny metody (Obr. 25), jedná se o obecný náhled, kde se graficky označí právě probíhající část kódu s tím, že je možné se „vnořovat hlouběji“.

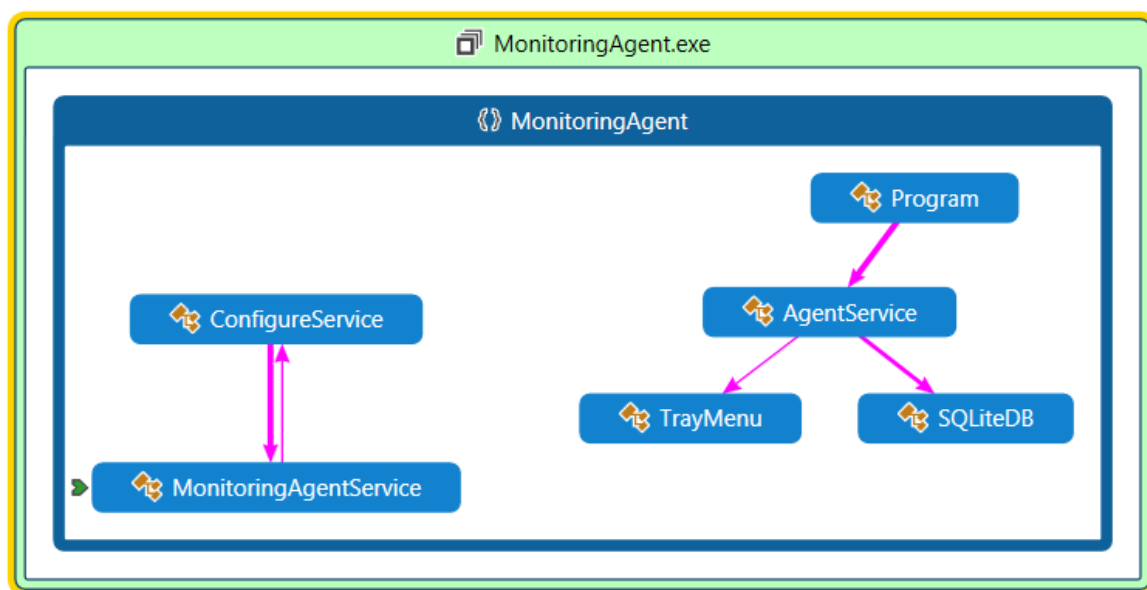
Při běhu třídy „**Program**“ (Obr. 26) obsahující v sobě více metod poskytuje o něco komplexnější náhled.

Hlavní částí aplikace je třída „**AgentService**“ (Obr. 27), která obsahuje největší množství metod a tudíž je i grafické rozhraní velmi komplexní a pro nezainteresovaného pozorovatele může působit i složitě.

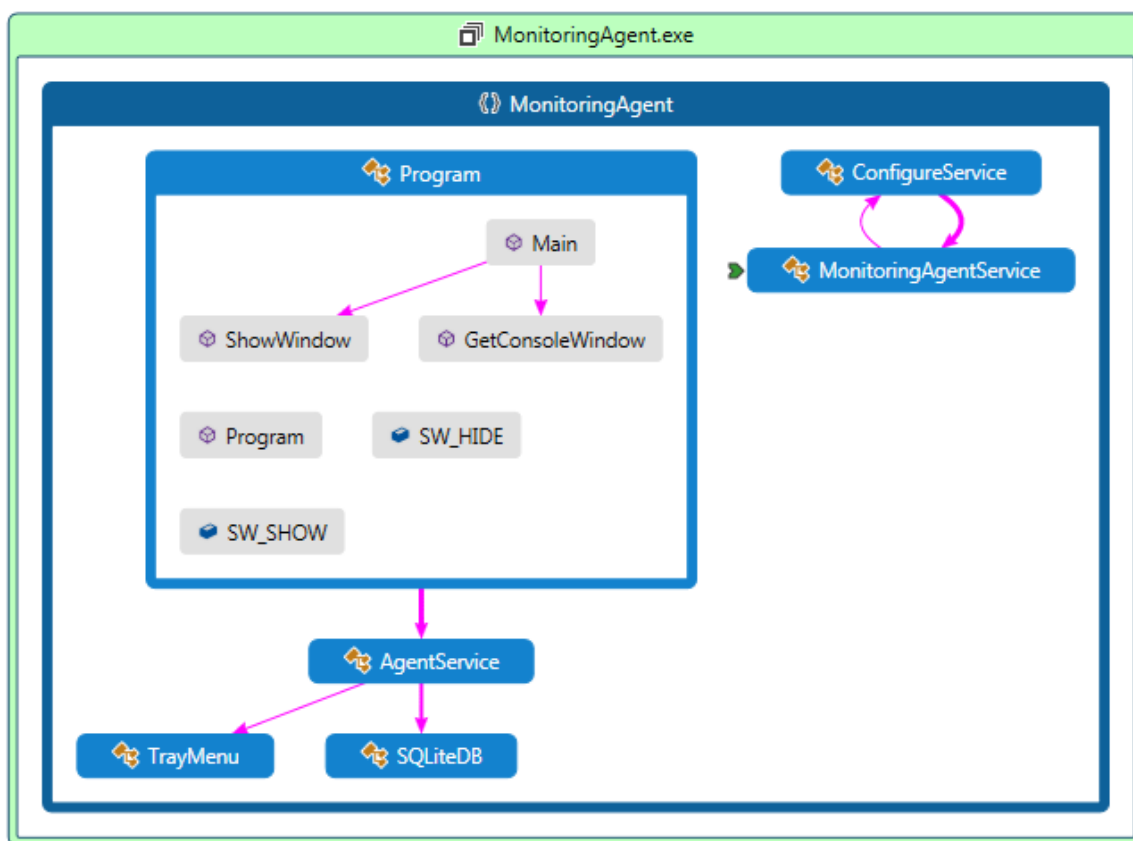
Při práci s databází SQLite je možné využít diagram týkající se této třídy (Obr. 28), jsou zde uvedeny metody odvolávané v rámci třídy. Jak je vidět nadřazeným prvkem pro metodu „**SQLiteDB**“ je metoda „**AgentService**“.

Grafické zobrazení menu má samostatnou třídu „**TrayMenu**“ (Obr. 29), obsahuje velké množství prvků a právě mapa kódu pomáhá s laděním.

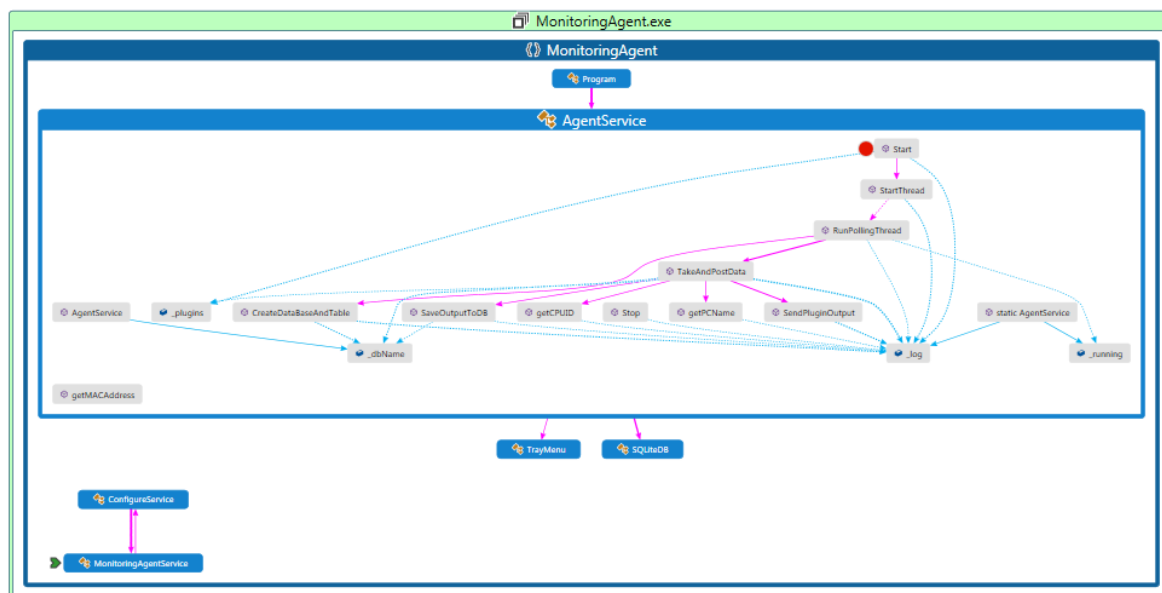
Načítání pluginů je samostatný projekt, proto je vazba označena přerušovanou čarou (Obr. 30). Třída „**PluginsCollection**“ (Obr. 31) je dynamicky se načítající objekt.



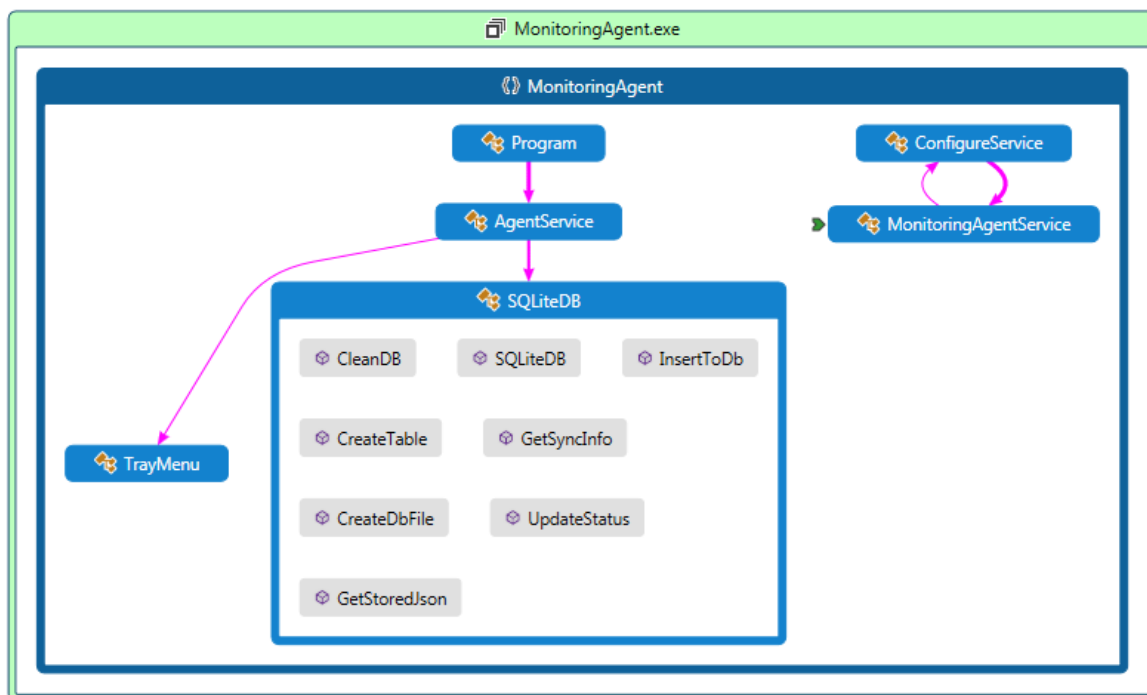
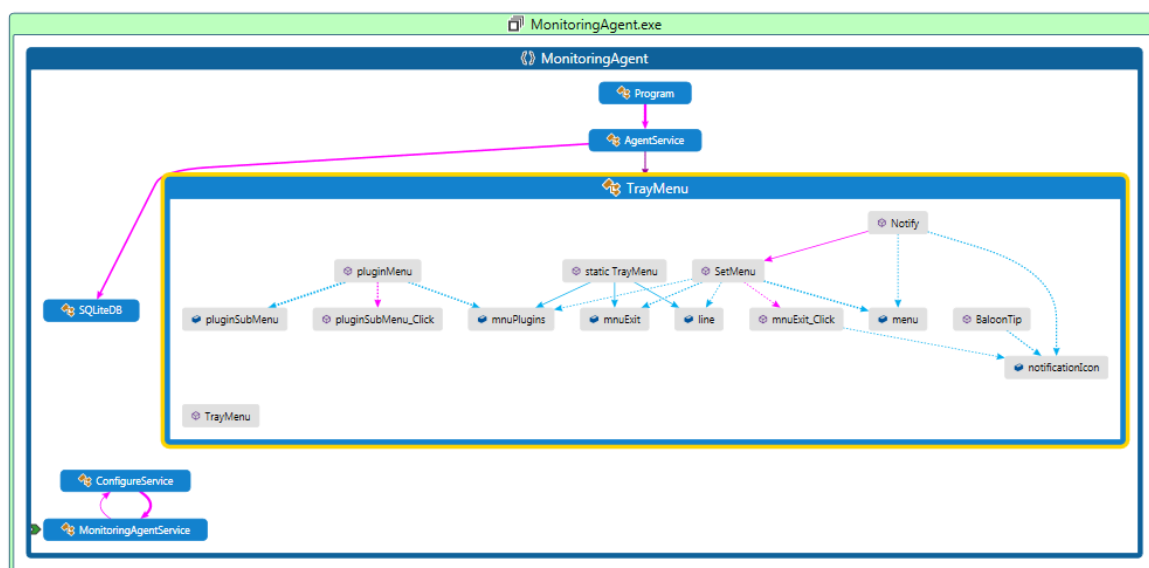
Obr. 25. Celkový pohled na mapu kódu

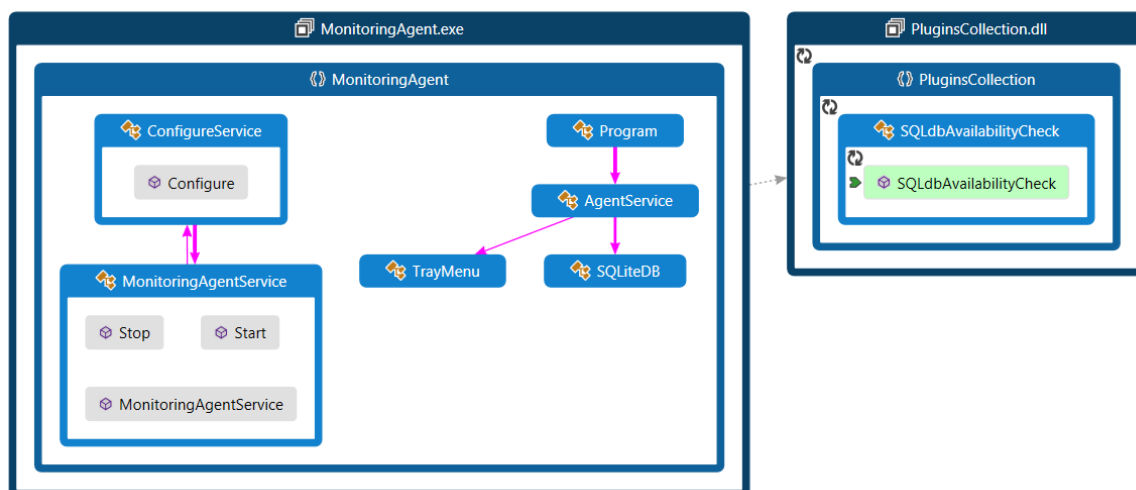


Obr. 26. Běh třídy Program

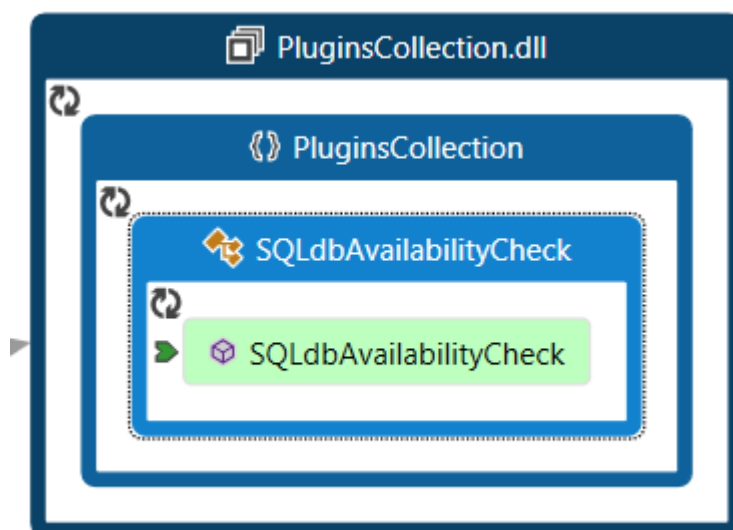


Obr. 27. Detail třídy AgentService

Obr. 28. Diagram třídy `SQLiteDb`Obr. 29. Diagram třídy `TrayMenu`



Obr. 30. Diagram s vazbou na PluginsCollection



Obr. 31. Ukázka volání dostupnosti SQL databáze v rámci pluginu

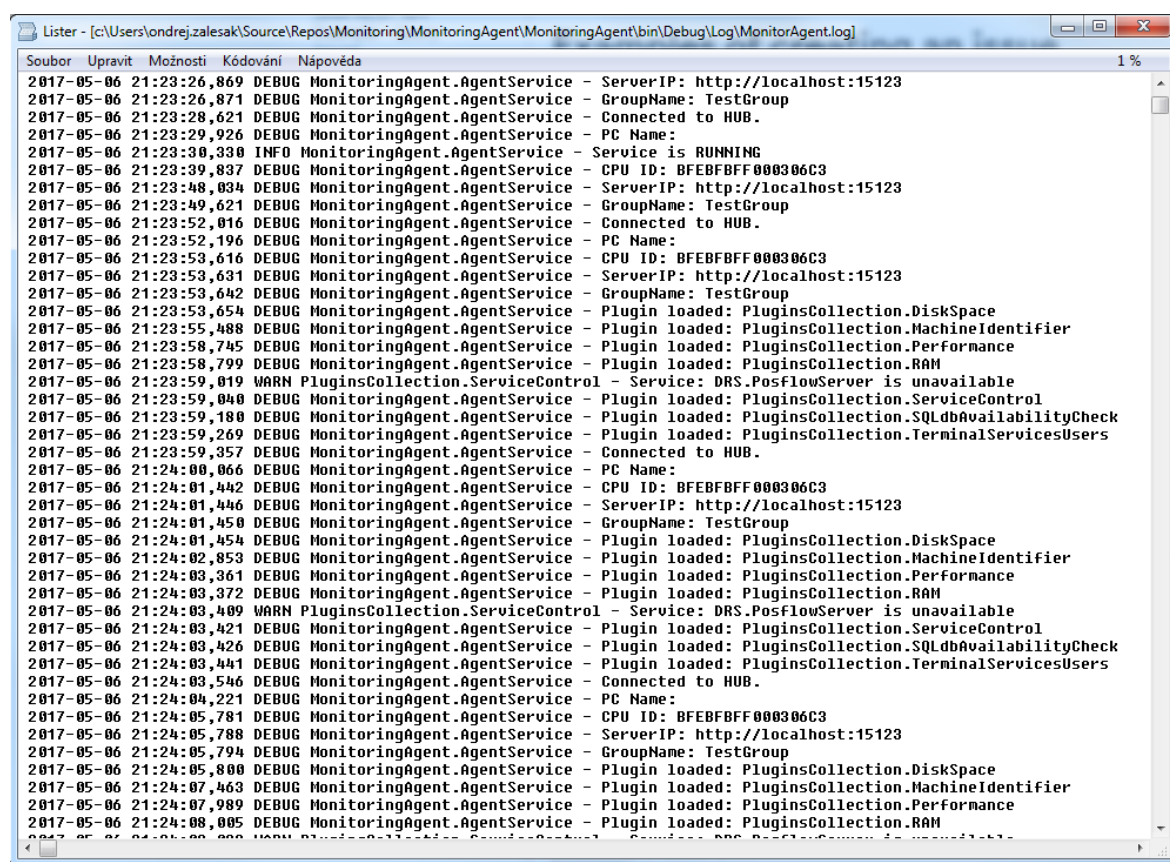
10.7 Logování

Aplikace při svém běhu generuje na základě předem definovaných událostí logovací soubor (Obr. 32). Jedná se o textový soubor, který obsahuje informace o datu a čase události ve formátu:

YYYY-MM-DD HH:mm:ss,fff

Význam znaků je tento:

Y = Year (rok)
M = Month (měsíc)
D = Day (den)
H = Hour (hodina)
m = Minute (minuta)
s = Second (vteřina)
fff = Milisecond (milisekunda)



Obr. 32. Textové zobrazení logu aplikace

Aby mohla aplikace tento soubor generovat, používá k tomu knihovnu **log4net**, což je knihovna od společnosti Apache určená pro .NET Framework a dovolující tvorbu logu s širokým spektrem možností a nastavení.

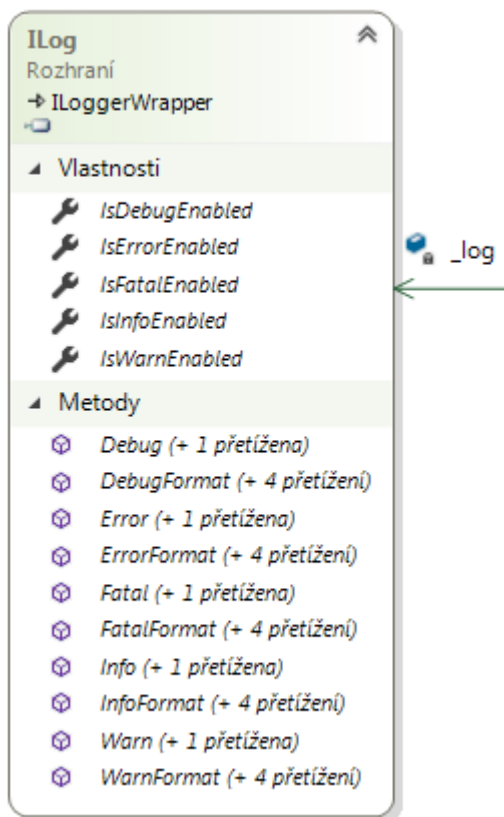
Pro volání logovací služby slouží rozhraní ILog (Obr. 33).

Práce s ní je velmi snadná a intuitivní, kód pro logování může vypadat v případě logování na úrovni „**Info**“ následovně:

```
_log.Info("Service is STOPPED");
```

Pokud by se jednalo o úroveň „**Error**“, je kód tento, do chyba je vypsána i text výjimky:

```
_log.Error("Polling Thread Aborted.\n\n" + e.ToString());
```



Obr. 33. log4net a jeho rozhraní ILog

Na základě nastavení v souboru **App.config.xml** knihovna log4net provede daný úkon. Jak je možné vidět na (Obr. 34), momentálně je nastaveno logování tak, aby zapisovalo každou událost, protože obrázky jsou z období ladění aplikace a vývojář má v takovém případě zájem na tom, aby měl přehled o každé události. Jakmile je aplikace doladěna a dojde k nasazení do ostrého provozu, úroveň logování se zpravidla nastavuje na úroveň „**ERROR**“, což znamená, že loguje pouze hlášení, které jsou typu „**ERROR**“ a „**FATAL**“-

```
<log4net>
  <root>
    <!-- level value="OFF" / -->
    <!-- level value="FATAL" / -->
    <!-- level value="ERROR" / -->
    <!-- level value="WARN" / -->
    <!-- level value="INFO" / -->
    <!-- level value="DEBUG" / -->
    <level value="ALL" />
    <appender-ref ref="log4NetAppender" />
  </root>
  <appender name="log4NetAppender" type="log4net.Appender.RollingFileAppender">
    <appendToFile value="true" />
    <maxSizeRollBackups value="30" />
    <maximumFileSize value="5MB" />
    <rollingStyle value="Size" />
    <staticLogFileName value="false" />
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
    <file value=".\Log\MonitorAgent.log" />
    <param name="AppendToFile" value="true" />
    <!-- filter type="log4net.Filter.LevelRangeFilter">
      <param name="LevelMin" value="ERROR"/>
      <param name="LevelMax" value="ERROR"/>
    </filter -->
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date %level %logger - %message%newline" />
    </layout>
  </appender>
</log4net>
```

Obr. 34. Ukázka konfiguračního souboru log4net

Další důležité parametry, které je možné nastavovat, jsou následující:

- **appendToFile** = Nastavení, které určuje, že se každý záznam přidává k existujícímu souboru
- **maximumFileSize** = Maximální velikost souboru, pokud tato velikost přesáhne uvedenou hodnotu, vytvoří se nový soubor a logování pokračuje v něm.
- **file** = umístění souboru, do kterého se loguje, cesta je relativní, tudíž se vztahuje k umístění aplikace.

10.8 Nasazení do reálného provozu

System je momentálně připraven s nasazením do reálného provozu, přičemž aktuálně probíhají testy na několika strojích.

Zatím se aplikace jeví jako stabilní a s nízkými nároky na běh, což jsou jedny z hlavních bodů zadání.

Pro nasazení do ostrého provozu je určena infrastruktura několika zákazníků firmy XY, kteří dali souhlas s tím, že se budou podílet na pilotním běhu aplikace a jsou srozuměni s tím, že existuje jistá míra pravděpodobnosti, která by mohla přinést předem nespécifikované problémy.

Vzhledem k tomu, že firma XY dodává ERP systém, je víceméně předem dáno, jaké testy budou na strojích probíhat, a díky integritě řešení bude možné snadno porovnávat jednotlivá prostředí napříč zákazníky.

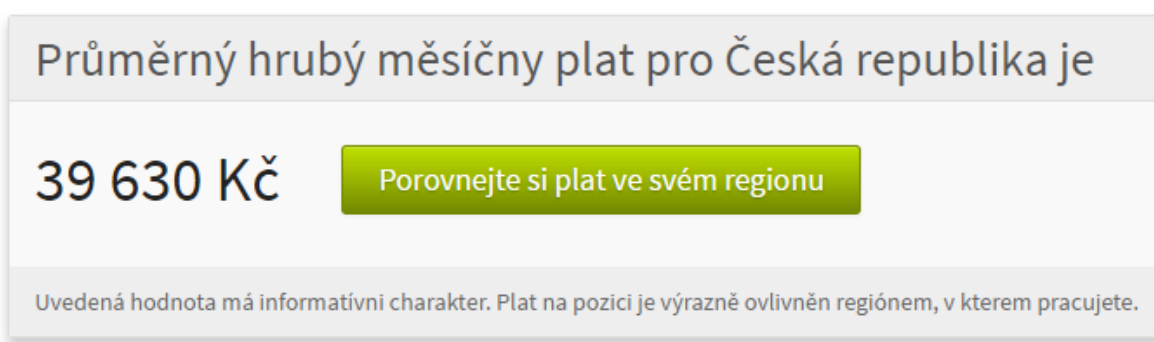
Po dokončení pilotního provozu je očekáváno sbírání informací o běhu, případné ladění aplikace, doplnění nových pluginů a následně rollout na všechny zákazníky firmy, což by mělo poskytnout přínos v kontrole jejich prostředí.

10.9 Ohodnocení nákladů implementace

Rozsah implementace klientské části monitorovacího systému je evidován jako 28 člověkodní (ManDays).

Pokud vezmeme průměrný měsíční plat pro programátora C#, který je v rámci ČR **39.630 Kč** (Obr. 35), měsíční náklady zaměstnavatele na zaměstnance jsou potom **53.185 Kč**.

Programátor C#



Obr. 35. Průměrná mzda Programátora C# v ČR [66]

Pokud počítáme 20 denní pracovní měsíc, jedná se o 1,4 pracovního měsíce a výsledné náklady na vývoj jsou tedy: $53.185 * 1,4 = 74.459 \text{ Kč}$.

Do této ceny nejsou zahrnuty náklady na testování a další vedlejší personální náklady a bonusy.

10.10 Plánovaný rozvoj

Aplikace je momentálně ve stavu, kdy je připravena k nasazení ve funkcionalitě, která je uvedena v této práci, nicméně už teď je jasné, že bude pokračovat další vývoj a rozšiřování v závislosti na požadavcích, které budou vzneseny a to jak směrem k optimalizaci výkonu, tak k rozšíření monitorovacích schopností.

Díky způsobu vývoje, který byl zvolen, je aplikace velmi flexibilní a otevřená k dalším úpravám a rozšíření.

Jako první krok, který je třeba udělat v rámci rozšíření, bude směřování aplikace k větší škálovatelnosti a pohodlnějším možnostem správy. Pokud by bylo třeba ji nasadit na větší množství zařízení, určitě by nebylo vhodné každou instanci spravovat ručně, ale jako vhodné řešení se jeví centrální správa.

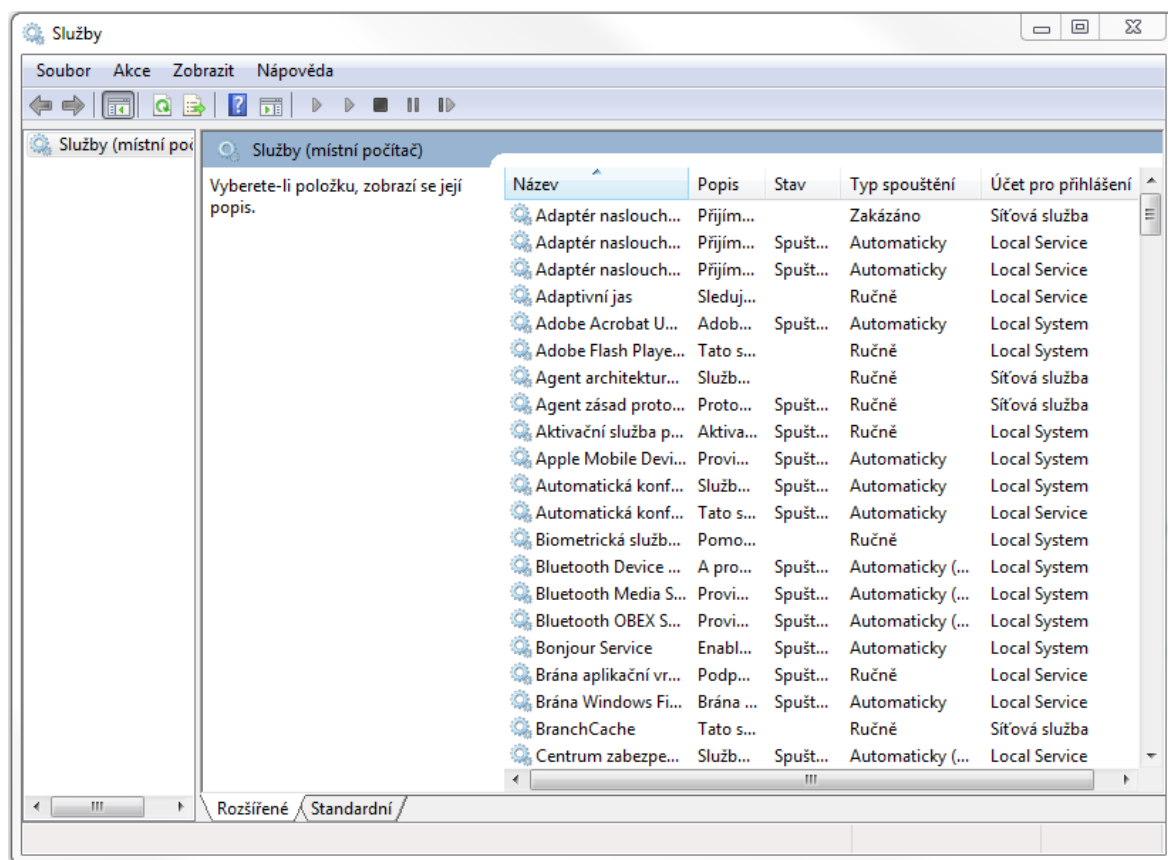
10.10.1 Možnost vzdáleně ovládat služby běžící na klientském stroji

Důležitou funkcionalitou každého systému, který monitoruje běh služeb je to, že dokáže vedle kontroly stavu služby jejich běh také ovládat.

Existují tyto stavy služeb systému Windows (Obr. 36):

- **Running** (běžící)
- **Stopped** (ukončená)
- **Paused** (pozastavená)
- **Stopping** (ukončuje se)
- **Starting** (nabíhá)

Na základě uživatelských oprávnění bude možné ze serverové aplikace zastavovat a spouštět služby, tato funkcionalita bude dosažena přes mechanismus `SignalR`, kdy server využije otevřené komunikace a na základě identifikátoru zařízení a identifikátoru služby zašle požadavek na změnu stavu služby, která bude následně provedena.



Obr. 36. Ukázka služeb Windows a jejich stavů

10.10.2 Možnost stahovat nové verze pluginů a aplikace přes FTP

Tak, jak bude přibývat pluginů a jejich verzí, stejně jako nových verzí aplikace, nebo případných hotfixů, bude třeba vyřešit systém updatování, tedy to, jak dostat updatovací balíček na klientský stroj a tam provést update na novou verzi, stáhnout nové pluginy, nebo opravit existující chyby.

Plánované řešení je to, že bude existovat FTP server, kde budou v předem dané struktuře uloženy updatovací balíčky označené jasnou identifikací, která bude obsahovat číslo verze a datum. Příklad:

MonitoringAgent_v10011_20170501.zip

Pokud dostane klient instrukci k tomu, aby si data stáhl, pokusí se otevřít FTP spojení na úložiště a data stáhnout. V případě, že data stáhne, provede update, případně doplní plugin. V opačném případě zašle na centrální server zprávu o tom, že se mu nepodařilo otevřít FTP spojení a je třeba problém odstranit.

V řešení je samozřejmě třeba dotáhnout i otázku autentifikace, přístup na FTP server nesmí být veřejně dostupný a musí existovat zabezpečení přístupu.

10.10.3 Připojení systému pro evidenci hlášení

Dalším krokem v rozvoji monitorovacího systému bude rozhraní, které dovolí monitorovacímu systému komunikovat se systémem pro evidenci hlášení, jako jsou například:

- **JIRA**
- **Redmine**
- **Bugzilla**
- **Trac**
- **Mantis**

Tyto systémy obsahují možnost pro zaslání hlášení pomocí API, často se jedná o REST API (Obr. 37).

Aby bylo možné toto obsloužit, systém musí být schopen zavolat webovou službu daného systému a zaslat zprávu v předem daném formátu.

Je také důležité, aby byl administrátor schopen rozlišit, která hlášení je třeba do systému zadat a také to, aby se hlášení neduplikovaly, tudíž každé odeslané hlášení musí být označeno.

Request

```
curl -D- -u fred:fred -X POST --data {see below} -H "Content-Type: application/json" http://localhost:8090/rest/api/2/issue/
```

Data

```
{
  "fields": {
    "project": {
      {
        "key": "TEST"
      },
      "summary": "REST ye merry gentlemen.",
      "description": "Creating of an issue using project keys and issue type names using the REST API",
      "issuetype": {
        "name": "Bug"
      }
    }
  }
}
```

Response

```
{
  "id": "39000",
  "key": "TEST-101",
  "self": "http://localhost:8090/rest/api/2/issue/39000"
}
```

Obr. 37. Příklad volání REST API pro založení ticketu v systému JIRA [64]

10.10.4 Integrace e-mailového klienta

Důležitou součástí monitorovacího systému je, stejně jako integrace na ticketovací systém, i integrace e-mailového klienta.

Tento klient má systému poskytnout možnost reportovat hlášení na předem definovanou skupinu uživatelů právě pomocí e-mailových zpráv.

V menu pro konfiguraci administrátor určí, jaké zprávy bude klient odesílat a v jaké frekvenci (jednou, opakovaně).

Bude použit jedno, nebo více z následujících e-mailových API:

- Mailgun
- Sendgrid
- Mandrill
- MailJet
- Amazon SES
- PostmarkApp [64]

10.10.5 Rozšíření možností návratových hodnot pluginů

Momentálně aplikace podporuje návratové hodnoty pluginů typu:

- Table (Tabulka)
- Unknown (Neznámý)
- Graph (Graf)

Do budoucna bude tyto možnosti třeba rozšířit například na typ struktura, nebo jednodušší typy jako jsou například typy:

- Boolean
- Integer

10.10.6 Příprava aplikace pro multiplatformní prostředí

V plánu je přepis aplikace do jazyka .NET Core, který poskytuje možnosti běhu v multiplatformních prostředích a není vázán pouze na operační systém Windows.

Proběhla zkouška a aplikace byla přepsána do frameworku .NET Core 1.1, ale bohužel ne všechny části běžely správně a podpora .NET Core verze 1.1 zatím není tak rozsáhlá, aby dovolila běh aplikace tak, jako by běžela v prostředí .NET Framework 4.6.2.

Dle dostupných informací bude verze 2.0 vydána ve 3. kvartálu roku 2017 (Obr. 38)

Ship Dates

Milestone	Release Date
.NET Core 2.0 Preview	Released on 2017/5/10
.NET Standard 2.0 Preview	Released on 2017/5/10
.NET Core 2.0	Q3 2017
.NET Standard 2.0	Q3 2017
.NET Core 2.1 (.NET Standard 2.0 for UWP)	Win10 Fall Creators Update

Obr. 38. Plánované datum vydání .NET Core verze 2.0 [62]

11 ZÁVĚR

V rámci práce proběhla analýza řešení pro monitoring a rešerše jejich vlastností, kladů a záporů. Proběhlo hodnocení pěti aplikací, které jsou dostupné na trhu. Bylo zjištěno, že jejich technické provedení i poskytované možnosti jsou obecně na velmi vysoké úrovni. Aplikace vesměs nabízejí možnosti uživatelské konfigurace a doplnění o vlastní zásuvné moduly, některé z nich mají dokonce možnost kontroly průběhu kódu vybraných aplikací, což napomáhá k rychlé analýze a často tato možnost dokáže odhalit problémy dříve, než skutečně mohou nastat.

Oproti funkčním a nefunkčním specifikům daných aplikací stojí jejich cena, která se skrze vybrané projekty velmi liší. Nejlevnější aplikace lze pořídit za 300 USD, nejdražší z nich vyjde na 25.000 USD, což je více než 80-násobný rozdíl. Každá z nich také nabízí jinou možnost licencování, některé z nich poskytují jednorázovou licenci, po jejímž nákupu se zákazník stane majitelem na neomezenou dobu, jiné mají možnost pronájmu.

Na základě analýzy trhu a specifikaci požadavků bylo, zejména kvůli možnostem optimalizace a dalšího rozvoje, přistoupeno ke kroku vývoje vlastního software, který bude mít možnosti šité na míru požadavkům zadávající firmy. Firma má několik desítek zákazníků, kteří používají její ERP a pokladní systémy a prozatím nemá komplexní software, který by dokázal poskytnout kontrolu běhu infrastruktury v reálném čase.

Došlo k návrhu modelu systému v konfiguraci klient-server, kdy každá část byla tvořena jedním vývojářem. Tato práce se zabývá částí klientskou, což je konzolová aplikace běžící jako služba napsaná v programovacím jazyce Visual C# a využívající funkcionalitu .NET Framework 4.6.2. Aplikace využívá systém zásuvných modulů (tvz. „pluginů“), které jsou při spuštění dynamicky načteny a vykonají požadované testy, jež jsou následně zaslány na serverovou část, která slouží jako zobrazovací a analytická část.

Po zhodnocení nákladů na vývoj klientské části bylo vypočítáno, že náklady na vývoj vyšly firmu na 74.459 Kč. Toto je částka, kterou stál samotný vývoj bez dalších personálních nákladů. Předpokladem je, že rozšíření se bude pohybovat v řádově nižších částkách, jelikož je základ řešení dostatečně robustní.

Do budoucna je plánováno několik rozšíření, které bude díky flexibilitě řešení velmi snadné implementovat. Dalším plánovaným krokem je refaktorizace kódu pro běh pod frameworkem .NET Core.

SEZNAM POUŽITÉ LITERATURY

- [1] Introduction to SignalR. *Introduction to SignalR* [online]. USA: Microsoft, 2014 [cit. 2017-04-29]. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr#what-is-signalr>
- [2] JavaScript: The Good Parts. Youtube [online]. USA: Youtube [cit. 2017-04-29]. Dostupné z: <https://www.youtube.com/watch?v=hQVTIJBZook&t=2405>
- [3] A Modern ReIntroduction to AJAX. *JavaScriptCoder.com* [online]. USA: JavaScriptCoder, 2017 [cit. 2017-04-29]. Dostupné z: <http://www.javascript-coder.com/tutorials/re-introduction-to-ajax.phtml>
- [4] JSON Redux AKA RFC7159. *Tbray.org* [online]. USA: Tim Bray, 2014 [cit. 2017-04-29]. Dostupné z: <https://www.tbray.org/ongoing/When/201x/2014/03/05/RFC7159-JSON>
- [5] Douglas Crockford - The JSON Saga. *Youtube.com* [online]. USA: Youtube, 2011 [cit. 2017-05-01]. Dostupné z: <https://www.youtube.com/watch?v=-C-JoyNuQJs>
- [6] The JSON Data Interchange Format. *ECMA International* [online]. USA: ECMA, 2013 [cit. 2017-05-01]. Dostupné z: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>
- [7] JSON Web Token. *IETF* [online]. USA: IETF, 2015 [cit. 2017-05-01]. Dostupné z: <https://tools.ietf.org/html/rfc7519>
- [8] JSON. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-01]. Dostupné z: <https://en.wikipedia.org/wiki/JSON>
- [9] JSON: The JavaScript subset that isn't. *Timelessrepo.com* [online]. USA: Magnus Holm, 2011 [cit. 2017-05-01]. Dostupné z: <http://timelessrepo.com/json-isnt-a-javascript-subset>
- [10] JQuery - Format a Microsoft JSON date. *Stackoverflow.com* [online]. USA, 2016 [cit. 2017-05-01]. Dostupné z: <https://stackoverflow.com/questions/206384/how-do-i-format-a-microsoft-json-date>
- [11] JSON Schema. *Json-schema.org* [online]. USA, 2015 [cit. 2017-05-01]. Dostupné z: <http://json-schema.org/>
- [12] JSON Schema: A Media Type for Describing JSON Documents: draft-wright-json-schema-00. *Json-schema.org* [online]. USA, 2016 [cit. 2017-05-01]. Dostupné z: <http://json-schema.org/latest/json-schema-core.html>

- [13] JSON Schema Software. *Json-schema.org* [online]. USA: JSON-Schema, 2016 [cit. 2017-05-01]. Dostupné z: <http://json-schema.org/implementations>
- [14] JSON: The Fat-Free Alternative to XML. *Json.org* [online]. USA: json.org, 2011 [cit. 2017-05-01]. Dostupné z: <http://www.json.org/xml.html>
- [15] YAML Ain't Markup Language (YAML™) Version 1.2. *Yaml.org* [online]. USA: yaml.org, 2009 [cit. 2017-05-01]. Dostupné z: <http://www.yaml.org/spec/1.2/spec.html>
- [16] Microsoft Visual Studio. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio
- [17] Visual Studio Development Environment Model. *Msdn.microsoft.com* [online]. USA: Microsoft, 2008 [cit. 2017-05-01]. Dostupné z: [https://msdn.microsoft.com/en-us/library/bb165114\(VS.80\).aspx](https://msdn.microsoft.com/en-us/library/bb165114(VS.80).aspx)
- [18] VSPackages and Managed Package Framework (MPF). *Msdn.microsoft.com* [online]. USA: Microsoft, 2008 [cit. 2017-05-01]. Dostupné z: [https://msdn.microsoft.com/en-us/library/bb166554\(VS.80\).aspx](https://msdn.microsoft.com/en-us/library/bb166554(VS.80).aspx)
- [19] Who Needs the Internet of Things? *Linux.com* [online]. USA: Eric Brown, 2016 [cit. 2017-05-04]. Dostupné z: <https://www.linux.com/news/who-needs-internet-things>
- [20] 21 Open Source Projects for IoT. *Linux.com* [online]. USA: Eric Brown, 2016 [cit. 2017-05-04]. Dostupné z: <https://www.linux.com/NEWS/21-OPEN-SOURCE-PROJECTS-IOT>
- [21] Internet of Things Global Standards Initiative. *ITU.int* [online]. USA: ITU, 2015 [cit. 2017-05-04]. Dostupné z: <http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>
- [22] VERMESAN, Ovidiu a Peter FRIESS. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems* [online]. Aalborg, Denmark: River Publishers., 2013 [cit. 2017-05-04]. ISBN 978-87-92982-96-4. Dostupné z: [Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013](#)
- [23] An Introduction to the Internet of Things (IoT). *Cisco.com* [online]. San Francisco, California, USA: Lopez Research, 2013 [cit. 2017-05-04]. Dostupné z: http://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf
- [24] The Internet of Things: Between the Revolution of the Internet and the Metamorphosis of Objects. *European Commission Community Research and Development Information Service* [online]. EU: Gérald Santucci, 2013 [cit. 2017-05-04]. Dostupné z: <http://cordis.europa.eu/fp7/ict/enet/documents/publications/iot-between-the-internet-revolution.pdf>

- [25] Internet of Things Done Wrong Stifles Innovation. *Informationweek.com* [online]. EU: Information Week, 2014 [cit. 2017-05-04]. Dostupné z: <http://www.informationweek.com/strategic-cio/executive-insights-and-innovation/internet-of-things-done-wrong-stifles-innovation/a/d-id/1279157>
- [26] Nice VS 2008 Code Editing Improvements. *Weblogs ASP.NET* [online]. USA: GUTHRIE, Scott, 2007 [cit. 2017-05-11]. Dostupné z: <https://weblogs.asp.net/scottgu/nice-vs-2008-code-editing-improvements>
- [27] Dumps: Visual Studio 2005. *MSDN Microsoft* [online]. USA: Microsoft, 2007 [cit. 2017-05-11]. Dostupné z: [https://msdn.microsoft.com/en-us/library/d5zhxt22\(VS.80\).aspx](https://msdn.microsoft.com/en-us/library/d5zhxt22(VS.80).aspx)
- [28] Walkthrough: Debugging at Design Time: Visual Studio 2005. *MSDN Microsoft* [online]. USA: Microsoft, 2007 [cit. 2017-05-11]. Dostupné z: [https://msdn.microsoft.com/en-us/library/83hd8f1e\(VS.80\).aspx](https://msdn.microsoft.com/en-us/library/83hd8f1e(VS.80).aspx)
- [29] ASP.NET: Learn About ASP.NET MVC. *Asp.net* [online]. USA: Microsoft, 2013 [cit. 2017-05-11]. Dostupné z: <https://www.asp.net/mvc>
- [30] OpenMP in Visual C++: Visual Studio 2005. *MSDN Microsoft* [online]. USA: Microsoft, 2005 [cit. 2017-05-11]. Dostupné z: [https://msdn.microsoft.com/en-us/library/tt15eb9t\(VS.80\).aspx](https://msdn.microsoft.com/en-us/library/tt15eb9t(VS.80).aspx)
- [31] C#. *Docs.microsoft.com* [online]. USA: Microsoft, 2016 [cit. 2017-05-11]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/articles/csharp/csharp>
- [32] What's New in Visual Basic and Visual C#: Visual Studio .NET 2003. *MSDN Microsoft* [online]. USA: Microsoft, 2005 [cit. 2017-05-11]. Dostupné z: <https://msdn.microsoft.com/en-us/library/aa984213%28VS.71%29.aspx>
- [33] What's New in Web Development for Visual Studio. *MSDN* [online]. USA: Microsoft, 2005 [cit. 2017-05-14]. Dostupné z: [https://msdn.microsoft.com/en-us/library/s57a598e\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/s57a598e(v=vs.80).aspx)
- [34] Team Foundation Server. *MSDN* [online]. USA: Microsoft, 2008 [cit. 2017-05-14]. Dostupné z: [https://msdn.microsoft.com/en-us/library/ms181238\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/ms181238(v=vs.90).aspx)
- [35] Microsoft debuts Visual Studio 2015 and .NET 2015 previews, free Visual Studio Community 2013. *Venture Beat* [online]. USA: Emil Protalinski, 2014 [cit. 2017-05-14]. Dostupné z: <https://venturebeat.com/2014/11/12/microsoft-debuts-visual-studio-2015-and-net-2015-previews-free-visual-studio-community-2013/>
- [36] VS 2010 Licensing Changes. *Microsoft* [online]. USA, 2009 [cit. 2017-05-14]. Dostupné z: <https://blogs.msdn.microsoft.com/bharry/2009/10/19/vs-2010-licensing-changes/>

- [37] Porovnání nabídek sady Visual Studio 2017. *Microsoft* [online]. USA, 2016 [cit. 2017-05-14]. Dostupné z: <https://www.visualstudio.com/cs/vs/compare/>
- [38] Introducing Visual Studio 97: A Well-stocked Toolbox for Building Distributed Apps. *Microsoft* [online]. USA, 1997 [cit. 2017-05-14]. Dostupné z: <https://www.microsoft.com/msj/0597/visualstudio97.aspx>
- [39] MUELLER, John. *Visual Studio 6: the complete reference*. Berkeley, Calif.: Osborne/McGraw-Hill, c1999. ISBN 978-007-8825-835.
- [40] PANDEY, Nitin., Yesh. SINGHAL a Mridula. PARIHAR. *Visual studio .NET all-in-one desk reference for dummies: the complete reference*. New York, NY: Hungry minds, c2002. ISBN 978-0764516269.
- [41] STEPHEN R. G. FRASER, Nitin., Yesh. SINGHAL a Mridula. PARIHAR. *Managed C and .NET development: [Visual Studio .NET 2003 edition]*. Berkeley, Calif: Apress, 2003. ISBN 978-159-0590-331.
- [42] NABOULSI, Zain., Sara FORD a Mridula. PARIHAR. *Coding faster: getting more productive with Microsoft Visual Studio : covers Microsoft Visual Studio 2005, 2008, and 2010*. Redmond, WA: Microsoft Press, c2011. ISBN 978-0735649927.
- [43] BANKS, Richard, Sara FORD a Mridula. PARIHAR. *Learn asp.net 4.5, c# and visual studio 2012 expert skills with the smart: courseware..* New Edition. S.l.: The Smart Method, 2013. ISBN 978-190-9253-056.
- [44] JOHNSON, Bruce., Sara FORD a Mridula. PARIHAR. *Professional visual studio 2013: courseware..* S.l.: The Smart Method, 2013. ISBN 978-1118832042.
- [45] JOHNSON, Bruce, Sara FORD a Mridula. PARIHAR. *Professional Visual Studio 2015: courseware..* Indianapolis: John Wiley Sons, 2015. Programmer to programmer. ISBN 978-1119068051.
- [46] Visual Studio 2017 je venku. *Microsoft Blog* [online]. USA: Microsoft, 2017 [cit. 2017-05-14]. Dostupné z: <https://blogs.msdn.microsoft.com/vyvojari/2017/03/07/visual-studio-2017-hotove/>
- [47] JAY A. KREIBICH. *Using SQLite*. Sebastopol, CA: O'Reilly, 2010. ISBN 978-059-6521-189.
- [48] NEWMAN, Chris. *SQLite*. Indianapolis, Ind.: Sams, c2005. ISBN 978-0672326851.
- [49] HALDAR, Sibsankar. *Inside SQLite*. Sebastopol, Calif: O'Reilly, 2007. ISBN 978-059-6550-066.

- [50] D. Richard Hipp. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-14]. Dostupné z: https://en.wikipedia.org/wiki/D._Richard_Hipp
- [51] VESPA, Roberto. *Signalr realtime application cookbook*. S.l.: Packt Publishing Limited, 2014. ISBN 978-178-3285-952.
- [52] AGUILAR, José M. *SignalR programming in Microsoft ASP.NET*. S.l.: Packt Publishing Limited, 2014. ISBN 978-0735683884.
- [53] LERMAN, Julia. a Rowan. MILLER. *Programming Entity Framework: Code First*. Sebastopol, CA: O'Reilly Media, c2012. ISBN 978-1449312947.
- [54] MEYLER, Kerrie., Cameron FULLER a John. JOYNER. *System center 2012 operations manager unleashed: Code First*. Indianapolis, Ind.: Sams, c2013. Unleashed. ISBN 978-0672335914.
- [55] How to buy System Center 2016. *Microsoft.com* [online]. USA: Microsoft, 2015 [cit. 2017-05-14]. Dostupné z: <https://www.microsoft.com/en-us/cloud-platform/system-center-pricing>
- [56] NPM Getting Started Guide. *Solarwinds.com* [online]. USA: SolarWinds, 2017 [cit. 2017-05-14]. Dostupné z: [https://support.solarwinds.com/Success_Center/Network_Performance_Monitor_\(NPM\)/NPM_Documentation](https://support.solarwinds.com/Success_Center/Network_Performance_Monitor_(NPM)/NPM_Documentation)
- [57] SolarWinds Product Pricing. *Solarwinds.com* [online]. USA: SolarWinds, 2017 [cit. 2017-05-14]. Dostupné z: <http://www.solarwinds.com/products/pricing/>
- [58] Product Overview. *AppDynamics.com* [online]. USA: AppDynamics, 2017 [cit. 2017-05-14]. Dostupné z: <https://www.appdynamics.com/product/>
- [59] AppDynamics Pricing. *G2crowd.com* [online]. USA: g2crowd, 2017 [cit. 2017-05-14]. Dostupné z: <https://www.g2crowd.com/products/appdynamics/pricing>
- [60] DynaTrace APM. *Dynatrace.com* [online]. USA: Dynatrace, 2017 [cit. 2017-05-14]. Dostupné z: <https://www.dynatrace.com/capabilities/application-performance-management/>
- [61] ScienceLogic Review. *Trustradius.com* [online]. USA: TrustRadius, 2017 [cit. 2017-05-14]. Dostupné z: <https://www.trustradius.com/reviews/sciencelogic-2017-02-17-14-50-31>
- [62] .NET Core Roadmap. *Github.com* [online]. USA: GitHub, 2017 [cit. 2017-05-14]. Dostupné z: <https://github.com/dotnet/core/blob/master/roadmap.md>

- [63] JIRA REST API Example - Create Issue. *Developer.atlassian.com* [online]. USA: Atlassian, 2017 [cit. 2017-05-14]. Dostupné z: <https://developer.atlassian.com/jiradev/jira-apis/jira-rest-apis/jira-rest-api-tutorials/jira-rest-api-example-create-issue>
- [64] List of 10+ Email APIs. *Mashape.com* [online]. USA: Orlando, 2015 [cit. 2017-05-15]. Dostupné z: <http://blog.mashape.com/list-of-10-email-apis/>
- [65] Priority levels of issues. *Drupal.org* [online]. USA: Drupal, 2017 [cit. 2017-05-15]. Dostupné z: <https://www.drupal.org/core/issue-priority>
- [66] Programátor C#: Průměrný hrubý měsíční plat pro Česká republika je. *Platy.cz* [online]. ČR: Profesia CZ, 2017 [cit. 2017-05-15]. Dostupné z: <http://www.platy.cz/platy/informacni-technologie/programator-csharp>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACID	Atomicity Consistency Isolation Durability
AJAX	Asynchronous JavaScript and XML.
API	Application Programming Interface.
BCL	Base Class Library
CD	Compact Disc
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
COM	Component Object Model
CORS	Cross-Origin Resource Sharing
CR	Carriage Return
CRT	C Runtime
CSS	Cascading Style Sheet
CSV	Comma Separated Values
CTP	Community Technology Preview
DBMS	Database Management System
ECMA	European Computer Manufacturers Association
ERP	Enterprise Resource Planning
FTS	Full Text Search
GNU	GNU's Not Unix
GUI	Graphical User Interface
HP	Hewlett-Packard
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HW	Hardware

IBM	International Business Machines
IDE	Integrated Development Environment
IIS	Internet Information Service
IoT	Internet of Things
JSON	JavaScript Object Notation
LF	Line Feed
MEF	Managed Extensibility Framework
MFC	Microsoft Foundation Class
MS	Microsoft
MSDN	Microsoft Developer Network
MSI	Microsoft Installer
MSIL	Microsoft Intermediate Language
MSSCCI	Microsoft Source Code Control Interface
MVC	Model View Controller
Obr.	Obrázek
OGDL	Ordered Graph Data Language
OS	Operační systém
PNG	Portable Network Graphics
QA	Quality Assurance
RAM	Random Access Memory
RC	Release Candidate
RFC	Request for comments
RPC	Remote Procedure Call
SCOM	System Center Operations Manager
SDK	Software Development Kit
SNMP	Simple Network Management Protocol

SQL	Structure Query Language
SW	Software
TFS	Team Foundation Server
UI	User Interface
UID	Unikátní identifikátor
UML	Unified Modeliing Language
VB	Visual Basic
VBA	Visual Basic for Applications
VS	Visual Studio
VSTA	Visual Studio for Applications
WAL	Write Ahead Logging
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations
YAML	Yet Another Markup Language

SEZNAM OBRÁZKŮ

Obr. 1. Prostředí Visual Studio 2017	15
Obr. 2. Příklad použití formátu JSON [8]	32
Obr. 3. Porovnání cen monitorovacích systémů na trhu	49
Obr. 4. Příklad Load Balancingu ve schematickém zobrazení	51
Obr. 5. Diagram Use Case pro monitorovací systém	55
Obr. 6. Ukázka běhu konzolové aplikace MonitoringAgent	59
Obr. 7. Zobrazení stromu projektu v IDE VS	60
Obr. 8. Ukázka verze použitého jazyka C#	60
Obr. 9. Ukázka aplikace běžící v liště	60
Obr. 10. Menu aplikace	61
Obr. 11. Vizualizace kompletního menu včetně výstupů z testů	61
Obr. 12. DiskSpace	65
Obr. 13. MachineIdentifier	66
Obr. 14. Třída Performance	67
Obr. 15. Třída RAM	68
Obr. 16. SQLdbAvailabilityCheck	69
Obr. 17. TerminalServicesUsers	70
Obr. 18. IPlugin	71
Obr. 19. Metody ve třídě SQLiteDB	73
Obr. 20. Tabulka MonitoringAgentCache v SQLite db	73
Obr. 21. Prostředí SQLiteStudio 3.1.1	74
Obr. 22. Diagram tříd včetně vzájemných vazeb	75
Obr. 23. Třídy aplikace ve zjednodušeném zobrazení	75
Obr. 24. Schéma tříd včetně metod a polí	76
Obr. 25. Celkový pohled na mapu kódu	77
Obr. 26. Běh třídy Program	78
Obr. 27. Detail třídy AgentService	78
Obr. 28. Diagram třídy SQLiteDb	79
Obr. 29. Diagram třídy TrayMenu	79
Obr. 30. Diagram s vazbou na PluginsCollection	80
Obr. 31. Ukázka volání dostupnosti SQL databáze v rámci pluginu	80
Obr. 32. Textové zobrazení logu aplikace	81

Obr. 33. log4net a jeho rozhraní ILog	82
Obr. 34. Ukázka konfiguračního souboru log4net.....	83
Obr. 35. Průměrná mzda Programátora C# v ČR [66].....	84
Obr. 36. Ukázka služeb Windows a jejich stavů	86
Obr. 37. Příklad volání REST API pro založení ticketu v systému JIRA [64].....	88
Obr. 38. Plánované datum vydání .NET Core verze 2.0 [62].....	90

SEZNAM TABULEK

Tab. 1. Porovnání cen a licencování monitorovacích systémů	49
--	----

SEZNAM PŘÍLOH

P I Zdrojový kód aplikace na přiloženém CD.