Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh a implementace systému pro centrální správu software ve firmě

Bc. Jiří Kejmar

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jiří Kejmar

Informatika

Název práce

Návrh a implementace systému pro centrální správu software ve firmě

Název anglicky

Design and implementation of system for centralized management of software in a company

---

Cíle práce

Diplomová práce je zaměřena na problematiku vývoje aplikačního software na platformě Windows s využitím .NET Frameworku. Hlavním cílem práce je s využitím znalostí získaných analýzou odborných informačních zdrojů a na základě známých uživatelských požadavků zpracovat analýzu a návrh systému pro centrální správu software ve firemním prostředí a tento systém posléze implementovat a otestovat.

Metodika

Metodika řešené práce je založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy teoretických poznatků a uživatelských požadavků bude navržen a implementován prototyp aplikace sloužící k centrální správě uživatelského software na počítačích ve firemní síti.

Doporučený rozsah práce

60-80 stran

Klíčová slova

C#, .NET, Windows, aktualizace, centralizovaná správa, software, databáze

---

Doporučené zdroje informací

ALBAHARI, Joseph, Ben ALBAHARI a Peter DRAYTON. C# 5.0 in a nutshell. 5th ed. Sebastopol: O'Reilly, 2012. ISBN 978-144-9320-102

DOSTÁLEK, Libor. Velký průvodce protokoly TCP/IP a systémem DNS. Vyd. 2. Praha: Computer Press, 2000. ISBN 80-722-6323-4.

SHARP, John. Microsoft Visual C# 2010: krok za krokem. Vyd. 1. Brno: Computer Press, 2010. ISBN 978-80-251-3147-3.

SPAANJAARS, Imar. Beginning ASP.NET 4.5 in C# and VB. Indianapolis, Ind.: J. Wiley & Sons, Inc., 2013. ISBN 978-1-118-31180-6.

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 21. 03. 2016

Čestné prohlášení

Prohlašuji, že svou diplomovou práci *Návrh a implementace systému pro centrální správu software ve firmě* jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších dostupných informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury v závěru práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením žádným způsobem neporušil autorská práva třetích osob.

V Praze dne 29. března 2016

Poděkování

Rád bych na tomto místě poděkoval vedoucímu diplomové práce panu Ing. Jiřímu Brožkovi, Ph. D. za vedení a cenné rady při vypracování diplomové práce.

Návrh a implementace systému pro centrální správu software ve firmě

Souhrn

Teoretická část diplomové práce se zabývá problematikou prostředí .NET Framework, vývojovým prostředím Microsoft Visual Studio a programovacím jazykem C#.

Praktická část diplomové práce se zabývá problematikou správy software ve firmě a následnou implementací vlastního řešení. V první části je zpracována analýza aktuálního procesu, na základě které jsou definovány požadavky pro návrh optimalizovaného procesu centrální správy software. Tento proces je navržen pomocí vícevrstvé architektury a zahrnuje uživatelské rozhraní na platformě ASP.NET, centrální databázi SQLite, vnitřní logiku systému realizovanou pomocí služby systému Windows a navržením centralizovaného úložiště zdrojových dat, aplikačních skriptů a instalačních záznamů. V druhé části jsou zpracovány návrhy a implementace jednotlivých částí systému. V závěru kapitoly je představena metodika ověření vytvořeného prototypu.

Závěrečné zhodnocení shrnuje diplomovou práci, zvolená řešení a výsledky testů provedených v reálném prostředí. Vytvořený systém umožňuje jednoduchým způsobem provádět centrální správu aplikací ve firmě.

Klíčová slova: C#, .NET Framework, ASP.NET, Windows, centrální správa software, databáze, SQLite, WMI, Windows služba.

Design and implementation of system for centralized management of software in a company

Summary

The theoretical part of the thesis deals with the Microsoft .NET Framework, development environment Visual Studio and C # programming language issues.

The practical part of the thesis deals with the Software management issues in company and the subsequent implementation of a customized solution. In the first part the practical part focuses on analysis of the current process which is based on defined requirements of the proposal for optimizing the processes of the central management software. This process is designed using a multi-layer architecture and includes a user interface on the ASP.NET platform, a central database SQLite, the internal logic of the system implemented using Windows system service and proposing a centralized repository of the source data, application scripts and installation records. In the second part focuses on the design and implementation of individual parts of the system. In the last chapter the verification methodology of the created prototype is introduced.

The final evaluation summarizes the thesis, selected solutions and the results of the tests performed in a real environment. The created system allows the simple way to centrally manage applications in a company.

Keywords: C#, .NET Framework, ASP.NET, Windows, Software management, database, SQLite, WMI, Windows service.

Obsah

1 Úvod ... 13
2 Cíl práce a metodika ... 14
2.1 Cíl práce ... 14
2.2 Metodika ... 14
3 Teoretická východiska ... 15
3.1 Microsoft .NET Framework ... 15
3.1.1 Historie ... 16
3.1.2 Architektura ... 17
3.1.3 Kompilace ... 19
3.1.4 Spuštění ... 21
3.1.5 Kompatibilita ... 23
3.1.6 Služby ... 23
3.1.7 Systémové požadavky ... 25
3.2 Microsoft Visual Studio ... 25
3.2.1 Funkce ... 25
3.2.2 Edice ... 26
3.3 Programovací jazyk C# ... 27
3.3.1 Objektově orientované programování ... 28
3.3.2 Syntaxe C# ... 28
3.3.3 Proměnné ... 29
3.3.4 Modifikátory ... 30
3.3.5 Rozhodovací příkazy ... 30
3.3.6 Metoda ... 34
3.3.7 Přetěžování metod ... 34
3.3.8 Třída ... 35
3.3.9 Konstruktor ... 35
3.3.10 Ošetřování výjimek ... 35
3.3.11 Hodnotové a referenční typy ... 37
3.3.12 Pole a kolekce ... 37
4 Problematika správy software ve firmě ... 39
4.1 Analýza produktivního procesu ... 39
4.1.1 Správa software ... 39

4.1.2 Instalační zdroje ... 40
4.1.3 Přístup ke vzdálené stanici ... 40
4.1.4 Evidence software ... 40
4.2 Zhodnocení produktivního procesu ... 41
4.3 Identifikované požadavky ... 41
4.4 Návrh nového procesu ... 42
4.4.1 Optimalizovaný proces ... 42
4.4.2 Uživatelské scénáře ... 43
4.4.3 Uživatelské role ... 44
4.4.4 Požadavky ... 44
5 Vlastní řešení ... 46
5.1 Databáze ... 46
5.1.1 Návrh ... 46
5.1.1 Implementace ... 49
5.2 Webová aplikace ... 51
5.2.1 Návrh ... 51
5.2.2 Implementace ... 65
5.3 Služba ... 70
5.3.1 Návrh ... 70
5.3.2 Implemenatace ... 71
5.4 Organizace dat ... 79
5.5 Testování ... 81
5.5.1 Metodika ... 82
5.5.2 Výsledky ... 82
6 Výsledky a diskuse ... 83
7 Závěr ... 84
8 Seznam použitých zdrojů ... 85
9 Seznam zkratek ... 87
10 Přílohy ... 88
A. LogFile stanice ... 88
B. LogFile služby ... 89
C. UseCase ... 94

Seznam obrázků

Obrázek 3.1: Vrstvy 15
Obrázek 3.2: Základní architektura .NET Framework 18
Obrázek 3.3: Dnešní pohled na architekturu .NET Framework 19
Obrázek 3.4: Klasická kompilace 20
Obrázek 3.5: Kompilace v .NET Framework 21
Obrázek 3.6: Common Language Runtime – JIT kompilace 22
Obrázek 3.7: CLR v prostředí managed a unmanaged aplikací 22
Obrázek 3.8: .NET Core 23
Obrázek 3.9: Edice Visual Studio 2015 26
Obrázek 3.10: Typový systém .NET 37
Obrázek 4.1: Vícevrstvý návrh systému 43
Obrázek 4.2: Stavový diagram – změny stavu požadavku 45
Obrázek 5.1: Datový model databáze 47
Obrázek 5.2: DB Browser for SQLite 48
Obrázek 5.3: Statický web 51
Obrázek 5.4: Dynamický web 52
Obrázek 5.5: Logický design – Homepage 53
Obrázek 5.6: Grafický design – Homepage 53
Obrázek 5.7: Logický design – Instalace 54
Obrázek 5.8: Grafický design – Instalace 54
Obrázek 5.9: Logický design – Odinstalace 55
Obrázek 5.10: Grafický design – Odinstalace 55
Obrázek 5.11: Logický design – Aktualizace 55
Obrázek 5.12: Grafický design – Aktualizace 56
Obrázek 5.13: Logický design – Aktualizace software 56
Obrázek 5.14: Grafický design – Aktualizace software 56
Obrázek 5.15: Logický design – Aktualizace software – potvrzení 57
Obrázek 5.16: Grafický design – Aktualizace software – potvrzení 57
Obrázek 5.17: Logický design – Aktualizace hardware 58
Obrázek 5.18: Grafický design – Aktualizace hardware 58
Obrázek 5.19: Logický design – Aktualizace hardware 58

Obrázek 5.20: Grafický design – Aktualizace hardware – potvrzení 59
Obrázek 5.21: Logický design – Požadavky 59
Obrázek 5.22: Grafický design – Požadavky 60
Obrázek 5.23: Grafický design – Požadavky – Žádné požadavky 60
Obrázek 5.24: Logický design – Evidence 61
Obrázek 5.25: Grafický design – Evidence 61
Obrázek 5.26: Logický design – Problémy 62
Obrázek 5.27: Grafický design – Problémy 62
Obrázek 5.28: Grafický design – Problémy – Žádné problémy 62
Obrázek 5.29: Responzivní design – šířka 1920px 63
Obrázek 5.30: Responzivní design – šířka 1024px 64
Obrázek 5.31: Responzivní design – šířka 480px 64
Obrázek 5.32: Organizace dat 80
Obrázek 5.33: Nastavení administrátorských práv pro skript 81

Seznam příkladů

Příklad 3.1: Program C# – Hello, World! 29
Příklad 3.2: Definice proměnné 30
Příklad 3.3: Podmínka if 31
Příklad 3.4: Podmínka switch 32
Příklad 3.5: Cyklus while 32
Příklad 3.6: Cyklus for 33
Příklad 3.7: Cyklus do 33
Příklad 3.8: Cyklus foreach 33
Příklad 3.9: Metoda 34
Příklad 3.10: Přetížení metody 34
Příklad 3.11: Konstruktor 35
Příklad 3.12: Přetížený konstruktor 35
Příklad 3.13: Ošetřování chyb – try, catch a finally 36
Příklad 3.14: Vyvolání výjimky 36
Příklad 5.1: Vytvoření tabulky software 49
Příklad 5.2: Vytvoření tabulky hardware 49

Příklad 5.3: Vytvoření tabulky records 50
Příklad 5.4: Vytvoření tabulky queue 50
Příklad 5.5: Vložení záznamů do tabulky 50
Příklad 5.6: Kód Site.master – obsah zjednodušen 65
Příklad 5.7: Kód install.aspx 66
Příklad 5.8: Kód install.aspx.cs – obsah metod skryt 67
Příklad 5.9: Metoda AddNewQueue 68
Příklad 5.10: Metoda LoadRecords – obsah zjednodušen 69
Příklad 5.11: Metoda CheckData 69
Příklad 5.12: Metoda DeleteQueue 70
Příklad 5.13: Vícevláknové zpracování 72
Příklad 5.14: Zamykání objektů – obsah zjednodušen 73
Příklad 5.15: Logování 74
Příklad 5.16: Požadavky 74
Příklad 5.17: Ukázka komunikace s databází – obsah zjednodušen 76
Příklad 5.18: Spuštění vzdáleného procesu 77
Příklad 5.19: Zjištění dostupnosti stanici v doméně 78
Příklad 5.20: Kopírování souborů 79

Seznam tabulek

Tabulka 3.1: Historický vývoj .NET Framework 16
Tabulka 3.2: Systémové požadavky pro .NET Framework 4 – 4.6 25
Tabulka 3.3: C# – základní syntaxe 29
Tabulka 3.4: C# – porovnávací a logické operátory 31

1 Úvod

V současné době stále narůstá využití výpočetní techniky ve všech oblastech a společně také roste množství používaného software. Tento trend se projevuje v malých, středních i velkých firmách.

Využívání obrovského množství software klade vysoké nároky na IT provoz i management. To s sebou přináší rostoucí personální náklady na administrátory.

Většina firem si nemůže, v dnešní ekonomicky nevyrovnané době, dovolit růst nepřímých nákladů v IT oblasti a raději se soustředí na důkladnou optimalizaci procesů.

Dále se přidávají požadavky definující nutnost udržovat centrální evidenci aplikací z důvodů softwarového auditu, správy licencí k zajištění legálnosti provozovaných aplikací a v neposlední řadě k optimalizaci počtu licencí a druhů pořízeného software.

Z těchto důvodů se autor rozhodnul analyzovat aktuální proces, navrhnout prototyp systému a tento prototyp posléze implementovat. Vytvořený prototyp bude pracovat na platformě. NET, pomůže sjednotit proces aktualizace software a zajistí jeho důslednou evidenci.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je zaměřena na problematiku vývoje aplikačního software na platformě Windows s využitím .NET Frameworku.

Hlavním cílem práce je s využitím znalostí získaných analýzou odborných informačních zdrojů a na základě známých uživatelských požadavků zpracovat analýzu a návrh systému pro centrální správu software ve firemním prostředí a tento systém posléze implementovat a otestovat.

2.2 Metodika

Metodika řešené práce je založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy teoretických poznatků a uživatelských požadavků bude navržen a implementován prototyp aplikace sloužící k centrální správě uživatelského software na počítačích ve firemní síti.

Návrh bude obsahovat analýzu produktivního procesu správy software ve firemním prostředí, na jejímž základě budou identifikovány požadavky pro návrh nového systému. Pro implementaci systému je nezbytné stanovit datový model a uživatelské rozhraní. Implementace bude provedena pomocí technologií .NET Framework, ASP.NET, databáze SQLite a jazyka C#. Vytvořený prototyp bude na závěr otestován v IT oddělení firmy Škoda Auto a.s. zabývajícím se systémovou integrací a podporou konstrukčních a datových systémů.

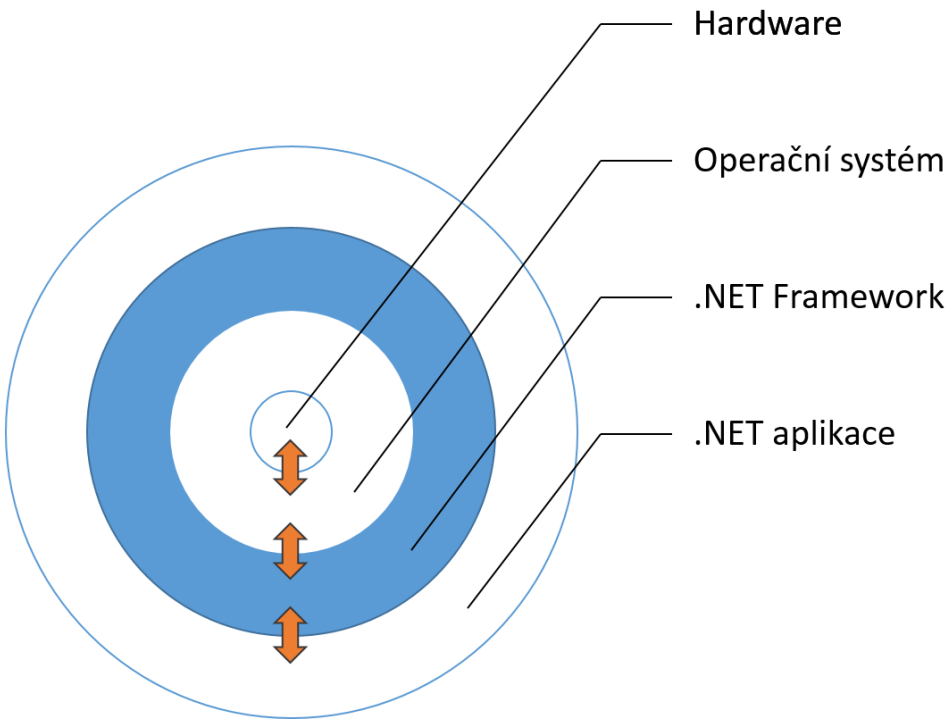
3 Teoretická východiska

3.1 Microsoft .NET Framework

.NET Framework je komplexní softwarová platforma, která je v dnešní době určena pro vývoj různorodých aplikací. Za pomoci .NET Framework lze vyvíjet nejen klasické formulářové nebo konzolové aplikace pro Windows, ale také webové aplikace, služby nebo aplikace pro mobilní zařízení a mnoho dalších. [1]

Nejedná se o operační systém ani o programovací jazyk. Obrázek 3.1 zobrazuje umístění vrstvy .NET Frameworku. Také je možné .NET Framework přirovnat virtuálnímu stroji. [2]

Obrázek 3.1: Vrstvy



Zdroj: autor

Microsoft .NET Framework se skládá ze dvou hlavních součástí. První součástí je komponenta Common Language Runtime, což je spouštěcí nástroj, který řídí běžící aplikace. Druhou součástí je knihovna tříd rozhraní .NET Framework, která poskytuje knihovnu opakovaně použitelného kódu, který mohou vývojáři použít ve vlastních aplikacích. [3]

3.1.1 Historie

Vzniku .NET Framework předcházela potřeba sjednotit běhové prostředí a vývojové nástroje, které umožní tvorbu softwaru s kompatibilním přístupem k Windows API neboli Application Programming Interface – aplikačnímu programovacímu rozhraní. [4]

Tabulka 3.1 zobrazuje historický vývoj verzí .NET Framework od roku 2002 a to včetně jeho klíčových vlastností.

Tabulka 3.1: Historický vývoj .NET Framework

VERZE	VÝVOJOVÉ PROSTŘEDÍ	VERZE C#	KLÍČOVÉ VLASTNOSTI
1.0	Visual Studio 2002	C# 1.0	• první verze Framework
1.1	Visual Studio 2003	C# 1.2	• dramatické zvýšení výkonu • rozšíření a změny API
2.0	Visual Studio 2005	C# 2.0	• nová verze runtime • podpora generiky • plná podpora 64 bitů • výrazná vylepšení knihoven a ASP.NET • parciální třídy • anonymní metody
3.0	Expression Blend	C# 3.0	• Windows Communication Foundation • Windows Presentation Foundation • Windows Workflow Foundation • Windows CardSpace
3.5	Visual Studio 2008	C# 3.0	• nové verze VB.NET a C# • LINQ • extension metody • podpora AJAX v ASP.NET • Entity Framework
4.0	Visual Studio 2010	C# 4.0	• dynamické datové typy • parallel extensions • vylepšení stávajících technologií

4.5	Visual Studio 2012	C# 5.0	• aplikace pro Metro • podpora pro pole > 2GB • asynchronní programovací model • JIT - využití vícejádrových procesorů
4.5.1	Visual Studio 2013	C# 5.0	• podpora Windows Phone Store aplikací • zlepšení výkonu a debuggování
4.5.2	Visual Studio 2013	C# 5.0	• nové API pro transakční systémy • nové API pro ASP.NET
4.6	Visual Studio 2015	C# 6.0	• využití kompilace .NET Native • vylepšené trasování událostí • technologie ASP.NET Core 5
4.6.1	Visual Studio 2015	C# 6.0	• zlepšení bezpečnosti

Zdroj: [1; 5]

Na začátku roku 2016 je poslední dostupná verze .NET Framework 4.6.1. [5]

Souběžně s postupným vývojem .NET Framework docházelo také k vývoji programovacích jazyků (např. C# viz kapitola 3.3, C++, VB.NET) a vývojového prostředí Visual Studio (viz kapitola 3.2).

3.1.2 **Architektura**

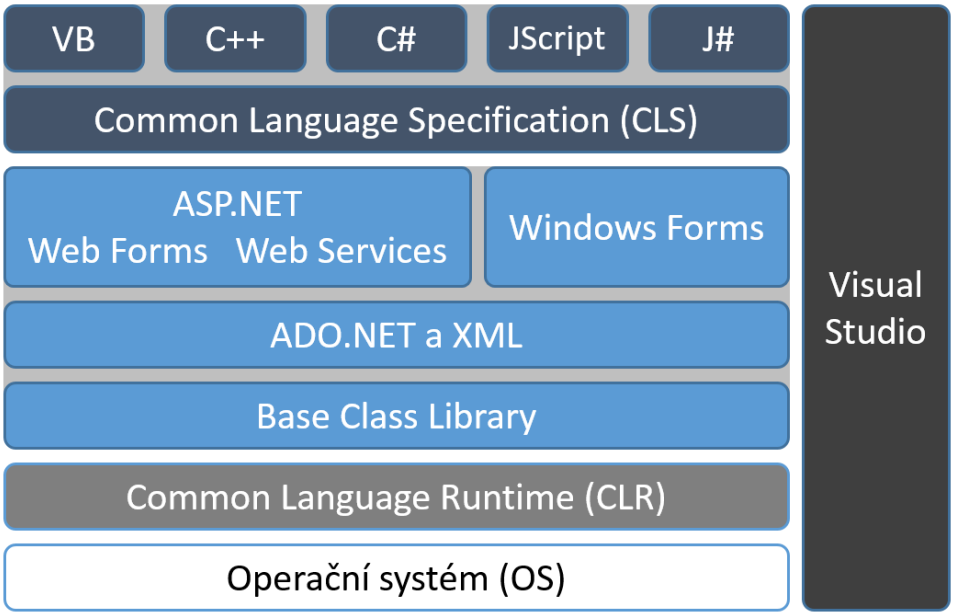
Na nejnižší úrovni se nachází CLR neboli Common Language Runtime realizující základní infrastrukturu, nad kterou je .NET Framework vybudován. Modul CLR spravuje paměť, provádí vlákna, provádí kód, ověřuje zabezpečení kódu, provádí kompilaci a další služby systému. [4; 6]

Nad CLR se nachází několik hierarchicky umístěných knihoven. Ty jsou rozděleny do jmenných prostorů. Základem je knihovna nazvaná Base Class Library. Knihovna tříd rozhraní .NET Framework je kolekce opakovaně použitelných typů, které jsou pevně integrovány s CLR .Knihovna tříd je objektově orientovaná, typová tzn., že umožňuje odvozování funkcí. Tím knihovna tříd umožnuje snížit dobu potřebnou k naučení vývoje aplikace nad .NET Frameworkem. Nad Base Class Library je knihovna pro přístup k datům a práci s XML soubory. Další vrstvou je sada knihoven usnadňující práci s uživatelským

rozhraním. Je rozdělena do dvou bloků. Tyto bloky slouží pro usnadnění vytváření webových aplikací a pro vytváření klasických formulářových aplikací. [4; 6]

Poslední vrstvu tvoří nelimitovaná množina programovacích jazyků. Jejich základní vlastnosti definuje CLS neboli Common Language Specification (viz Obrázek 3.2). [4]

Obrázek 3.2: Základní architektura .NET Framework



Zdroj: [4]

Obrázek 3.2 také dokazuje jazykovou univerzálnost a otevřenost tohoto řešení. Zdrojový kód nemusí být napsán pouze v programovacích jazycích, které jsou součástí Visual Studia (viz kapitola 3.2), ale v libovolném jazyce, jehož kompilátor bude dodržovat pravidla CLS. Všechny jazyky, které splňují požadavky CLS můžou být spuštěny na CLR (viz podkapitola 3.1.3.2). Lze uvést například jazyky Visual Basic .NET, Visual C++, Visual C#, a JScript. [7]

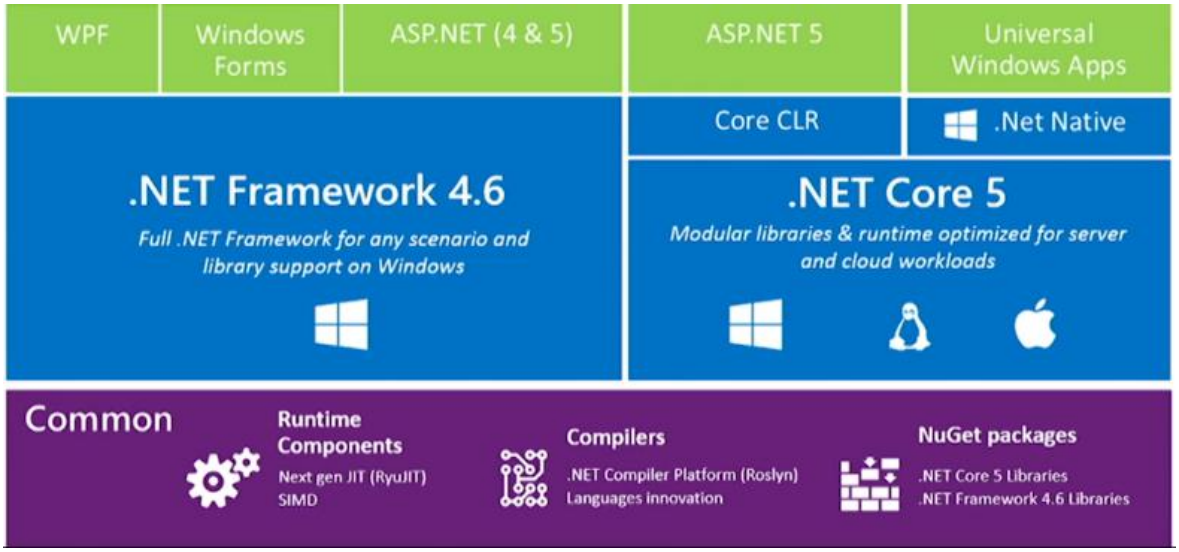
Dnešní architektura je velmi komplexní a umožnuje široké využití. Objevuje se více verzí Frameworku:

- **.NET Framework 4.6** je plnohodnotná platforma pro vše související s .NET, ale postrádá knihovny a runtime optimalizované více platformní a cloudové nasazení (Windows, Linux and Mac).
- **.NET Core** je na druhou stranu podmnožinou .NET Framework 4.6, který je optimalizováno především pro více platformní a cloudové nasazení. Více informací v kapitole 3.1.5.

- **.NET Native** se používá především k vývoji univerzálních aplikací, kde budou plnohodnotně fungovat na jakémkoli zařízení s platformou Windows. Netýká se webových aplikací. [2]

Obrázek 3.3 zobrazuje vyšší pohled na architekturu .NET Frameworku 4.6. [2]

Obrázek 3.3: Dnešní pohled na architekturu .NET Framework



Zdroj: [2]

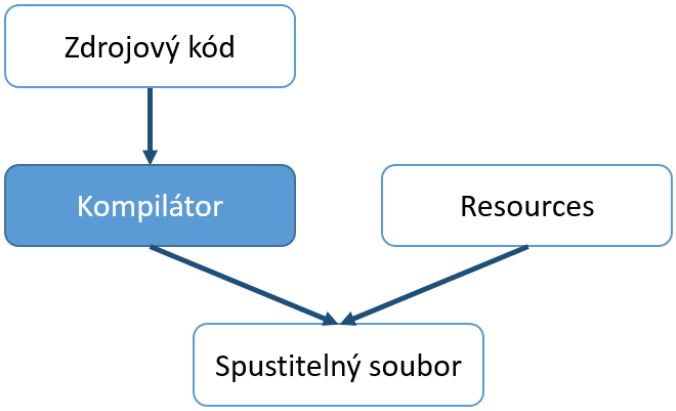
3.1.3 Kompilace

Aby bylo možné spustit jakýkoliv zdrojový kód napsaný pro .NET Framework v jeho běhovém prostředí CLR, musí být nejprve zkompilován. Pro korektní pochopení použité kompilace v prostředí .NET Framework, bude nejprve vysvětleno, jak se provádí klasická kompilace.

3.1.3.1 Klasická kompilace

V případě kompilace programu napsaného například v jazyce C++ nebo Visual Basic 6 je výstupem kompilátoru přímo spustitelný strojový kód viz Obrázek 3.4. [1]

Obrázek 3.4: Klasická kompilace



Zdroj: [1]

Ve výsledku se jedná se o tzv. unmanaged (neřízený) kód. Je zpravidla velmi rychlý, protože procesor pracuje přímo se strojovým kódem, výstup z kompilátoru je pro procesor již nachystaný. Bohužel je zde také několik nevýhod. Mezi nevýhody klasické kompilace patří náchylnost na chyby v aplikaci a také bezpečnostní rizika. [1]

Například jazyk C++ často využívá ve svém kódu tzv. pointery neboli ukazatele na konkrétní paměťová místa. Díky tomu je možné zapsat jakýkoli kód na libovolné paměťové místo. Jedna z možných chyb v aplikaci může způsobit pád aplikace, kvůli přístupu nebo zápisu na neplatnou adresu. [1]

Této vlastnosti lze také zneužít pomocí tzv. podvržení dat v paměťovém místě, kde aplikace očekává jiná data, a následně se spustí podvržená část programu. [1]

3.1.3.2 Kompilace v .NET Framework

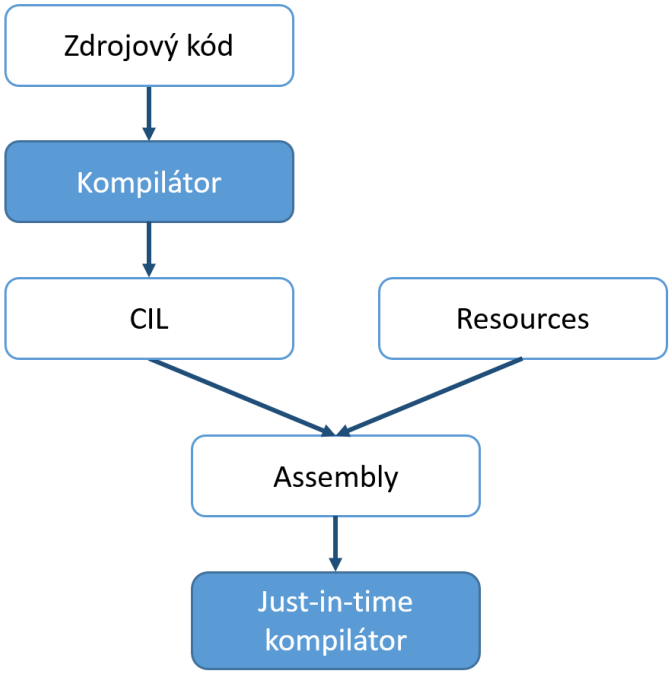
Pojem CLR neboli Common Language Runtime je možné přirovnat k pojmu virtuálního stroje používaného v programovacím jazyce Java. [4]

Kompilátor v prvním kroku nejprve zpracuje zdrojové kódy a jeho výstupem není strojový kód, tak jako u klasické kompilace, ale tzv. CIL (Common Intermediate Language) nebo MSIL (Microsoft Intermediate Language). [1]

Tento kód je již velmi podobný kódu strojovému, protože jej tvoří relativně jednoduché a nízko úrovňové instrukce. Jeho velkou výhodou je, že je nezávislý na platformě a lze spustit kdekoli, kde je dostupné běhové prostředí .NET Framework. [1]

V dalším kroku se kód přeložený do Common Intermediate Language společně s dalšími datovými soubory (Resources) zabalí do tzv. Assembly (viz Obrázek 3.5). Assembly je reprezentován jako soubor s příponou .exe nebo .dll. [1]

Obrázek 3.5: Kompilace v .NET Framework



Zdroj: [1]

Na první pohled jsou pro uživatele výsledné soubory totožné a nelze poznat, jaký typ kompilace byl použit. Velký rozdíl nastává až při jejich spuštění. [1]

3.1.4 Spuštění

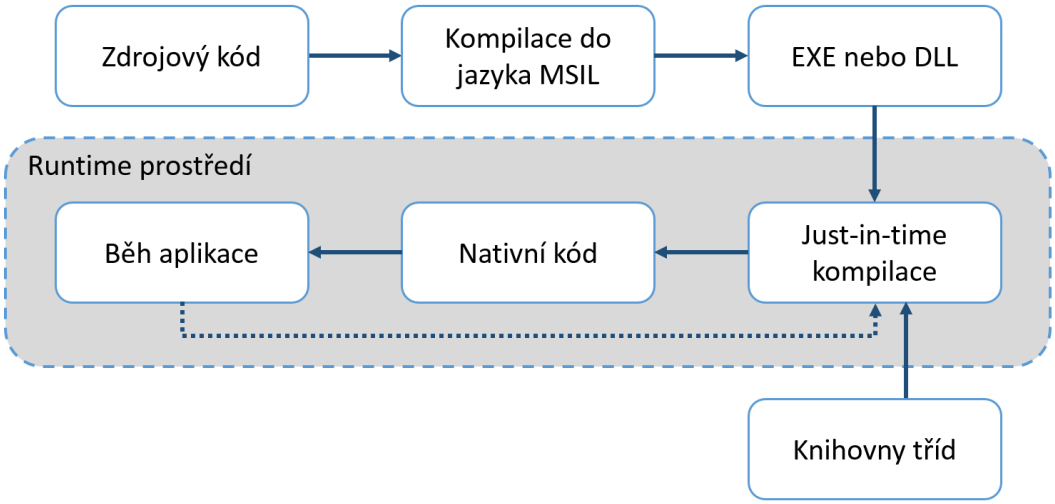
Při spuštění .NET Assembly se provede pouze tzv. částečná kompilace. Metody, které jsou pro běh aplikace nezbytné, se přeloží do strojového kódu a následně se spustí aplikace. Kompiluje se na úrovni metod (tedy funkcí a procedur). Tento kód se nazývá managed (řízený). Již od počátku se počítá s bezpečností a korektností přístupů do paměti, striktní typovou kontrolou a mechanismem obsluhy výjimek. [1]

Pokud je nutné během práce v programu zavolat funkci, pro kterou není doposud zkompilována metoda, tak ji JIT neboli Just In Time překladač v reálném čase přeloží a výsledek si uschová pro další případné zavolání. Tento typ prostředí se nazývá Runtime. [1]

Výše popsané vlastnosti odstraňují nevýhodu pomalejšího startu .NET aplikace oproti klasicky kompilované aplikaci. Společnost Microsoft dokonce tvrdí, že jde dokonce o zvýšení výkonu. Dokazuje to tím, že ke kompilaci MSIL kódu do strojového jazyka dochází až v době spuštění aplikace, kdy má kompilátor zcela jasné informace o konfiguraci počítače, především o procesoru, na kterém poběží tato aplikace. Díky tomu je aplikace optimalizována. [1]

Schematicky vyjádřené Runtime prostředí zobrazuje Obrázek 3.6.

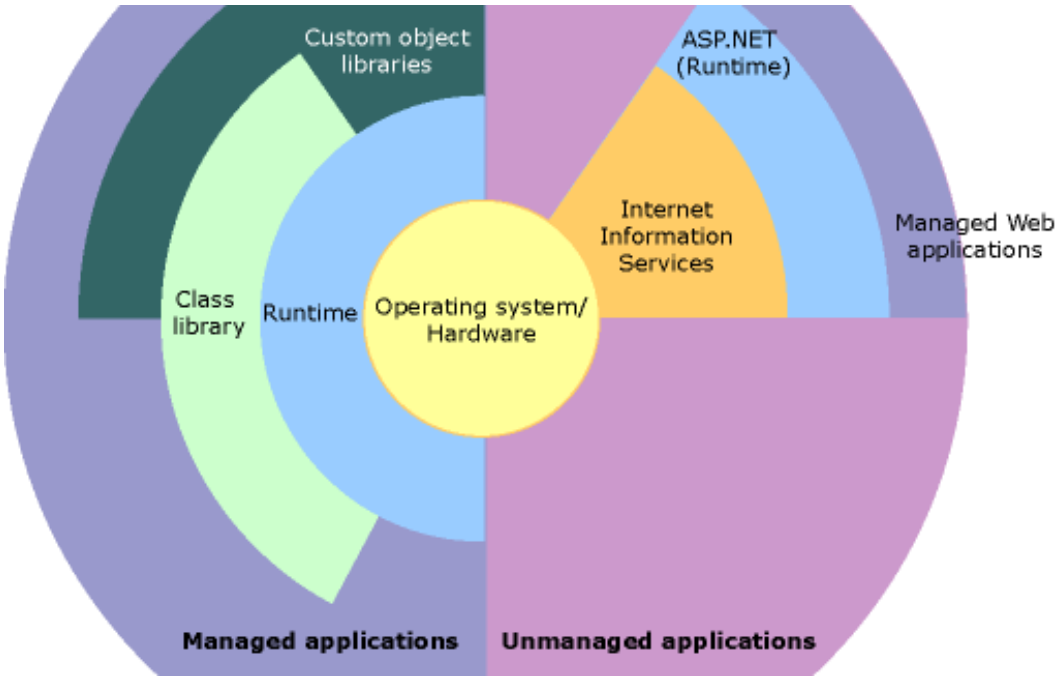
Obrázek 3.6: Common Language Runtime – JIT kompilace



Zdroj: [4]

Po vysvětlení pojmů managed a unmanaged kód lze prozkoumat následující cibulový diagram. Obrázek 3.7 ukazuje vzájemný vztah CLR (na diagramu uveden pouze jako Runtime) a knihovny tříd k aplikacím a operačnímu systému. Například technologie ASP.NET dokáže nabídnout škálovatelné, serverové řešení pro managed kód, přestože se pohybuje v unmanaged prostředí. [6]

Obrázek 3.7: CLR v prostředí managed a unmanaged aplikací



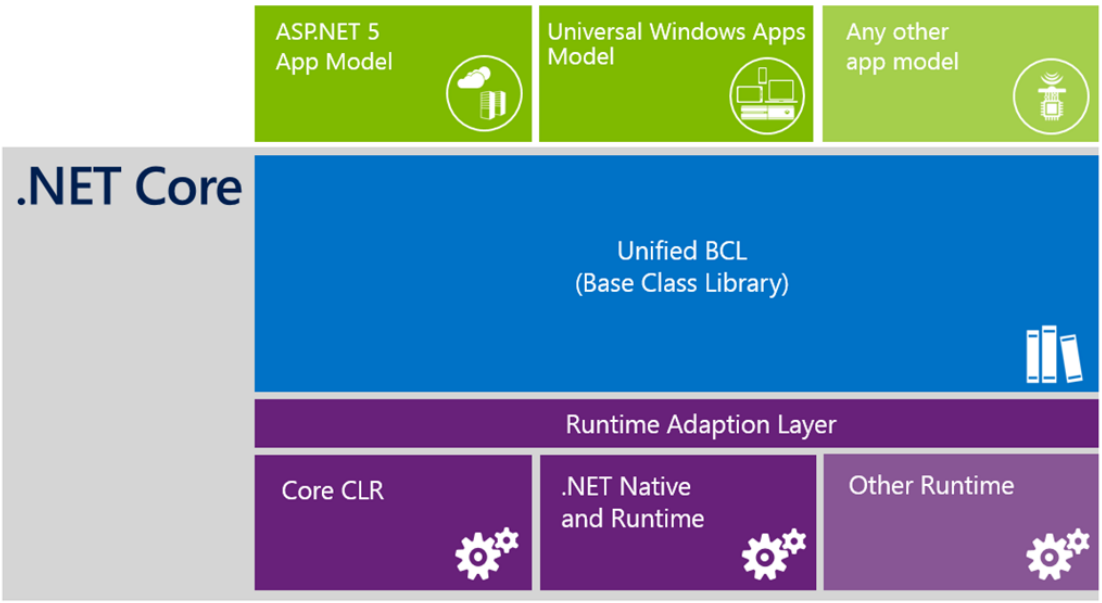
Zdroj: [6]

3.1.5 **Kompatibilita**

Pro spuštění aplikace je bezpodmínečně nutné mít nainstalovaný kompatibilní .NET Framework na koncovém zařízení s operačním systémem Windows. Na zařízení je možné instalovat více verzí .NET Frameworku. [3]

V poslední době se objevil poměrně malý optimalizovaný runtime .NET Core (viz Obrázek 3.8), který lze instalovat jak na operační systém Windows, tak i na operační systémy Mac OS a Linux. [8]

Obrázek 3.8: .NET Core



Zdroj: [2]

Jedná se vysoce výkonný a modulární návrh. Obsahuje obdobný CLR, shodný Garbage Collector a JIT jako .NET Framework, ale neosahuje vlastnosti jako aplikační domény nebo zabezpečení přístupu ke kódu. .NET Core obsahuje také do jisté míry shodné knihovny jako .NET Framework, které byly zredukovány a optimalizovány pro udržení kompaktnosti. [8; 2]

3.1.6 **Služby**

Služby poskytované rozhraním .NET Framework spuštěným aplikacím poskytují:

- **Správu paměti** - v tradičních programovacích jazycích programátoři zodpovídají za přidělování a uvolňování paměti a správu životnosti objektů. .NET Framework zajišťuje tyto služby pomocí CLR namísto samotné aplikace.
- **Systém společných typů** - v tradičních programovacích jazycích jsou kompilátorem definovány základní typy, což ztěžuje vzájemnou slučitelnost jednotlivých jazyků. V rozhraní .NET Framework jsou základní typy definovány v rozhraní.NET Framework a tím pádem jsou společné pro všechny jazyky používané v .NET Framework.
- **Rozsáhlou knihovnu tříd** - místo psaní velkého objemu zdrojového kódu, pro zpracování běžných operací nízké úrovně, mohou programátoři využít dostupnou knihovnu typů a tříd rozhraní .NET Framework.
- **Vývojové rámce a technologie** -.NET Framework obsahuje knihovny pro konkrétní oblasti vývoje aplikací, například rozhraní ASP.NET, ADO.NET nebo službu Windows Communication Foundation.
- **Vzájemnou funkční spolupráci jazyka** - Kompilátory jazyků, které jsou zaměřeny na rozhraní .NET Framework, generují zprostředkující kód s názvem Common Intermediate Language. Díky této funkcí jsou dostupné rutiny v libovolném jazyce. Programátoři se tak mohou soustředit na vytváření aplikací v jejich preferovaném jazyku.
- **Kompatibilitu verzí** – Každá nová verze rozhraní .NET Framework zachovává funkce předchozích verzí a přidává nové funkce. V praxi však může být tato kompatibilita porušena zdánlivě nevýznamnými změnami v rozhraní .NET Framework a změnami v programovacích postupech.
- **Cílení na více verzí** – aplikace jsou psány pro rozhraní .NET Framework, a tím pádem fungují na různých platformách operačních systémů. Například Windows 7, Windows 8.1, atd. [3]

3.1.7 Systémové požadavky

3.1.7.1 Podporované verze operačního systému Windows

Aktuální verzi Framework 4.6 lze nainstalovat na operační systémy od Windows Vista SP2, Windows 7 SP1, Windows Server 2008, až po nejnovější Windows 10 a Windows Server 2012 R2. [5]

Podpora pro starší operační systémy Windows XP a Windows Server 2003 je pouze do verze Framework 4.0 včetně. [9]

3.1.7.2 Minimální systémové požadavky

Tabulka 3.2 zobrazuje minimální hardwarové požadavky, které jsou pro verze .NET Framework 4 až 4.6 shodné již řadu let. [9]

Tabulka 3.2: Systémové požadavky pro .NET Framework 4 – 4.6

PROCESOR	1 GHz
PAMĚŤ RAM	512 MB
MÍSTO NA PEVNÉM DISKU	850 MB (x86) až 2 GB (x64)

Zdroj: [9]

3.2 Microsoft Visual Studio

Pro vytvoření kvalitní aplikace s využitím výše popsaného .NET Framework je nezbytné používat profesionální vývojové prostředí. Společnost Microsoft jej nabízí pod názvem Visual Studio, které je rozvíjeno souběžně s postupným vývojem .NET Framework. Aktuální verze je Visual Studio 2015 a je dostupná v mnoha edicích. [10]

3.2.1 Funkce

Za pomocí Visual Studia lze již roky vytvářet například Windows Forms aplikace, ASP.NET webové aplikace, knihovny nebo Windows služby. V současné době je možné vyvíjet také mobilní aplikace na platformách Windows Mobile, Android i iOS, cloudové služby, aplikace pro Office nebo dokonce hry v prostředí Unity. [10]

Mezi aktuální trendy patří teamová spolupráce, správa životního cyklu, pokročilé metody testování a nástroje pro ladění a diagnostiku. [10]

3.2.2 Edice

Microsoft Visual Studio je distribuován ve více edicích a na více platformách. Mezi primární edice Visual Studia patří:

- **Visual Studio Community** - Bezplatný, plnohodnotný a rozšiřitelný nástroj pro vývojáře aplikací, které nejsou určené pro podniky.
- **Visual Studio Professional** - Profesionální vývojářské nástroje a služby pro jednotlivé vývojáře nebo malé týmy.
- **Visual Studio Enterprise** - Podnikové řešení s pokročilými možnostmi pro týmy pracující na projektech jakékoli velikosti nebo složitosti, včetně pokročilého testování, vývoje a provozu. [10]

Obrázek 3.9 zobrazuje podrobný přehled funkcí primárních edic.

Obrázek 3.9: Edice Visual Studio 2015

	Visual Studio Community	Visual Studio Professional	Visual Studio Enterprise	Visual Studio Test Professional	MSDN Platforms
Podporované scénáře použití	▮▮▮▯	▮▮▮▮	▮▮▮▮	▮▮▮▮	▮▮▮▮
Ladění a diagnostika	▮▮▮▯	▮▮▮▯	▮▮▮▮	▯▯▯▯	▯▯▯▯
Testovací nástroje	▮▯▯▯	▮▯▯▯	▮▮▮▮	▮▮▯▯	▮▮▯▯
Integrované vývojové prostředí	▮▮▮▯	▮▮▮▯	▮▮▮▮	▯▯▯▯	▯▯▯▯
Podpora vývojové platformy	▮▮▮▮	▮▮▮▮	▮▮▮▮	▯▯▯▯	▯▯▯▯
Architektura a modelování	▮▯▯▯	▮▯▯▯	▮▮▮▮	▯▯▯▯	▯▯▯▯
Lab Management	▯▯▯▯	▯▯▯▯	▮▮▮▮	▮▮▮▮	▮▮▮▮
Funkce Team Foundation Serveru	▯▯▯▯	▮▮▮▯	▮▮▮▮	▮▮▮▮	▮▮▮▮
Nástroje pro spolupráci	▮▮▮▮	▮▮▮▮	▮▮▮▮	▮▮▮▯	▮▮▮▯
Výhody týmové spolupráce	Součást všech předplatných				
Výhody pro předplatitele	Součást ročních cloudových předplatných a standardních předplatných				
Výhody programu Visual Studio Dev Essentials	Zdarma pro všechny vývojáře				

Zdroj: [10]

Další edice Visual Studia jsou:

- **Visual Studio Code** – Lze vytvářet a tzv. dubugovat moderní webové nebo cloudové aplikace. Nástroj je bezplatný a umožnuje využívat na platformě Windows, Linux a Mac OS X.
- Edice **Visual Studia Express** - Edice Express nabízejí bezplatné nástroje pro vývoj aplikací pro konkrétní platformy.
    - **Express for Desktop** - Umožnuje vytváření desktopových aplikací pro Windows.
    - **Express for Web** – Umožnuje tvorbu webů, které jsou založené na standardech a nabízejí výbornou odezvu, a také webová rozhraní API nebo online prostředí fungující v reálném čase pomocí technologie ASP.NET.
    - **Express for Windows** - Poskytuje základní nástroje pro vytváření atraktivních a inovativních aplikací pro univerzální platformu Windows 10.
    - **Team Foundation Server 2015 Express** - Platforma pro správu zdrojového kódu, řízení projektů a týmovou spolupráci
- **Visual Studio Test Professional** - Podpora kvality a spolupráce v celém procesu vývoje. Obsahuje pokročilé testovací nástroje a metody.
- **Visual Team Foundation Server** - Server podnikové úrovně pro týmy ke sdílení kódu, sledování práce a dodávání softwaru. [10]

3.3 Programovací jazyk C#

Jazyk C# se vyvinul z jazyka C a byl uvolněn v roce 2002 společně s .NET Framework 1.0. V současné době se používá již C# verze 6.0.

C# je programovací jazyk využívající principů objektově orientovaného programování. To je přístup, při kterém je program rozčleněn do objektů a jejich instancí. Jazyk C# je silně typovým programovacím jazykem, protože u každé proměnné nebo instance objektu je jasně definováno a vyžadováno, jakého je typu. [11]

V této podkapitole bude přiblíženo co to je jazyk C# a nejnutnější znalosti k jeho použití.

3.3.1 Objektově orientované programování

Mezi hlavní pojmy objektově orientovaného programování patří zapouzdření, dědičnost a polymorfismus.

3.3.1.1 Zapouzdření

Pomocí zapouzdření se řídí přístupnost metod a dat, řídí se tedy použití třídy. [12]

Na každý objekt lze nahlížet z vnějšího a vnitřního pohledu. Zvnějšku se pracuje s objektem pomocí jeho metod a vlastností, aniž by byla známa vnitřní implementace. [13]

Důležitým faktorem je také zabezpečení. Správně aplikované zapouzdření zabezpečuje vnitřní data objektů, aby nebyla veřejně přístupná. Mohlo by dojít k nežádoucí modifikaci dat. [12]

3.3.1.2 Dědičnost

Dědičností se rozumí možnost vytvářet objekty jako potomky jiných objektů. Tito potomci dědí jak vlastnosti, tak i metody původního objektu. Je možné některé změnit nebo k nim přidat nové. Členům třídy se musí nastavovat přístupnost a tím řídit jejich chování během dědění. [13]

Dědění je velmi dobrý způsob jak se vyhnout duplicitnímu kódu. Lze vytvářet nové třídy na základě již existujících, tak že se funkce předka mohou doplňovat nebo i překrývat. Pro vnější pohled to znamená výhodu, protože se dá k potomkům přistupovat metodami a vlastnostmi jejich předků. [12]

3.3.1.3 Polymorfismus

Polymorfismus lze popsat jako schopnost zaměnitelnosti různých tříd, které mají stejné vlastnosti, i když je implementuje každá jinak. Této schopnosti se často využívá v kombinaci s děděním, kdy potomci stejného rodiče jsou vzájemně zaměnitelní. To je výhodné v situacích, kdy se předem neví, který z potomků třídy bude v programu použit, protože to závisí například na volbě uživatele. [13]

3.3.2 Syntaxe C#

Syntaxe vychází z původního jazyka C. Jazyk C# je case sensitive tzn., že rozlišuje malá a velká písmena. [11]

Příklad 3.1 ukazuje základní prvky jazyka C#.

Příklad 3.1: Program C# – Hello, World!

```
using System;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
            // v konzoli se vypíše věta „Hello world“
        }
    }
}
```

Zdroj: autor

Pro jasné pochopení programu je nutné uvést použité znaky (viz Tabulka 3.3).

Tabulka 3.3: C# – základní syntaxe

ZNAČKA	VÝZNAM
;	Středník slouží k ukončování příkazů.
{}	Příkazy jsou shlukovány pomocí složených závorek.
//	Uvozují jednořádkové komentáře.
/* text */	Uvozují začátek a konec víceřádkových komentářů.
“text“	Textové řetězce se vkládají do uvozovek.

Zdroj: [11]

Příkaz `namespace` definuje název sestavení, ke kterému napsaný zdrojový kód patří. Příkaz `using` se definuje v případě, že se používají metody jiného `namespace`. Například je vhodné použít direktivu `using` pokud je potřeba použít `System.Console`, ale díky deklaraci `using System` stačí napsat pouze příkaz `Console`. Příkaz `class` deklaruje třídu a její obsah je uzavřen mezi složené závorky. [11]

3.3.3 Proměnné

Příklad 3.2 zobrazuje deklaraci číselné proměnné `pocet`. Dále zobrazuje inicializaci proměnné `pocet` pomocí operátoru přiřazení `=`. [11]

Příklad 3.2: Definice proměnné

```
private int cislo;
public void MojeMetoda()
{
    cislo = 15890;
}
```

Zdroj: autor

3.3.4 Modifikátory

Přístupové modifikátory implementují základní princip objektově orientovaného programování a to zapouzdření (viz kapitola 3.3.1.1).

Lze použít následující možnosti modifikátorů přístupu.:

- **Public –** přístup není nijak omezen.
- **Internal** – přístup je omezen pouze na aktuální namespace.
- **Private** – položka je přístupná jen uvnitř položky, která ji obsahuje.
- **Protected** – přístup je omezen mimo obsažené třídy nebo odvozené typy.
- **Protected internal –** kombinace modifikátorů. Přístup je omezen pouze na aktuální namespace nebo odvozené typy od obsahující třídy. [11]

Další modifikátory jsou:

- **Static** – nepracuje se s konkrétní instancí, ale statickým objektem. Objekt neumožnuje vytvářet instance.
- **New** – umožnuje skrýt zděděnou metodu se stejnou signaturou.
- **Virtual** – metodu je možné v odvozené třídě překrýt.
- **Abstract** – virtuální metoda, která definuje pouze předpis metody, ale neobsahuje její implementaci.
- **Override** – používá se k překrývání zděděné virtuální nebo abstraktní metody.
- **Extern** – využívá se pro metody implementované pomocí jiného programovacího jazyka. [11]

3.3.5 Rozhodovací příkazy

Nástroje pro řízení běhu jsou nutné pro každý programovací jazyk, který nepostupuje lineárně řádek po řádku. Nástroje se rozdělují na podmínky, cykly a příkazy skoku. [12]

3.3.5.1 Operátory

Tabulka 3.4 zobrazuje porovnávací a logické operátory jazyka C#.

Tabulka 3.4: C# – porovnávací a logické operátory

OPERÁTOR	VÝZNAM
= =	rovná se
!=	nerovná se
<	menší než
<=	menší nebo rovno
>	větší než
>=	větší nebo rovno
i++	zkrácený zápis pro i = i + 1.
i--	zkrácený zápis pro i = i - 1.
&&	logický součin (konjunkce)
||	logický součet (disjunkce)

Zdroj: [14]

3.3.5.2 Podmínky

Podmínky umožnují větvení programu podle specifikovaných kritérií. Příklad 3.3 zobrazuje základní syntaxi podmínky `if`. Příkaz provede jeden ze dvou bloků na základě platnosti podmínky. [11]

Blok `else` není povinný. V tomto případě se po neúspěšném vyhodnocení podmínky neprovede nic. Podmínky je možno do sebe vnořovat. [11; 12]

Příklad 3.3: Podmínka if

```
if (podmínka)
{…}
else
{…}
```

Zdroj: autor

Příklad 3.4 zobrazuje základní syntaxi podmínky `switch`. Příkaz `switch` se využívá pro testování proměnné a jejích hodnot. Mohla by se použít i strukturovaná soustava příkazů `if`, nicméně by byl takový kód pomalejší. Pokud nabývá proměnná hodnotu uvedenou u klíčového slova `case`, provedou se odpovídající příkazy. V případech, kdy neodpovídá žádný specifikovaný `case`, tak se provede volba `default`. [11]

Příklad 3.4: Podmínka switch

```
switch (promenna)
{
    case 1:
        příkazy; //provede se, pokud je platí promenna == 1
        break;
    case 5:
        příkazy; //provede se, pokud je platí promenna == 5
        break;
    default:
        příkazy; //provede se, pokud nebyl proveden žádný předchozí case
        break;
}
```

Zdroj: autor

Příkaz `break` znamená vyskočení z příkazu `switch` po provedení příkazů. [11]

3.3.5.3 Cykly

Další nástroje pro řízení běhu jsou cykly `while`, `for`, `do` a `foreach`. Umožňují spouštět určitou část kódu opakovaně po dobu, pokud platí definovaná podmínka. [11]

Cyklus `while` je základní iterační příkaz. Obsahuje pouze jednu hodnotu a to logický výraz. Cyklus běží, dokud platí podmínka (viz Příklad 3.5). [11]

Příklad 3.5: Cyklus while

```
while (i < 23)
{
    //příkazy
    i++;
}
```

Zdroj: autor

Příklad 3.6 zobrazuje definice cyklu `for`. První hodnota je tzv. inicializace (číslo), druhá hodnota je logický výraz a poslední je aktualizace řídící proměnné. Inicializační proměnná se nastavuje pouze při spuštění. Logický výraz se testuje při každém běhu, a pokud

je jeho hodnota `false`, ukončí cyklus svou činnost. Hodnota iterace udává, o kolik se bude řídící proměnná měnit. [11]

Příklad 3.6: Cyklus for

```
for (int i = 0; i < 20; i++)
{…}
```

Zdroj: autor

Cykly `while` a `for` testují logický výraz na začátku každí iterace. Cyklus `do` se liší tím, že provádí vyhodnocení až na konci každé iterace (viz Příklad 3.7). Této vlastnosti lze využít. I pokud neplatí podmínka, máme jistotu, že se příkazy provedou vždy minimálně jednou. [11]

Příklad 3.7: Cyklus do

```
do
{
    //příkazy
    i++;
}
while (i < 50);
```

Zdroj: autor

Další cyklus se nazývá `foreach`. Tento cyklus používá sekvenční zpracování. Příklad 3.8 ukazuje rozdělení textového řetězce na jednotlivé znaky. Využití nachází cyklus `foreach` v práci s kolekcemi. Nejprve je nutné připravit kolekci dat. Následná definice v cyklu tuto kolekci prochází po jednotlivých prvcích. [11]

Příklad 3.8: Cyklus foreach

```
foreach (char pismono in "slovo")
{
    Console.WriteLine(pismeno);
}
```

Zdroj: autor

3.3.5.4 Příkazy skoku

Příkazy skoku se využívají v rozhodovacích příkazech. Jsou to příkazy `break`, `continue` a `return`. [11]

Příkaz `break` se využívá pro vyskočení z podmínky nebo cyklu na následující řádek za příkazem. `Continue` se používá pro ukončení aktuální iterace cyklu. Cyklus pokračuje

další iterací. Příkaz `return` lze požít pro ukončení metody a může předat návratovou hodnotu volající metodě. [11]

3.3.6 Metoda

Příklad 3.9 zobrazuje vytvoření metody `Soucin` s návratovým typem `int` (musí být vždy explicitně deklarován). Tato metoda musí obsahovat příkaz `return`. Pokud by byla metoda bez návratového typu, zapíše se před název metody pouze klíčové slovo `void`. Typ `void` znamená, že se neočekává žádný návratový kód a tím pádem nemůže obsahovat příkaz `return`. Lze použít další hodnotové typy (viz 3.3.11). [11; 12]

Příklad 3.9: Metoda

```
protected void MojeMetoda()
{
    int reseni =  Soucin(150,8);
}
private int Soucin (int prvni, int druhe)
{
    return prvni * druhe;
}
```

Zdroj: autor

3.3.7 Přetěžování metod

Další zásadní pojem programování v jazyce C# je přetěžování metod. Umožňuje objektům volání jedné metody se stejným jménem, ale s jinou implementací.

Metodu lze přetížit za předpokladu stejného jména metody a definování rozdílných argumentů (viz Příklad 3.10). Po zavolání metody se spustí ta definice, která odpovídá argumentům volajícího objektu. [11]

Příklad 3.10: Přetížení metody

```
public class PretizeniMetod
{
    public void Pretizeni (string retezec) {...}
    public void Pretizeni (char znak) {...}
    public void Pretizeni (int cislo, string retezec) {...}
}
```

Zdroj: autor

Nikdy ale nelze přetížit metodu, pokud mají obě metody zvolený stejné typy atributů. Například `int x` a `int y`. [11]

3.3.8 Třída

Třída je základním pojmem klasifikace. Při vytváření třídy se systematicky uspořádají informace a chování do smysluplné entity. Příklad 3.1 již zobrazil použití třídy ve zdrojovém kódu. U tříd lze také jako u metod využívat modifikátory. Nejčastěji se jedná o modifikátory `public` a `static`. [12]

3.3.9 Konstruktor

Konstruktor se definuje jako metoda, musí mít stejný název jako třída a nesmí obsahovat žádný návratový typ (viz Příklad 3.12). Konstruktor je použit vždy, když je vytvořena nová instalace třídy. [11; 12]

Příklad 3.11: Konstruktor

```
public class TridaSKonstruktorem
{
    public TridaSKonstruktorem ()
    {…}
}
```

Zdroj: autor

Příklad 3.12 zobrazuje, že je možné konstruktor také přetížit pomocí definice atributů. Při vytvoření objektu vzniká instalace s požadovanou implementací. [12]

Příklad 3.12: Přetížený konstruktor

```
public class PretizenyKonstruktor
{
    public PretizenyKonstruktor ()
    {…}
    public PretizenyKonstruktor (string retezec)
    {…}
}
```

Zdroj: autor

3.3.10 Ošetřování výjimek

Každý aplikace musí být schopna detekovat chyby a efektivním způsobem je ošetřit. Neošetřené výjimky mají totiž za následek nestabilní aplikaci. V jazyce C# se k tomu využívají příkazy `try`, `catch` a `finally`. Příklad 3.13 zobrazuje použití příkazů. [11]

Příklad 3.13: Ošetřování chyb – try, catch a finally

```
        private void Metoda()
        {
            try
            {
                //provádí příkazy
            }
            catch (FormatException exFormat)
            {
                //zde bude zpracována výjimka typu FormatException
                Console.WriteLine(exFormat.Message);
            }
            Catch (Exception ex)
            {
                //zde bude zpracována případná výjimka
                 Console.WriteLine(ex.Message);
            }
            finally
            {
                //Spustí kód vždy i při výskytu výjimky
            }
         }
```

Zdroj: autor

Příkaz `catch` je možné používat vícenásobně. Lze použít k zachytávání konkrétních výjimek, pro které provede konkrétní ošetření. [12]

Při programování se často stává, že má metoda vracet hodnoty v určitém rozsahu. Pokud se tak nestane, je nutné tuto situaci patřičně ošetřit. Při znalosti používaní výjimek v jazyce C# lze vyvolat výjimku pomocí příkazu `throw` (viz Příklad 3.14). [12]

Příklad 3.14: Vyvolání výjimky

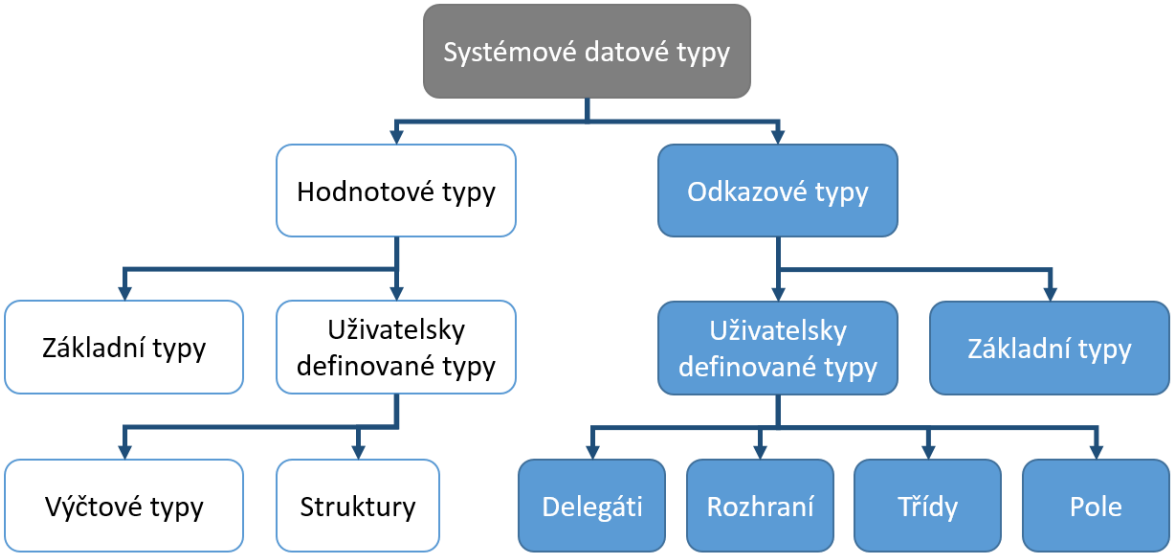
```
        public static string NazevDne(int den)
        {
            switch (den)
            {
                case 1:
                    return "Pondělí";
                ...
                case 2:
                    return "Úterý";
                ...
                case 7:
                    return "Neděle";
                default:
                    throw new ArgumentOutOfRangeException("Chybné číslo dne");
            }
         }
```

Zdroj: autor

3.3.11 Hodnotové a referenční typy

Jelikož může každý programátor používat jiný programovací jazyk v prostředí .NET Framework, musí jednotlivé jazyky využívat shodné datové typy. Toto zajišťuje tzv. CTS (Common Type System) neboli společný typový systém, který je součástí již zmiňovaného CLR. Mezi hlavní úkoly CTS patří integrace zdrojového kódu z různých jazyků, zajištění typové bezpečnosti a realizace objektově orientovaného modelu. Společný typový systém rozděluje datové typy do dvou základních skupin na hodnotové a odkazové. Tyto kategorie se dále dělí (viz Obrázek 3.10). [15]

Obrázek 3.10: Typový systém .NET



Zdroj: [15]

U proměnných hodnotového typu obsahují proměnné přímo data, která jsou alokována na stacku nebo v datové struktuře. Příkladem jsou typy `bool`, `int`, `double` nebo `char`. Uživatelsky definované hodnotové typy lze tvořit například pomocí `enum`. [12; 15]

U proměnných odkazového typu neobsahují proměnné přímo hodnotu, ale obsahují referenci na hodnotu. Tento odkaz je umístěn také na stacku nebo v datové struktuře, ale hodnota je umístěna na heapu. Příkladem jsou typy `string` a `object`. [15]

3.3.12 Pole a kolekce

Pole je nespořádaná posloupnost prvků. Všechny prvky v poli jsou stejného typu (hodnotový typ). K prvkům se přistupuje pomocí celočíselného indexu. [12]

Kolekce odstraňují nutnost přistupovat k prvkům pomocí celočíselného indexu jako u pole. Druhá výhoda je, že nemusí obsahovat pouze hodnotové typy, ale také referenční (objekty). Existuje třída kolekci v direktivě `System.Collections`. [12]

Příkladem kolekcí je například:

- **Fronta** (`Queue`), která implementuje mechanismus FIFO (First In - First Out). Prvky jsou zařazovány na konec fronty metodou `Enqueue`, odebírány ze začátku metodou `Dequeue`.
- **Zásobník** (`Stack`), který implementuje mechanismus LIFO (Last In - First Out). Prvek je vložen na vrchol zásobníku metodou `Push` a odebírán také ze začátku metodou `Pop`. [12]

4 Problematika správy software ve firmě

4.1 Analýza produktivního procesu

Analýza bude prováděna na základě uživatelských zkušeností z firmy Škoda Auto a.s, konkrétně z oddělení zabývajícího se systémovou integrací a podporou konstrukčních a datových systémů. Tyto aktivity zajišťuje včetně externího personálu řádově několik desítek zaměstnanců. Výstupy z analýzy procesu bude možné vztáhnout obecně na menší až středně velké firmy, pro které by měl být nově vytvořený systém řešením.

4.1.1 Správa software

Za správu aplikace je zodpovědný správce aplikace. Správu lze tematicky rozdělit na instalaci, aktualizaci, instalační zdroje, přístup ke vzdálené stanici a evidenci software.

4.1.1.1 Instalace

Na základě požadavku (telefonicky, e-mailem, change request) spustí administrátor instalaci obvykle pomocí vytvořeného skriptu. Spuštění probíhá pomocí příkazové řádky operačního systému Windows (CMD). Tyto skripty jsou napsané ve skriptovacím jazyce Batch nebo Perl. Každý správce používá vlastní skripty osahující kontrolu základních podmínek, než dojde ke spuštění koncového instalačního programu nebo skriptu od dodavatele. U minoritních aplikací provede administrátor instalaci zcela manuálně pomocí nástroje vzdálené plochy.

Po dokončení instalace provede manuálně zápis do vlastní evidence.

4.1.1.2 Aktualizace

Pro hromadnou aktualizaci nejrůznějších druhů software se nyní používají skripty. Tyto skripty jsou analogicky jako u instalace napsané ve skriptovacím jazyce Batch nebo Perl.

Tyto skripty, běžící z příkazové řádky, zajišťují hromadnou aktualizaci pomocí rotace načteného pole stanic z předem připraveného textového souboru. Skripty vyhodnocují, zda je vzdálená stanice spuštěna a zda existuje přístup pro uživatele s administrátorskými právy. Pokud jsou vstupní podmínky splněny, dojde ke spuštění koncového instalačního programu nebo skriptu od dodavatele.

V průběhu aktualizace se vytváří textový soubor s úspěšně aktualizovanými stanicemi. Pro další spuštění doposud neaktualizovaných stanic musí administrátor manuálně vytvořit nový instalační seznam a znovu spustit skript pomocí příkazové řádky. Seznam definuje jako deltu všech stanic z původního souboru s již úspěšně aktualizovanými stanicemi.

Taktéž u minoritních aplikací provádí administrátor aktualizaci zcela manuálně pomocí vzdálené plochy.

Případné problémy s instalací musí vyčíst administrátor z výpisu příkazové řádky, případně logů, pokud si je ve vlastním skriptu vytváří.

4.1.2 Instalační zdroje

Administrátoři spravují instalační zdroje obvykle na sítových discích nebo na vlastních virtuálních serverech. Jedná se sice o zálohovaná úložiště, ale nikoli centralizovaná v rámci firmy.

4.1.3 Přístup ke vzdálené stanici

Pro systémový přístup ke vzdáleným stanicím se využívá nástroj PsExec obsažený v balíčku nástrojů Windows Sysinternals. [16]

Pří používání nástroje PsExec dochází relativně často k narušení stávajícího procesu a způsobuje další manuální práci administrátora.

Například velmi častý problém nastane, pokud se ve firmě používají rozdílné verze nástroje PsExec. V tomto případě je nutné jiným způsobem vynutit ukončení běžící služby psexesvc.exe na vzdálené stanici a znovu spustit skript.

Mezi další obtíže patří nahodilé vracení obvykle nesmyslných chybových kódů. Řešením bývá pouze restart cílové stanice, což vede k nepatřičnému přerušení práce uživatele.

4.1.4 Evidence software

Evidence počítačů, které obsahují nainstalovaný software nebo mají být aktualizovány, se provádí manuálním zápisem do vlastní evidence. Evidence je převážně spravována v aplikaci Microsoft Excel případně Microsoft Access. Takovou evidenci si

udržuje každý administrátor nebo skupina administrátorů software. Evidence jednotlivých verzí software se zpravidla neprovádí.

V případech auditu nebo softwarové inventury je velmi složité zjistit, zda firma používá správný počet licencí. Pro tyto účely se musí počítače provozním oddělením pravidelně skenovat.

4.2 Zhodnocení produktivního procesu

Z výše popsaného procesu jasně vyplývá mnoho nesystémových a duplicitních aktivit. Jelikož není proces standardizovaný, nevzniká zde ani možnost zastupitelnosti jednotlivých administrátorů a vedení to neumožnuje pružně plnit nové úkoly, případně dodržovat tzv. SLA (Service Level Agreement).

Na základě provedené analýzy a zkušeností s výše popsaným procesem vyplývá několik možných kroků k optimalizaci celkového procesu, stability instalací, úspoře času administrátorů a zajištění požadavků na centrální evidenci.

4.3 Identifikované požadavky

Mezi identifikované požadavky patří:

- Jednotná evidence počítačů.
- Jednotná evidence aplikací.
- Jednotná evidence aplikací instalovaných na počítače.
- Možnost vyhledávání v evidenci.
- Přehledné a intuitivní webové rozhraní.
- Rozhraní plnohodnotně dostupné bez ohledu na koncové zařízení.
- Rozhraní umožňující řešit neúspěšné instalace.
- Náhrada duplicitních skriptů zajištujících základní přístupové kontroly
- Potřeba zajištění automatického zpracování požadavků.
- Potřeba zajištění automatické aktualizace evidence.
- Odstranění duplicitních činností administrátorů při správě software a hardware.
- Centrální správa instalačních zdrojů.
- Nalezení stabilní metody přístupu ke vzdáleným stanicím.

- Řešení využitelné pro počítače s operačním systémem Windows 7, Windows 8.1 a Windows 10.

4.4 Návrh nového procesu

4.4.1 Optimalizovaný proces

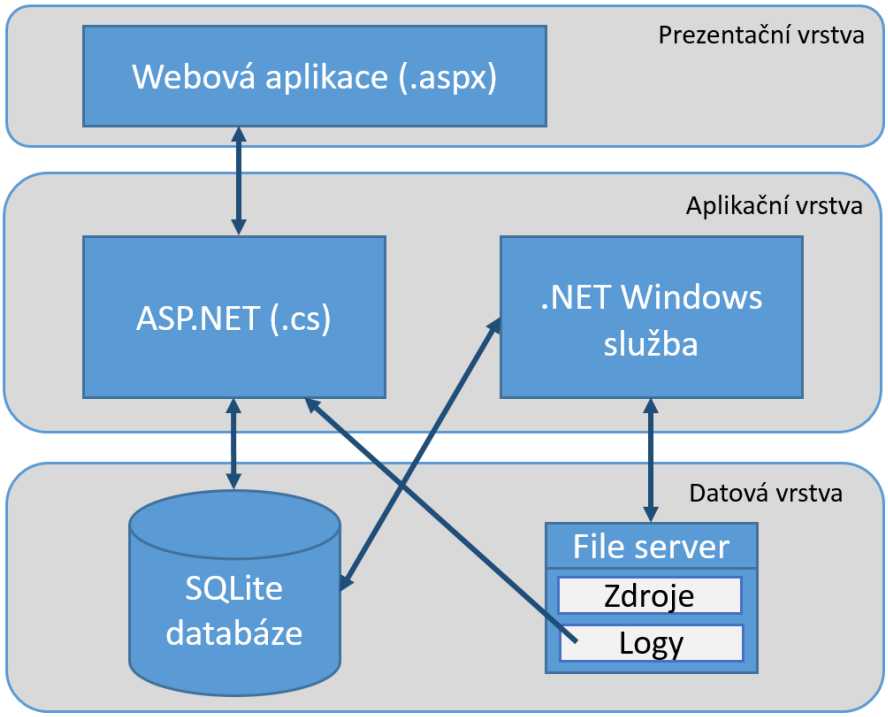
Optimalizovaný proces centrální správy software je navržen pomocí vícevrstvé architektury a zahrnuje:

- Přehledné rozhraní na platformě ASP.NET, které bude na jedné straně zajišťovat komunikaci s databází, na druhé straně připravovat prezentaci pro uživatele systému. Bude splňovat nároky na responzivní design pro plnohodnotné zobrazení na jakémkoliv typu zařízení.
- Centrální databázi obsahující veškeré evidované aplikace, spravované počítače a také nezbytné tabulky obsahující instalovaný software na jednotlivých stanicích a tabulku s požadavky a jejich stavy. Pro tuto práci byla zvolena systémově nenáročná databáze SQLite.
- Službu systému Windows, která zajistí komunikaci s databází, rotaci požadavků, základní kontroly přístupnosti stanic a jejich konečné spuštění na vzdálených stanicích pomocí služeb WMI. Nezbytným požadavkem pro zajištění robustního škálovatelného řešení je použití více vláknové technologie.
- Centralizované úložiště zdrojových dat, aplikačních skriptů a instalačních logů. Realizováno pomocí souborové struktury na zálohovaném File serveru.

Takto navržený proces zcela zohledňuje identifikované požadavky z analýzy současného produktivního procesu.

Obrázek 4.1 zobrazuje vícevrstvý návrh systému, umístění jednotlivých komponent a jejich vzájemné propojení.

Obrázek 4.1: Vícevrstvý návrh systému



Zdroj: autor

Uživatel přichází ke styku pouze s rozhraním ve webovém prohlížeči na prezentační vrstvě. Výkonná část webové aplikace je umístěna na webovém serveru v aplikační vrstvě. Z této vrstvy přistupuje aplikace do databáze za účelem čtení uložených dat a tvorbě požadavků. Webová aplikace dokáže zobrazovat logy z File serveru.

Služba vůbec nezasahuje do prezentační vrstvy a její funkce je uživateli skryta. Služba komunikuje s databází za účelem čtení požadavků, úpravě jejich stavu a zapisování nových záznamů. Služba instaluje aplikace ze zdrojových souborů umístěných na File serveru, kam také zapisuje veškeré aplikační logy. Logy o běhu samotné služby zůstávají dostupné pouze administrátorovi systému v aplikační vrstvě.

4.4.2 Uživatelské scénáře

- **Instalace** – uživatel vybere kombinaci z dostupné nabídky evidovaného software a hardware. Systém zajistí spuštěni instalačního skriptu příslušné aplikace na příslušné stanici. Po úspěšném dokončení provede zápis do evidence záznamů.

- **Aktualizace** – uživatel vybere druh aktualizace a provede specifikaci z dostupné nabídky evidovaných záznamů. Systém zajistí spuštěni aktualizačního skriptu příslušných aplikací na příslušných stanicích.
- **Odinstalace** – uživatel vybere kombinaci z dostupné nabídky evidovaných záznamů. Systém zajistí spuštění odinstalačního skriptu příslušné aplikace na příslušné stanici. Po úspěšném dokončení provede vymazání z evidence záznamů.

4.4.3 Uživatelské role

V novém systému lze specifikovat uživatelské role a jejich účel.

- **Uživatel** – Jedná se o administrátora spravované aplikace. Má přístup na webové rozhraní, kde může zadávat nové instalace, aktualizace, odinstalace nebo nahlížet do evidence. Dále má dostupné rozhraní pro řešení nepovedených instalací. Na File serveru má přidělen přístup do složek, které náleží do jeho kompetence.
- **Administrátor** – spravuje běh celého systému, a to především dostupnost webové aplikace, databáze a služby. K tomu má dostupné rozhraní pro webový server (IIS), rozhraní DB Browser for SQLite pro přístup do databáze, záznamy o běhu služby a možnost konfigurace spuštění. Také řídí přidělování oprávnění na File serveru. Role může být ve velkých firmách rozdělena do více funkčních celků.

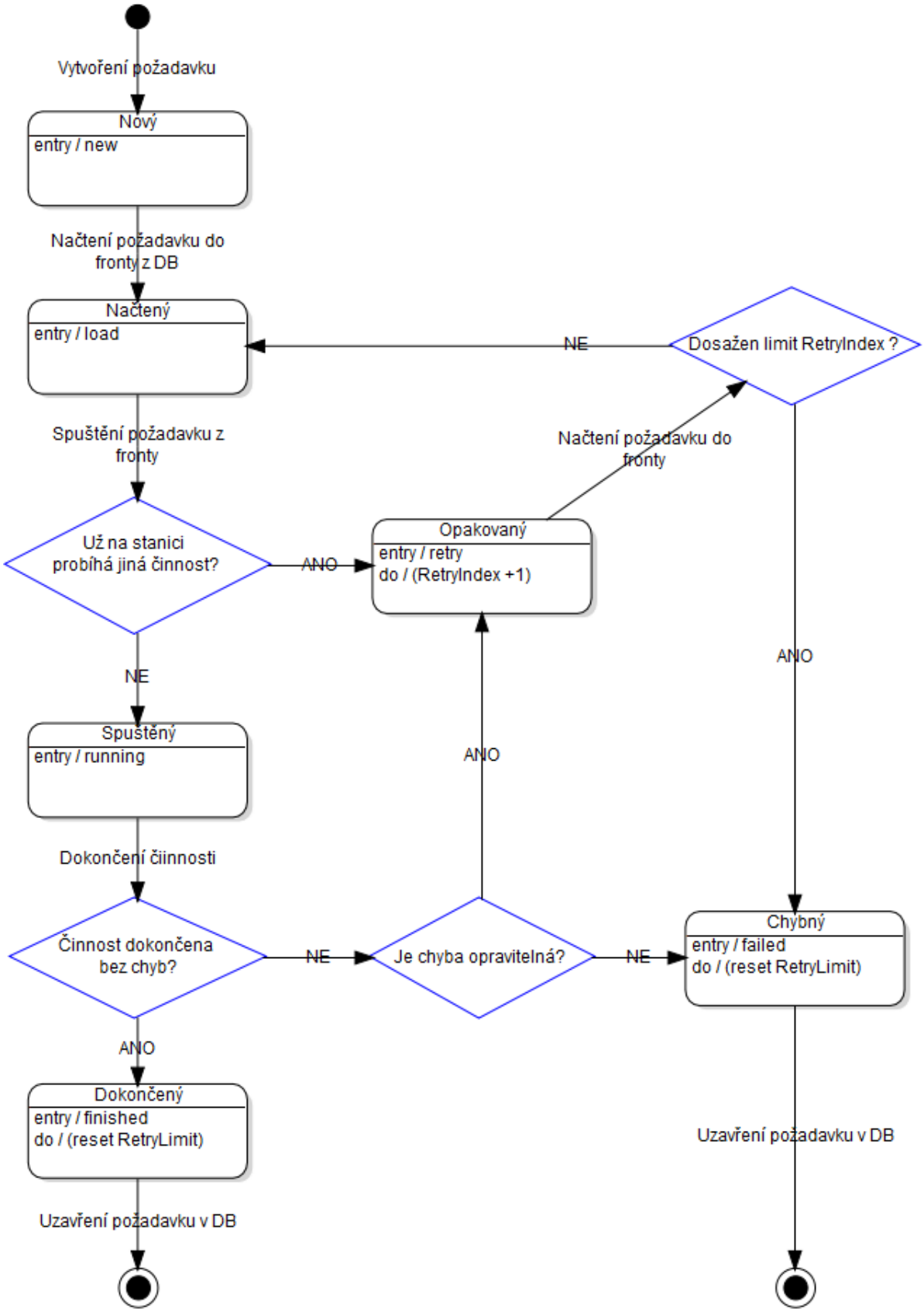
4.4.4 Požadavky

Jakýkoliv podnět administrátora (instalace, aktualizace) je brán v systému jako požadavek. Tyto požadavky jsou složeny z položek: software, hardware, stav a operace.

Požadavek může nabývat pouze stavy new, retry, load, running, finished a failed. Pro přehledné algoritmické vyjádření změny stavu požadavku je vytvořen tzv. Stavový diagram (viz Obrázek 4.2).

Na obrázku je zmiňován tzv. RetryIndex. Tato vlastnost je potřebná, aby nezůstávaly v systému donekonečna rotovat nevyřešitelné požadavky. V implementaci systému bude nastavena hodnota maximálního počtu iterací (viz kapitola 5.3.2.1).

Obrázek 4.2: Stavový diagram – změny stavu požadavku



Zdroj: autor

5 Vlastní řešení

Vlastní řešení jednotlivých komponent lze rozdělit na návrh vnitřní funkcionality, posléze její implementaci a na závěr důkladné otestování vzniklého prototypu systému.

5.1 Databáze

SQLite je relační databázový systém, který obsahuje transakční SQL databázový stroj. Na rozdíl od ostatních SQL databází, využívajících princip klient-server, kde je databázový server spuštěn jako samostatný proces, je SQLite pouze knihovna, která se přilinkuje k aplikaci a pomocí jednoduchého rozhraní ji lze začít využívat. Jedná se o soubory s příponou sqlite (například mojedatabaze.sqlite). Kompletní SQL databáze s tabulkami, triggery a pohledy, je obsažena v jednom souboru na pevném disku. SQLite je velice kompaktní knihovna, velikost knihovny může být menší než 500 KiB. [17]

Formát souboru databáze je multiplatformní tzn., že lze databázi libovolně kopírovat mezi 32bitovými a 64bitovými systémy. [17]

SQLite je šířena s licencí public domain a je tak zdarma k použití jak pro soukromé tak i komerční účely. [18]

Díky těmto vlastnostem je SQLite velmi populární volba pro využití jako jednoduché a systémově nenáročné databáze použitelné jak na počítačích (nezávisle na platformě), tak i na tabletech a v mobilních telefonech. [17]

5.1.1 Návrh

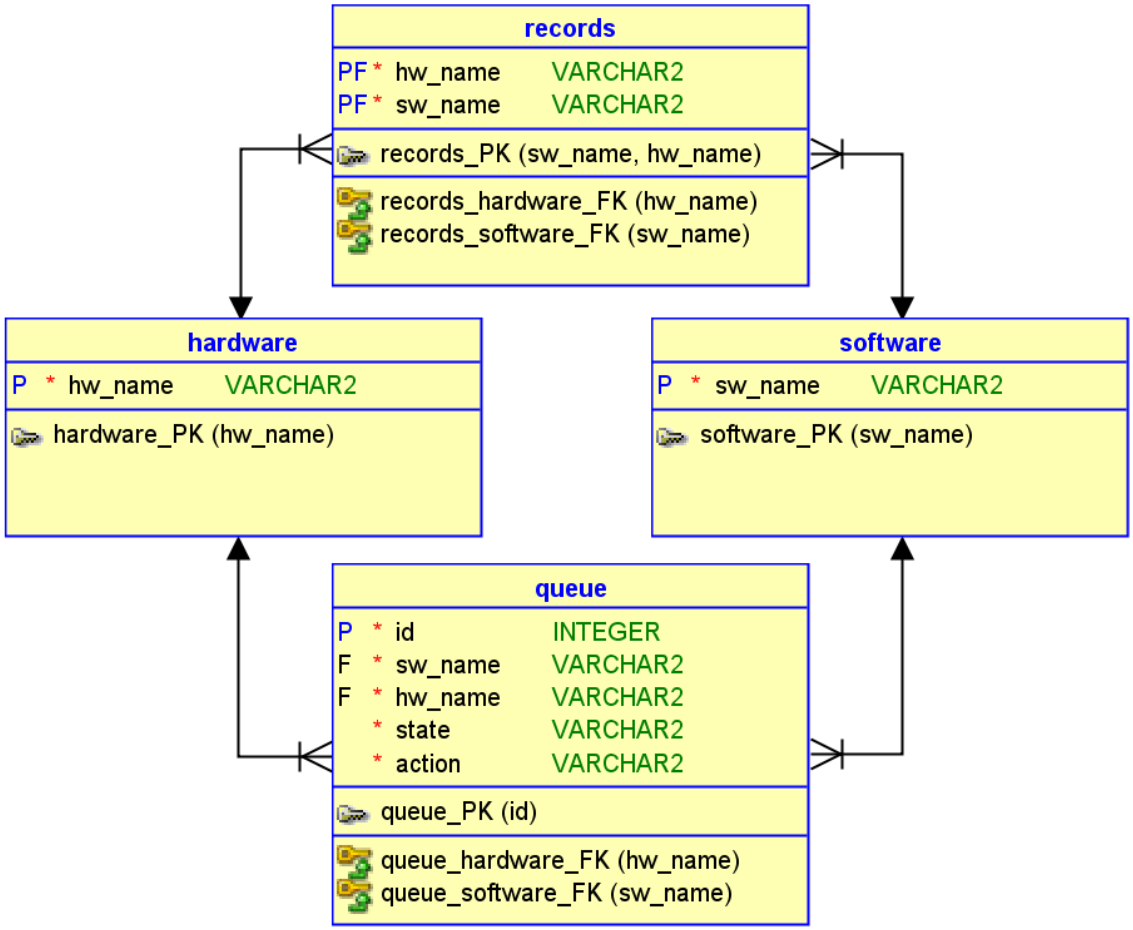
V návrhu relačního datového modelu databáze (viz Obrázek 5.1) jsou použity vstupní tabulky software a hardware. Tyto tabulky obsahují pouze atomické hodnoty a to kolekce názvu software a kolekce názvu hardware. Tyto atributy také i primárními klíči.

Další tabulka slouží pro evidenci konkrétního software na konkrétní hardware. Je reprezentována jako vazební množina nazvaná records. Primárním klíčem tabulky je unikátní kombinace cizích klíčů názvu software a názvu hardware.

Poslední tabulka je také vazební a jmenuje se queue. Slouží pro evidenci požadavků, jejich stavů a akcí. Primárním klíčem je sloupec s identifikátorem. Tabulka obsahuje cizí klíče a to název software a název hardware.

Tabulka může nabývat stavy podle specifikace v kapitole 4.4.4. Dle předem definovaných uživatelských scénářů v kapitole 4.4.2, je možné použít akce install, update a delete.

Obrázek 5.1: Datový model databáze



Zdroj: autor

Databáze byla normalizována dle 3. normální formy.

5.1.1.1 Nástroj na správu databáze

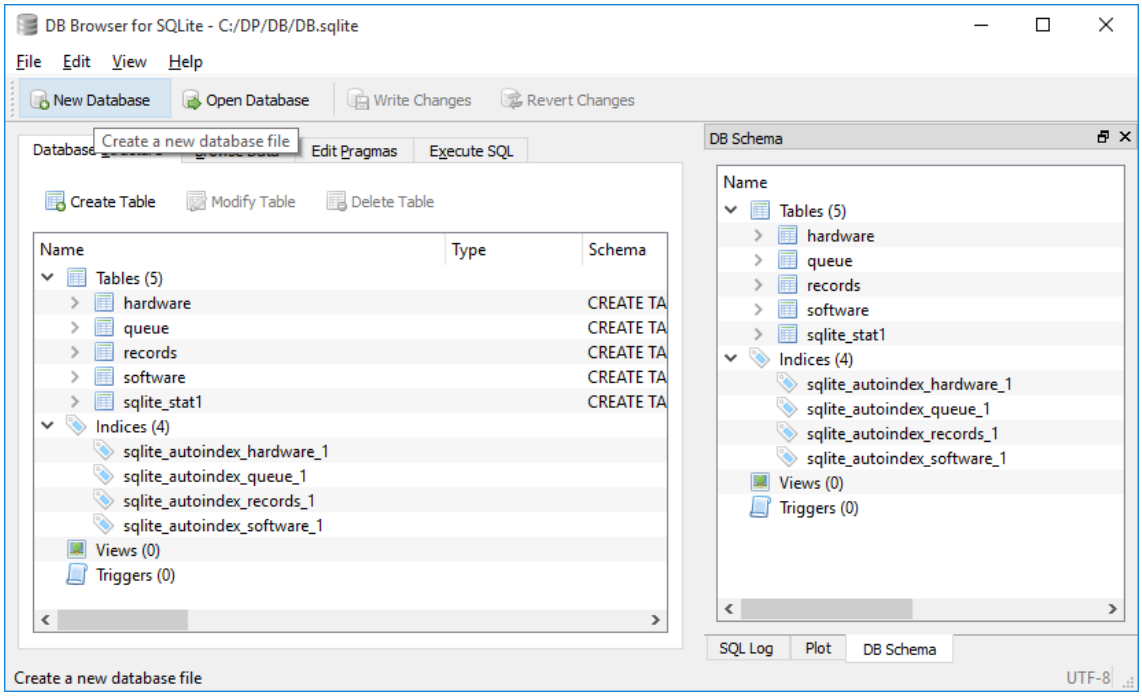
Pro vytvoření a správu databáze je vhodné používat nástroj, ačkoliv databáze SQLite umožnuje ovládání z příkazové řádky.

DB Browser for SQLite je open source nástroj s grafickým rozhraním pro vytváření a editaci databází kompatibilních s SQLite. Hlavní výhodou tohoto nástroje, je umožnit i netechnickým uživatelům vytvářet, upravovat a nastavovat databázi pomocí nástroje s přehledným rozhraním. [19]

Pomocí nástroje DB Browser for SQLite je možné:

- Vytvořit databázi (viz Obrázek 5.2).
- Vytvořit, upravit a odstranit tabulky.
- Procházet, editovat, přidávat a odstraňovat záznamy.
- Vyhledávat seznamy, vytvářet pohledy.
- Importovat a exportovat záznamy.
- Importovat a exportovat tabulky z/do CSV souborů.
- Importovat a exportovat databáze z/do SQL dump souborů.
- Provádět SQL dotazy and procházet výsledky.
- Procházet logy všech SQL příkazů provedených aplikací. [19]

Obrázek 5.2: DB Browser for SQLite



Zdroj: autor

5.1.1.2 Přístup do databáze z Visual Studia

Pro zpřístupnění funkcí SQLite databáze ve vývojovém prostředí Visual Studio je nutné stáhnout speciální knihovnu, která pomocí jednoduchých metod zajistí nezbytnou komunikaci. Toho lze docílit přímo z menu prostředí Visual Studia – Tools → NuGet Package Manager → Manage NuGet Packages for Solution… Po zobrazení stačí vyhledat klíčové slovo sqlite a zvolit knihovnu System.Data.SQLite. [20]

System.Data.SQLite je oficiální ADO.NET provider, Je kompatibilní, jak s 32bitovou, tak i 64bitovou verzí operačního systému. [20]

5.1.1 Implementace

Další krok vzniku databáze je vytvoření jednotlivých tabulek, a nadefinování jejich vlastností. K tomu byl použit SQL příkaz `CREATE TABLE`.

Tabulka software obsahuje sloupec `sw_name` (viz Příklad 5.1). Položky tohoto sloupce mohou nabývat libovolný řetězec, který musí být vždy neprázdný a také musí být unikátní. Jedná se taktéž o primární klíč tabulky.

Příklad 5.1: Vytvoření tabulky software

```
CREATE TABLE software
(
sw_name VARCHAR PRIMARY KEY  NOT NULL  UNIQUE
);
```

Zdroj: autor

Tabulka hardware obsahuje, analogicky jako tabulka software, sloupec `hw_name` (viz Příklad 5.2). Položky tohoto sloupce mohou nabývat libovolný řetězec, který musí být vždy neprázdný a také musí být unikátní. Jedná se taktéž o primární klíč tabulky.

Příklad 5.2: Vytvoření tabulky hardware

```
CREATE TABLE hardware
(
hw_name VARCHAR PRIMARY KEY  NOT NULL  UNIQUE
);
```

Zdroj: autor

Tabulka records obsahuje sloupce `sw_name` a `hw_name` (viz Příklad 5.3). Položky tohoto sloupce musí být vždy neprázdné. Jedná se o cizí klíče z tabulek software (`sw_name`) a hardware (`hw_name`). Jejich vzájemná kombinace tvoří primární klíč této tabulky.

Příklad 5.3: Vytvoření tabulky records

```
CREATE TABLE records
(
sw_name VARCHAR NOT NULL,
hw_name  VARCHAR NOT NULL,
FOREIGN KEY(sw_name) REFERENCES software(sw_name),
FOREIGN KEY(hw_name) REFERENCES hardware(hw_name),
PRIMARY KEY (sw_name, hw_name)
);
```

Zdroj: autor

Poslední tabulka queue obsahuje sloupce `id`, `sw_name`, `hw_name`, `state`, `action` (viz Příklad 5.4). Veškeré položky musí být nenulové. Sloupec `id` je primární klíč tvořený indexem, který automaticky roste při zápisu nového záznamu. Je tedy unikátní. Položky `sw_name` a `hw_name` jsou opět cizí klíče z tabulek software (`sw_name`) a hardware (`hw_name`). Zbývající sloupce `action` a `state` musí obsahovat předem specifikovaný řetězec.

Příklad 5.4: Vytvoření tabulky queue

```
CREATE TABLE queue
(
id INTEGER NOT NULL UNIQUE ,
sw_name VARCHAR NOT NULL,
hw_name  VARCHAR NOT NULL,
state VARCHAR NOT NULL ,
action VARCHAR NOT NULL,
FOREIGN KEY(sw_name) REFERENCES software(sw_name),
FOREIGN KEY(hw_name) REFERENCES hardware(hw_name),
PRIMARY KEY (id)
);
```

Zdroj: autor

Přidávání záznamů lze provádět pomocí výše popsaného grafického rozhraní, nebo pomocí příkazu `INSERT INTO`. Příklad 5.5 zobrazuje přidání třech nových záznamů do tabulky records.

Příklad 5.5: Vložení záznamů do tabulky

```
INSERT INTO "records" VALUES('KVS','SKDATEST9997');
INSERT INTO "records" VALUES('KVS','SKDATEST9998');
INSERT INTO "records" VALUES('KVS','SKDATEST9999');
```

Zdroj: autor

5.2 Webová aplikace

Webová aplikace je jediné grafické uživatelské rozhraní celého systému. Tvorbu ve fázi návrhu lze rozdělit na představení použitých technologií a vytvoření logického a grafického designu. Následně je zpracována implementace aplikace.

5.2.1 Návrh

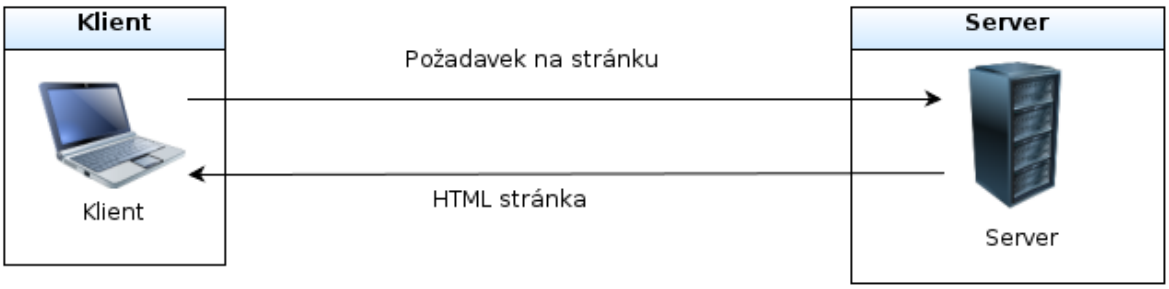
5.2.1.1 ASP.NET

ASP.NET je webový Framework, který je tvořen sadou knihoven, umožňujících tvorbu webových aplikací. Knihovny obsahují již hotová řešení mnoha základních problémů, které se ve webových technologiích vyskytují. ASP.NET je pro upřesnění jedna ze součástí .NET Frameworku (viz kapitola 3.1). [14]

Technologie je založena na architektuře klient-server. ASP.NET aplikace běží na straně serveru. Výstupem ASP.NET aplikace je vygenerovaná HTML stránka. Pro lepší pochopení je zde uveden rozdíl mezi klasickým statickým webem a dynamickým webem (ASP.NET) [14]:

- **Statický web** – Pokud uživatel požaduje stránku, server mu odešle předem uloženou stránku (viz Obrázek 5.3). HTML stránky jsou uložené na serveru. [21]

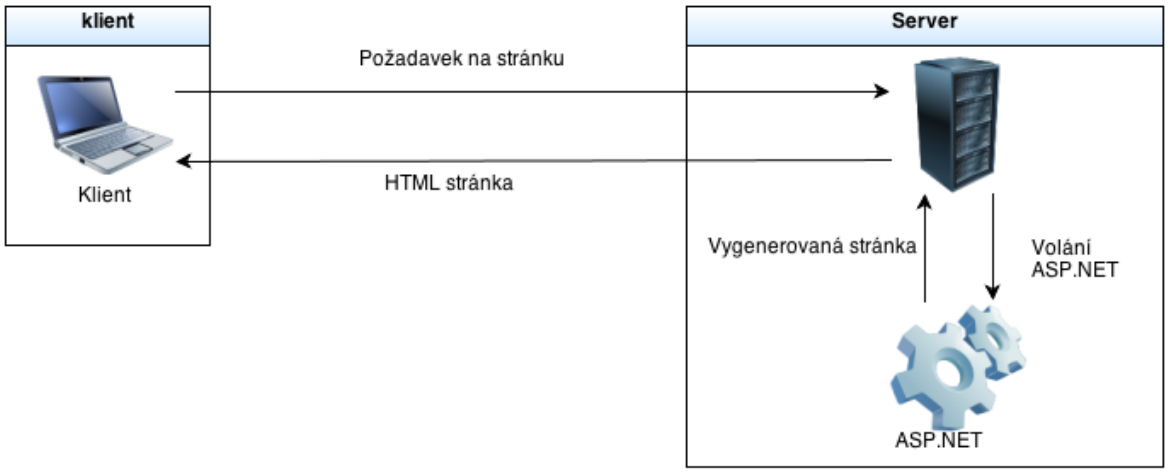
Obrázek 5.3: Statický web



Zdroj: [21]

- **Dynamický web** – Na základě požadavku uživatele vygeneruje aplikace webovou stránku a odešle ji uživateli. Uživatel dostane výsledný HTML soubor (viz Obrázek 5.4). Poznávacím znakem je pouze suffix názvu stránky – namísto .html je zkratka .aspx. [21]

Obrázek 5.4: Dynamický web



Zdroj: [21]

Pomocí ASP.NET je možné vytvářet, jak malé osobní weby, tak i velké korporátní projekty s použitím technologií HTML5, CSS3 a JavaScript. Aplikace lze tvořit pomocí Microsoft Visual Studia (viz kapitola 3.2) a jazyku C# (viz kapitola 3.3). ASP.NET aplikace běží na serveru webovém IIS neboli Internet Information Services. [14]

Ačkoliv je webový protokol HTTP tzv. bezstavový, událostmi řízené programování vyžaduje zachování stavu. Jedná se o zachování kontextu mezi jednotlivými požadavky. ASP.NET tento problém řeší kombinací HTML a JavaScriptu pomocí dvou technik:

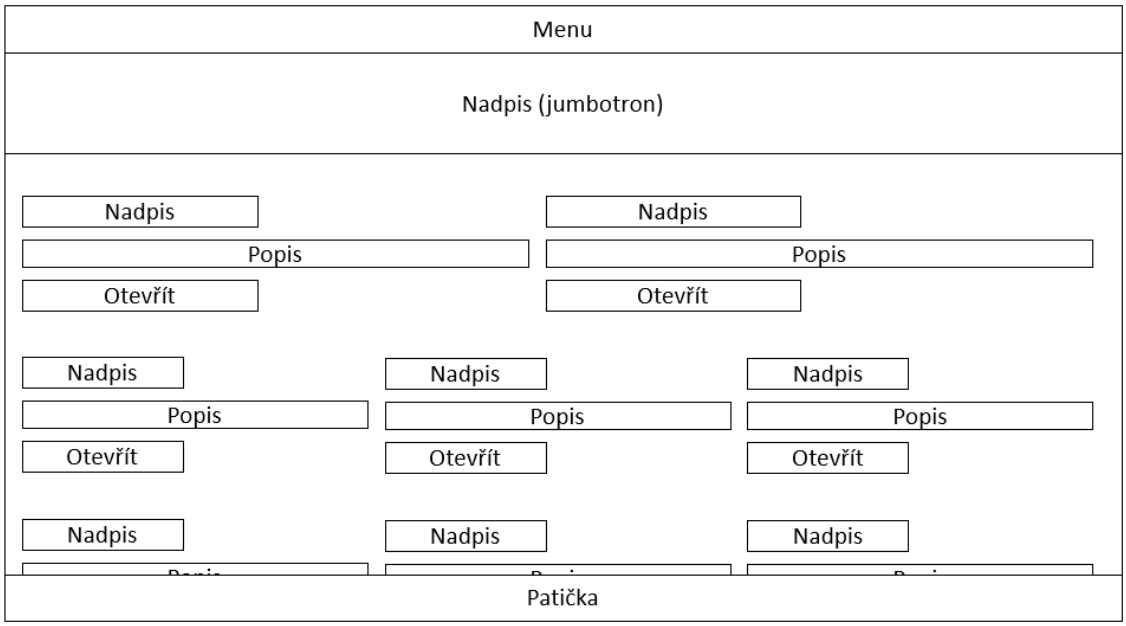
- **ViewState** – Uchovává informace, mezi opakovanými odesíláními formuláře na server tzv. postbacky, v zakódovaném tvaru ve skrytých formulářových polích. Výhodou je, že se využívá pouze HTML a nevyžadují se další zpracování na straně serveru nebo klienta. Nevýhodou je, že se mezi serverem a klientem přenáší větší objem dat, především pokud je ViewState využíván nesprávně.
- **SessionState** – Ukládá veškeré informace na straně serveru a předává pouze jednoznačný identifikátor. Tento přístup zmenšuje objem přenášených dat, ale vytváří vyšší nároky na výkon serveru. V případě, že se SessionState používají nesprávně, může být server náchylný i k DoS (Denial of Service) útokům. [14]

5.2.1.2 Logický a grafický design

Logický a grafický design byl navržen tak, aby byl co nejvíce přehledný a snadno ovladatelný. Systém je možné ovládat také pouze pomocí klávesnice. Vytvořený prototyp systému dostal název Management of Software. Použitá barevná paleta a grafický návrh byly inspirovány aktuální verzí Windows 10. Webová aplikace obsahuje stránky:

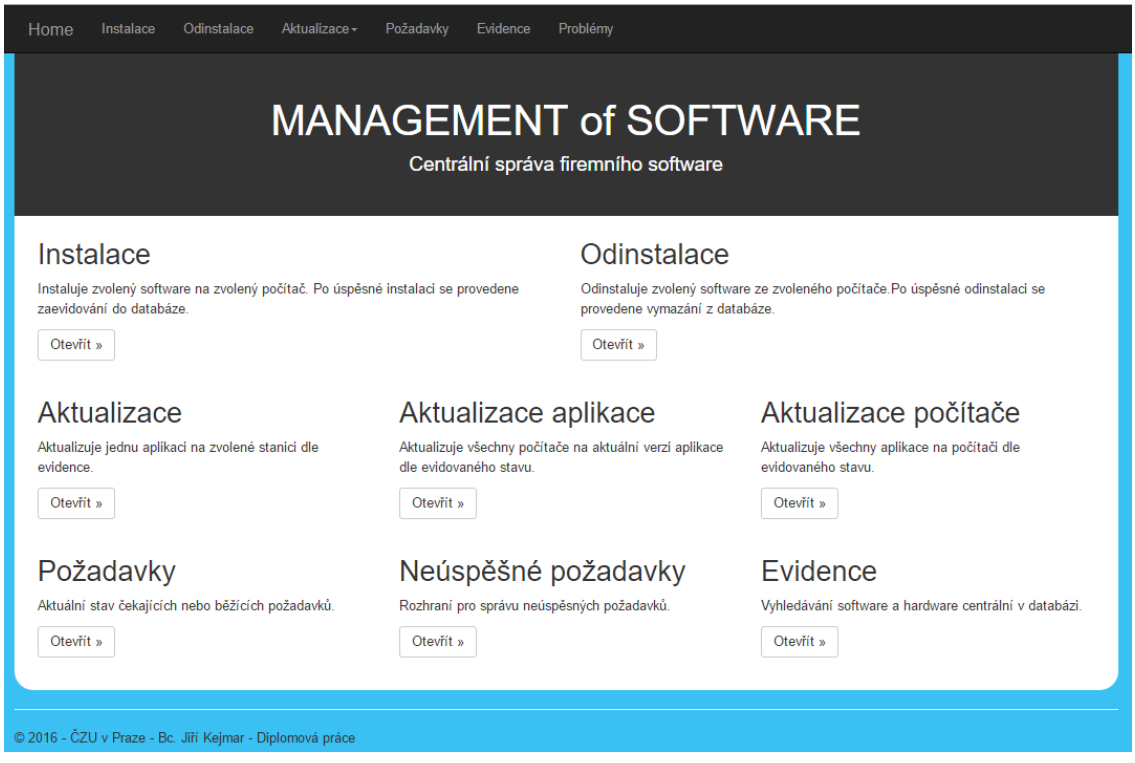
- **Homepage** – Po spuštění se zobrazí prostředí webové aplikace (viz Obrázek 5.5 a Obrázek 5.6). Na úvodní stránce je dostupný přehled a popis funkcí systému s přímými odkazy. Horní část všech stránek je vyhrazena navigaci.

Obrázek 5.5: Logický design – Homepage



Zdroj: autor

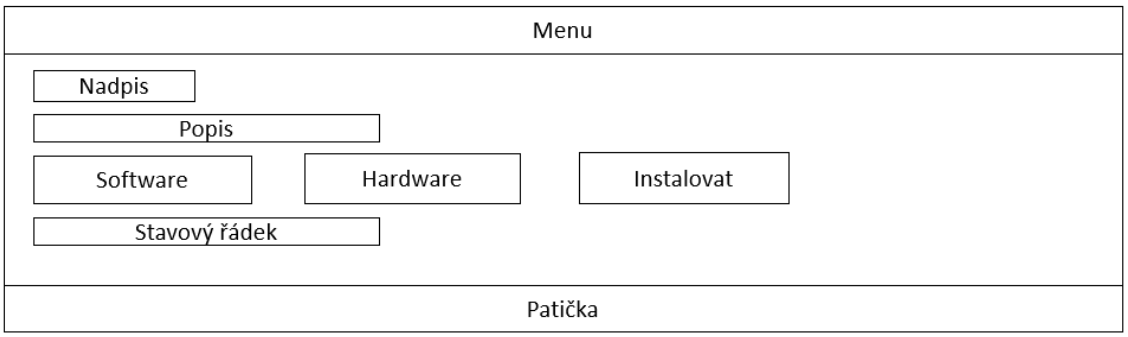
Obrázek 5.6: Grafický design – Homepage



Zdroj: autor

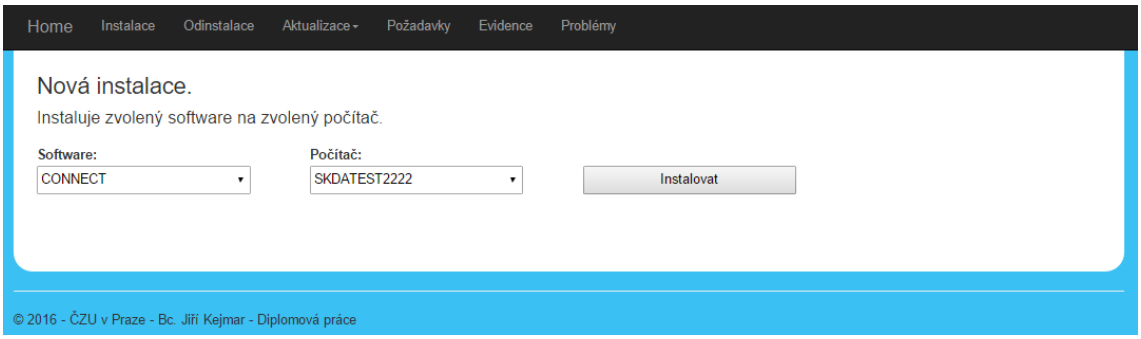
- **Instalace** – Stránka obsahuje dva rozbalovací seznamy a tlačítko (viz Obrázek 5.7 a Obrázek 5.8). Po vybrání konkrétní aplikace a počítače dojde, po zmáčknutí tlačítka, k vytvoření požadavku. Pokud nejsou obě pole vyplněná nebo již existuje nedokončený požadavek se stejnou kombinací aplikace a počítače, jsou hodnoty vynulovány a uživatel upozorněn chybovou hláškou.

Obrázek 5.7: Logický design – Instalace



Zdroj: autor

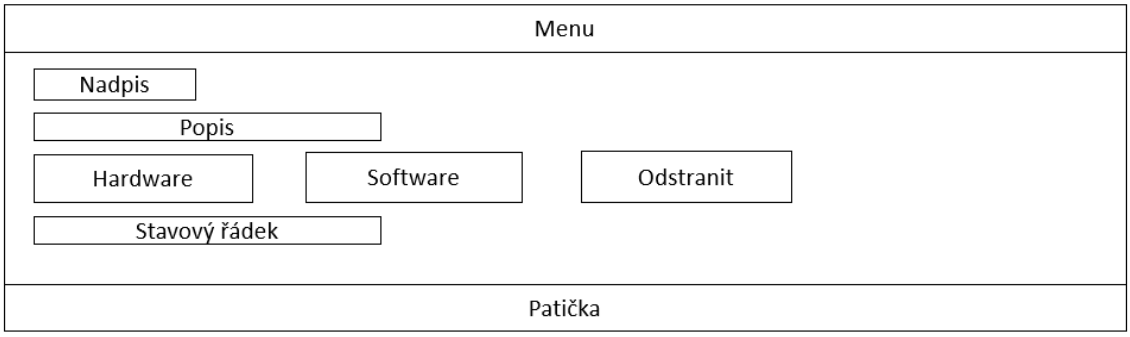
Obrázek 5.8: Grafický design – Instalace



Zdroj: autor

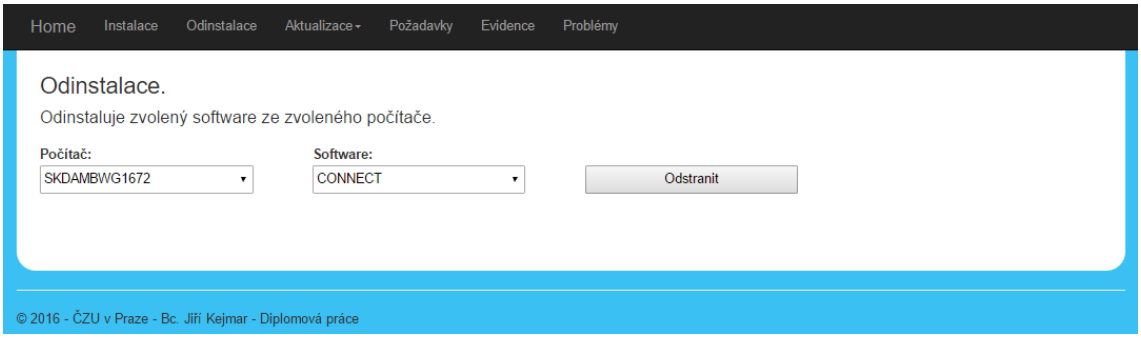
- **Odinstalace** – Stránka obsahuje také dva rozbalovací seznamy a tlačítko (viz Obrázek 5.9 a Obrázek 5.10). Dostupný je pouze seznam počítač, který obsahuje všechny počítače z databáze. Po vybrání konkrétního počítače dojde, k načtení pouze aplikací evidovaných na tento počítač. Po kliknutí je vytvořen požadavek. Pokud nejsou obě pole vyplněna nebo již existuje nedokončený požadavek se stejnou kombinací aplikace a počítače, jsou hodnoty vynulovány a uživatel upozorněn chybovou hláškou.

Obrázek 5.9: Logický design – Odinstalace



Zdroj: autor

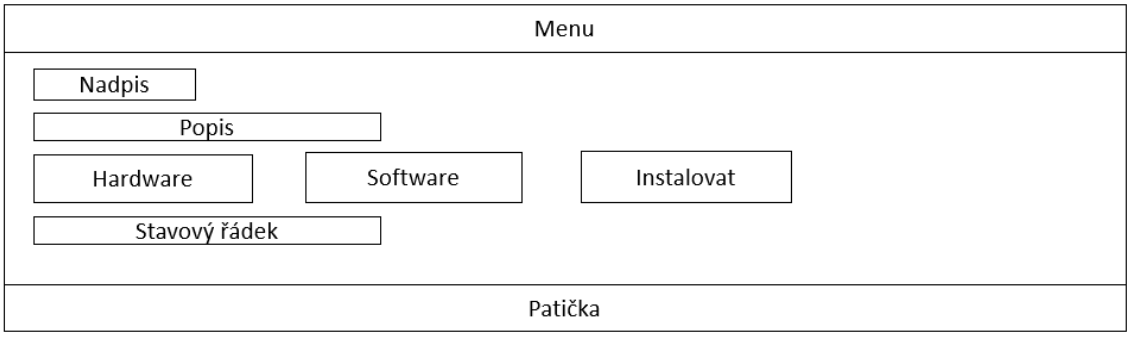
Obrázek 5.10: Grafický design – Odinstalace



Zdroj: autor

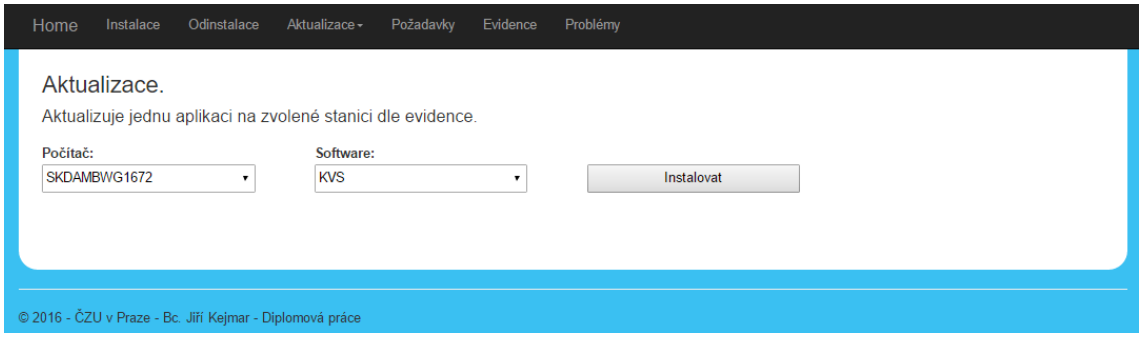
- **Aktualizace** – Stránka funguje analogicky jako stránka odinstalace. Pouze vytvořený požadavek je v jiném formátu (viz Obrázek 5.11 a Obrázek 5.12).

Obrázek 5.11: Logický design – Aktualizace



Zdroj: autor

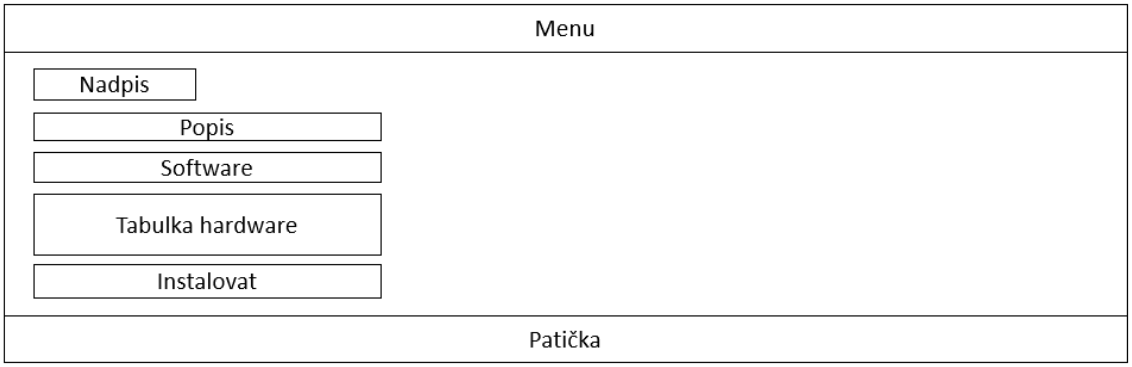
Obrázek 5.12: Grafický design – Aktualizace



Zdroj: autor

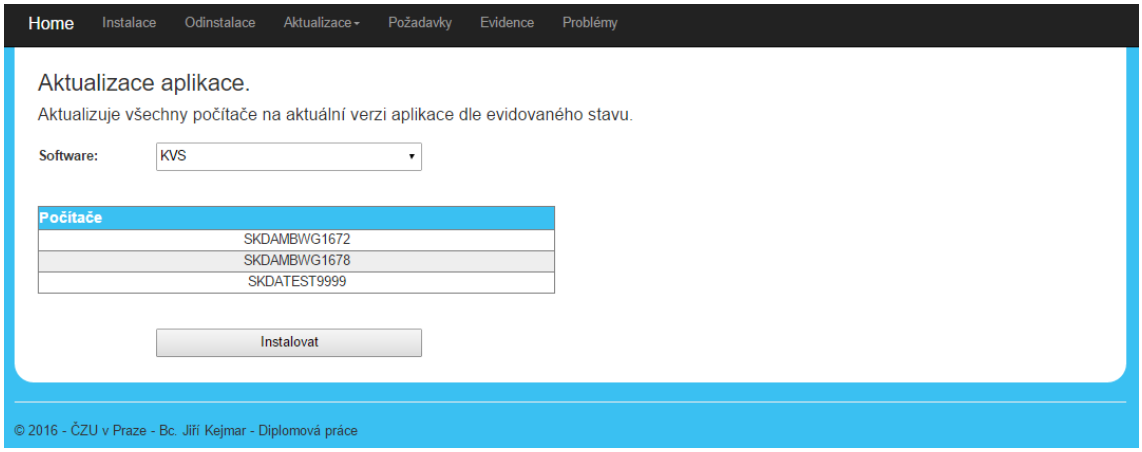
- **Aktualizace software** – Stránka obsahuje nejprve jeden rozbalovací seznam. Ten má načtené všechny aplikace z databáze. Po kliknutí se zobrazí seznam počítačů, které jsou na tuto aplikaci evidované (viz Obrázek 5.13 a Obrázek 5.14). Tlačítko vytvoří požadavky. Pokud již existuje nedokončený požadavek se stejnou kombinací aplikace a počítače, není tento požadavek vytvořen.

Obrázek 5.13: Logický design – Aktualizace software



Zdroj: autor

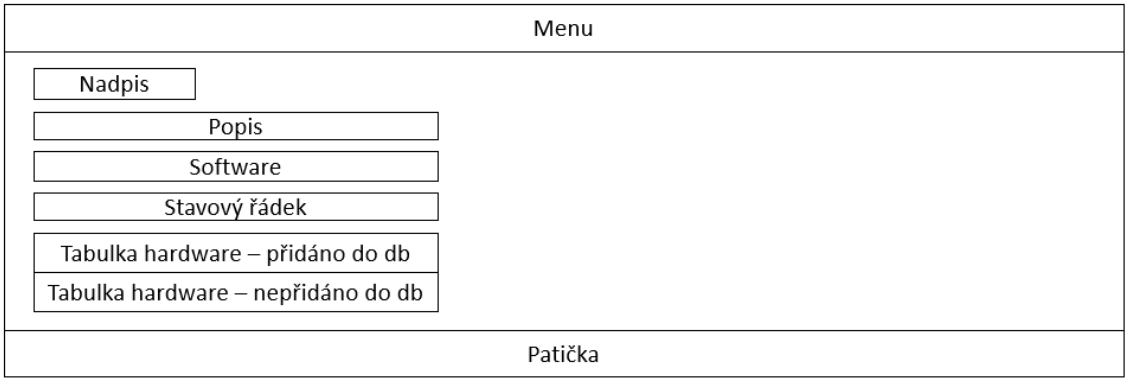
Obrázek 5.14: Grafický design – Aktualizace software



Zdroj: autor

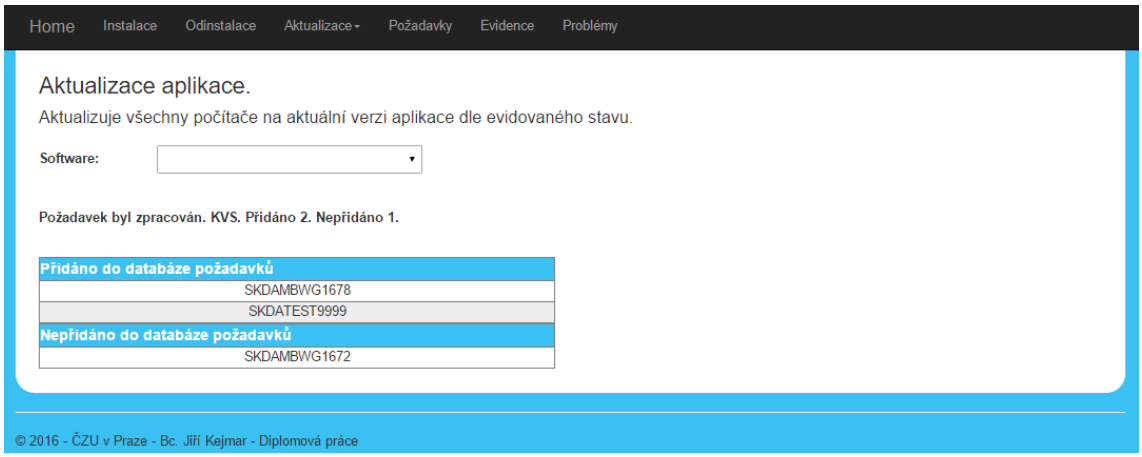
Po potvrzení zmizí tlačítko a původní tabulka s počítači. O výsledku vytvoření požadavků je uživatel informován pomocí hlášky a dvou tabulek s úspěšnými a neúspěšnými požadavky (viz Obrázek 5.15 a Obrázek 5.16).

Obrázek 5.15: Logický design – Aktualizace software – potvrzení



Zdroj: autor

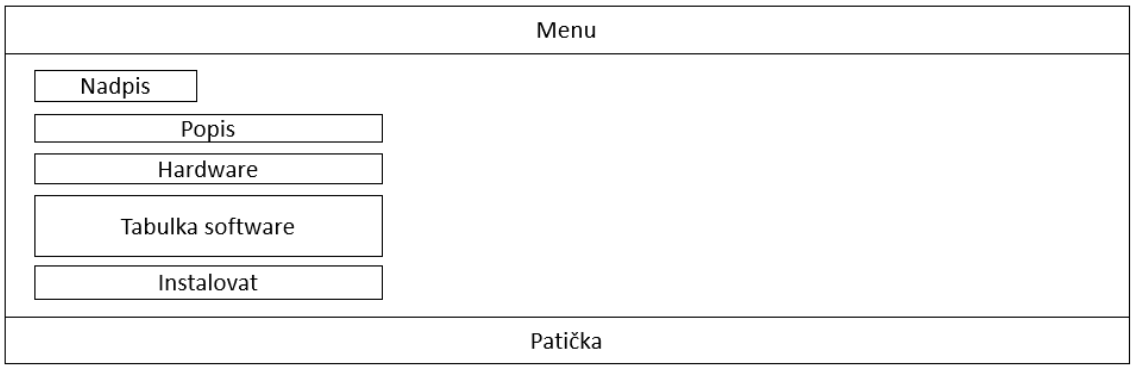
Obrázek 5.16: Grafický design – Aktualizace software – potvrzení



Zdroj: autor

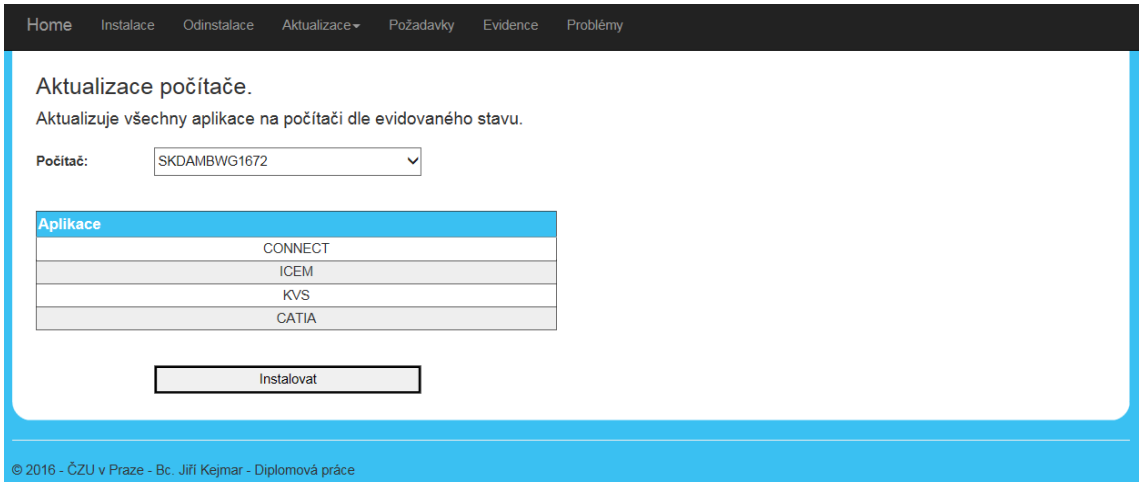
- **Aktualizace hardware** – Tato stránka funguje analogicky k aktualizaci software, pouze s rozdílem, že jsou aktualizovány všechny aplikace evidované na konkrétním počítači (viz Obrázek 5.17 a Obrázek 5.18). Lze využívat například po reinstalaci operačního systému, nebo při změně počítače v případě operativního leasingu.

Obrázek 5.17: Logický design – Aktualizace hardware



Zdroj: autor

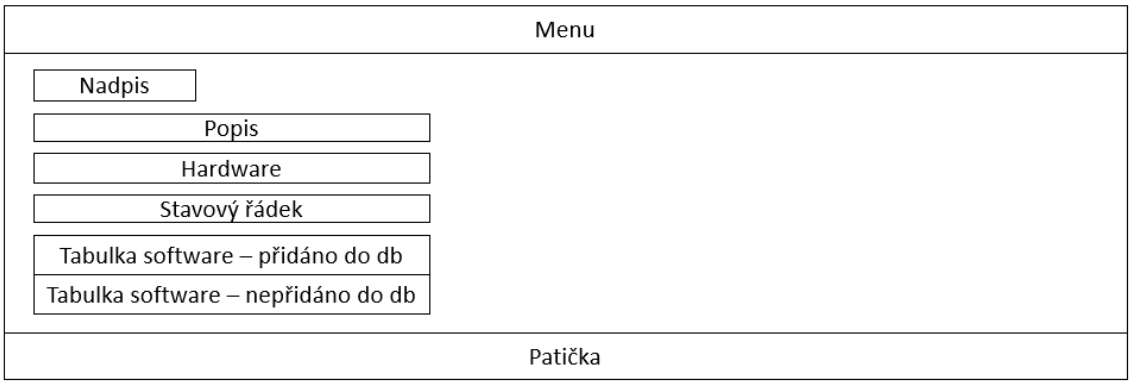
Obrázek 5.18: Grafický design – Aktualizace hardware



Zdroj: autor

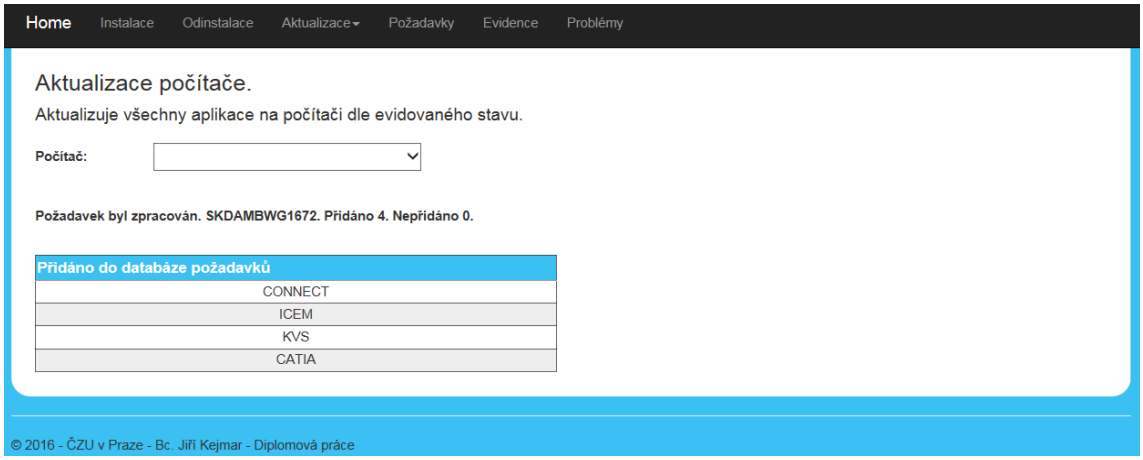
Po potvrzení zmizí tlačítko a původní tabulka s aplikacemi. Zobrazí se tabulky s úspěšně a nespěšně zařazenými aplikacemi (viz Obrázek 5.19 a Obrázek 5.20).

Obrázek 5.19: Logický design – Aktualizace hardware



Zdroj: autor

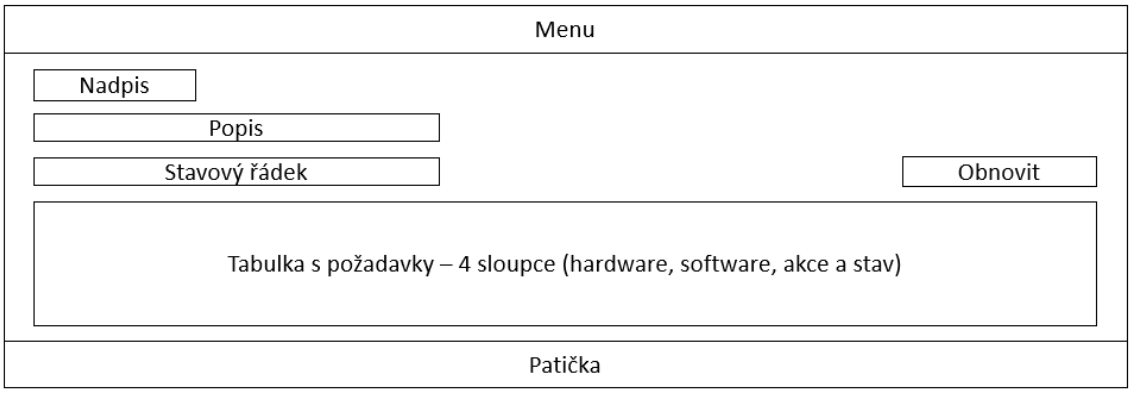
Obrázek 5.20: Grafický design – Aktualizace hardware – potvrzení



Zdroj: autor

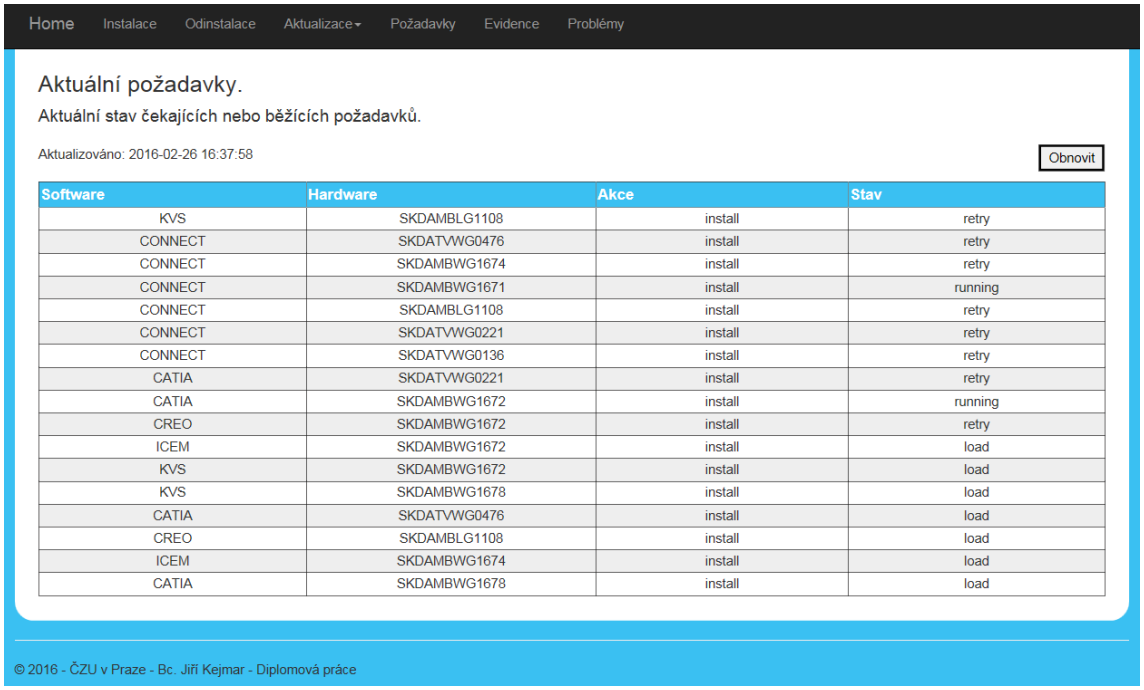
- **Požadavky** – Stránka zobrazuje aktuální stav požadavků v databázi pomocí tabulky. Jsou zobrazovány pouze požadavky se stavem new, load, retry a running (viz Obrázek 5.21 a Obrázek 5.22). Tlačítko obnovit provede načtení aktuálních dat a změní čas poslední aktualizace.

Obrázek 5.21: Logický design – Požadavky



Zdroj: autor

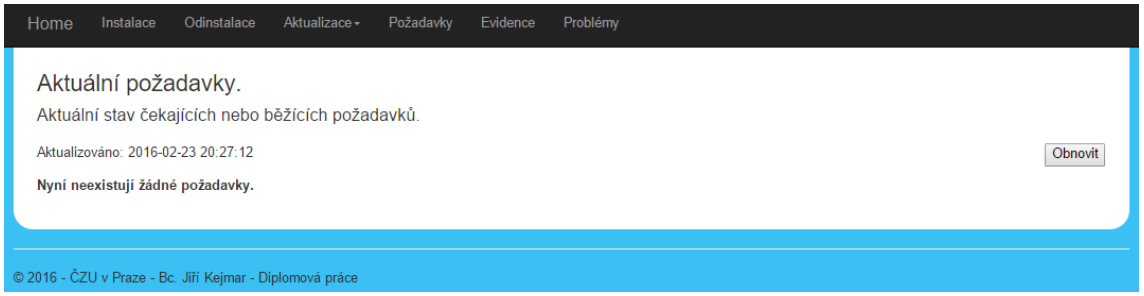
Obrázek 5.22: Grafický design – Požadavky



Zdroj: autor

Pokud nejsou v systému žádné požadavky, není tabulka zobrazena (viz Obrázek 5.23).

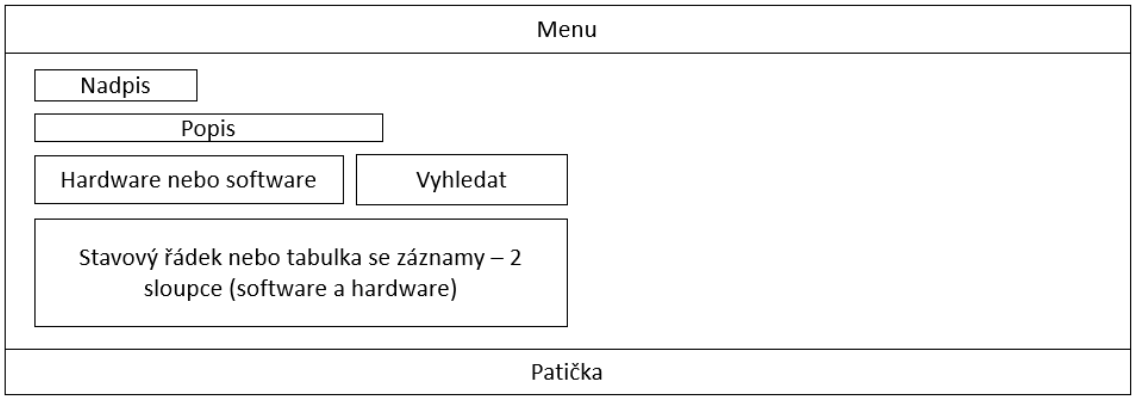
Obrázek 5.23: Grafický design – Požadavky – Žádné požadavky



Zdroj: autor

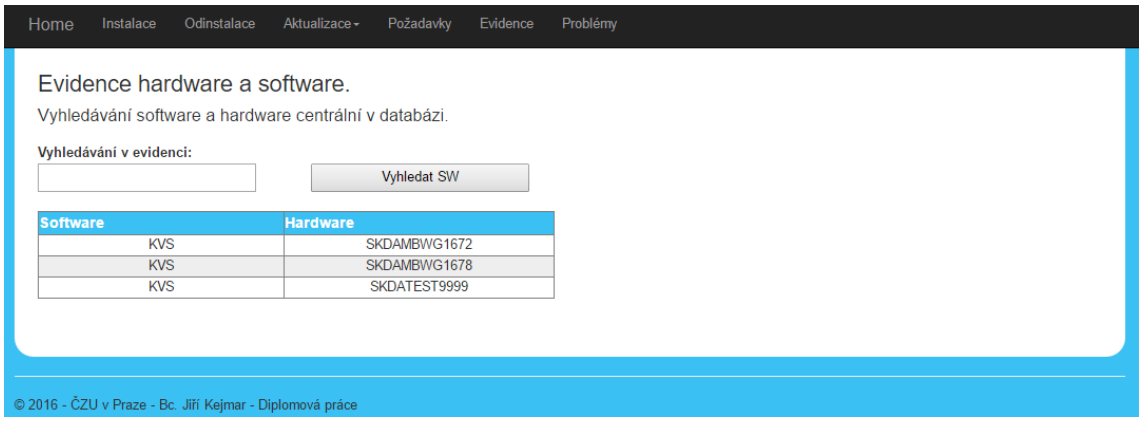
- **Evidence** – Stránka obsahuje nejprve jedno textové pole a tlačítko. Po kliknutí se zobrazí tabulka z evidence se seznamem počítačů a aplikací, které obsahující klíčové slovo uvedené v textovém poli (viz Obrázek 5.24 a Obrázek 5.25). Textové pole je univerzální, jak pro hledání podle aplikací, tak i pro hledání podle počítačů. Pokud se hledaný řetězec v evidenci nevyskytuje, tak je uživatel informován pomocí hlášky.

Obrázek 5.24: Logický design – Evidence



Zdroj: autor

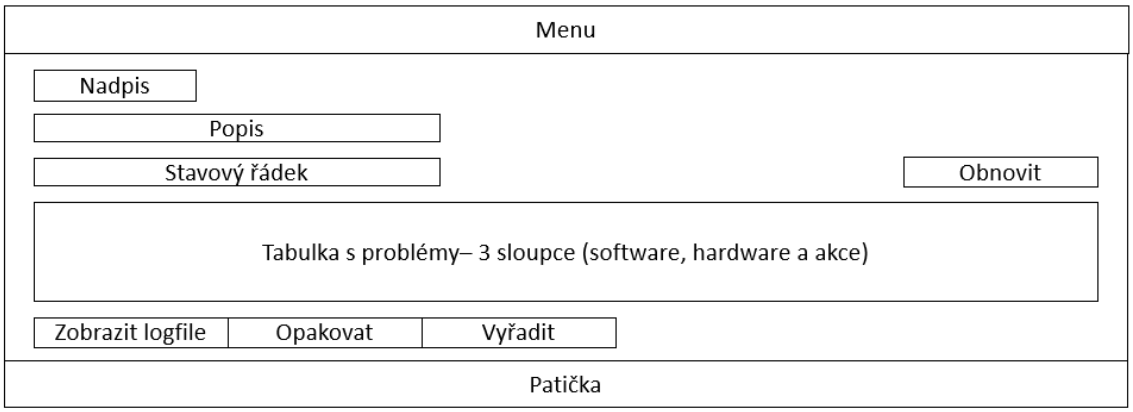
Obrázek 5.25: Grafický design – Evidence



Zdroj: autor

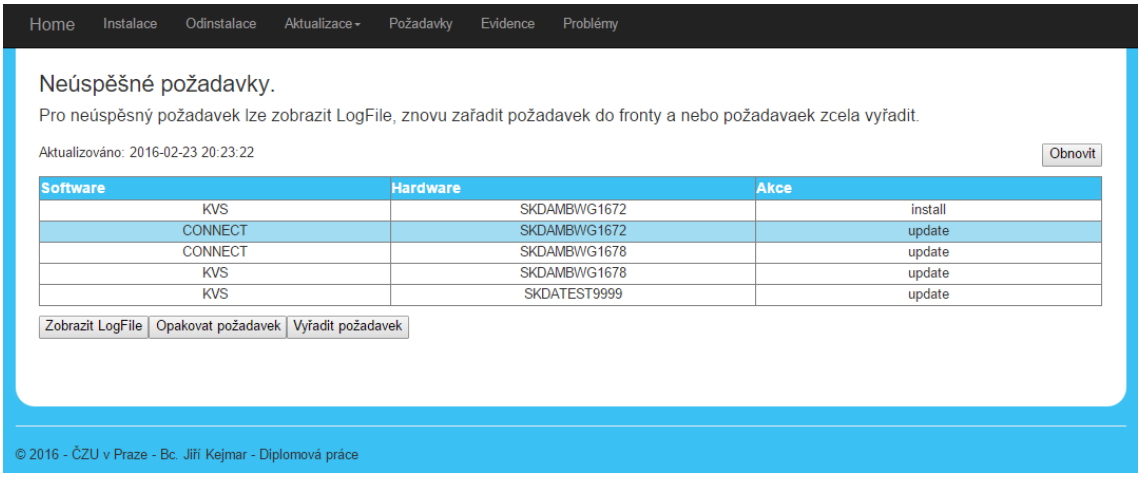
- **Problémy** – Stránka zobrazuje problematické požadavky pomocí tabulky. Jsou zobrazovány pouze požadavky se stavem failed (viz Obrázek 5.26 a Obrázek 5.27). Tlačítko obnovit provede načtení aktuálních dat a změní čas poslední aktualizace. Po kliknutí na řádek se provede jeho selekce. K požadavku je možné otevřít záznam o běhu instalace (logFile). Dále lze požadavek opakovat například po odstranění příčiny problému. Poslední možností je vyřazení požadavku ze systému.

Obrázek 5.26: Logický design – Problémy



Zdroj: autor

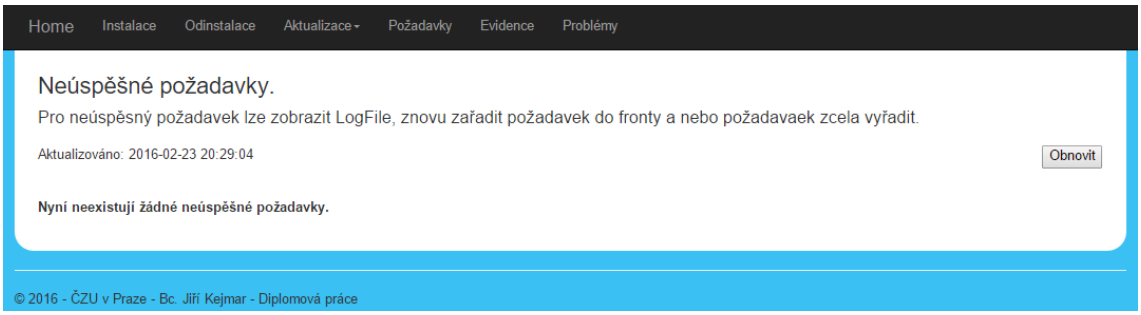
Obrázek 5.27: Grafický design – Problémy



Zdroj: autor

Pokud nejsou v systému žádné požadavky, nejsou tabulka ani tlačítka zobrazeny (viz Obrázek 5.28).

Obrázek 5.28: Grafický design – Problémy – Žádné problémy



Zdroj: autor

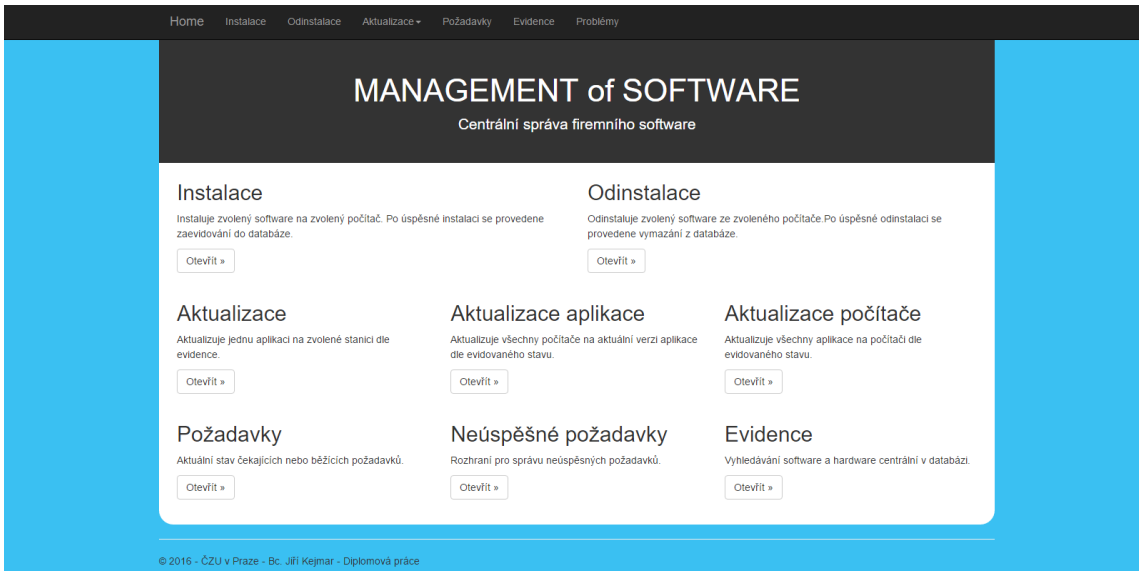
5.2.1.1 **Responzivní design**

Responzivní design je způsob stylování HTML dokumentu, který zaručí, že zobrazení stránky bude optimalizováno pro všechny druhy zařízení. Například počítače, mobily, notebooky a tablety. Toho lze dosáhnout především díky vlastnosti Media Queries, která je zahrnuta ve specifikaci CSS3. Tato vlastnost umožnuje rozpoznat vlastnosti zařízení, na kterém je stránka prohlížena a přizpůsobit mu tak samotnou stránku a její obsah. [22]

Pro implementaci responzivního webu se hodí v dnešní době masivně používaný open source framework Bootstrap. Tento framework kombinuje využití technologií HTML, CSS, a JavaScript. Bootstrap lze také využít ve Visual Studiu pro vývoji ASP.NET aplikace. Lze stáhnout pomocí NuGet Manageru. [22]

Obrázek 5.29 zobrazuje vstupní stránku v okně o šířce 1920px. Uspořádání vychází z uživatelsky obvyklého paradigmatu webových stránek.

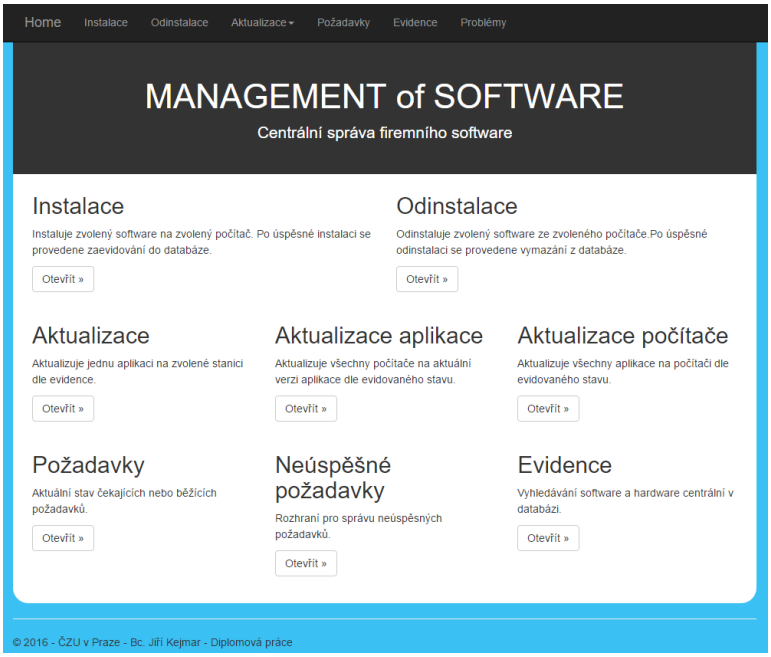
Obrázek 5.29: Responzivní design – šířka 1920px



Zdroj: autor

Obrázek 5.30 zobrazuje vstupní stránku v okně o šířce 1024px. Při zobrazení v tomto rozlišení se využívá maximální plocha obrazovky. Došlo ke zmenšení šířky objektů, aby mohla být zachována dostatečná velikost textu.

Obrázek 5.30: Responzivní design – šířka 1024px



Zdroj: autor

Obrázek 5.31 zobrazuje vstupní stránku v okně o šířce 480px. V tomto zobrazení se nejvíce hledí na využití možné plochy, zachování textu v podobě vhodné pro čtení a ovládání na malém displeji. Došlo k přesunutí bloků pod sebe. U menu došlo ke kontrakci do tzv. Hamburger menu. Menu je dostupné po kliknutí na ikonu v pravém horním rohu.

Obrázek 5.31: Responzivní design – šířka 480px

Zdroj: autor

5.2.2 Implementace

Aplikace obsahuje webové stránky default, delete, install, problems. queue, records, upradeExact, updateOneHW a updateOneSW.

5.2.2.1 Tvorba stránek

Definici základní struktury a shodných objektů (menu, patička) webových stránek obsahuje soubor Site.master (viz Příklad 5.6). Jedná se o tzv. MasterPageFile. Pro názornost byla ukázka zjednodušena. Nyní neobsahuje definici skriptů a realizaci menu pomocí odkazů `a`, které jsou strukturované pomocí nečíslovaných seznamů `ul` a `li`.

Příklad 5.6: Kód Site.master – obsah zjednodušen

```
<%@ Master Language="C#" AutoEventWireup="true" %>
<!DOCTYPE html>
<html lang="en">
<head runat="server">
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>ManagementOfSoftware - <%: Page.Title %></title>
    <asp:PlaceHolder runat="server">
       <%: Scripts.Render("~/bundles/modernizr") %>
    </asp:PlaceHolder>
    <webopt:bundlereference runat="server" path="~/Content/css" />
    <link href="~/install-icon.png" rel="shortcut icon" type="image/x-icon" />
</head>
<body>
    <form runat="server">
        <asp:ScriptManager runat="server">
            <Scripts><%--skripty--%></Scripts>
        </asp:ScriptManager>

        <div class="navbar navbar-inverse navbar-fixed-top">
            <div class="container">
                    <%--navigace--%>
            </div>
        </div>
        <div class="container body-content">
            <asp:ContentPlaceHolder ID="MainContent" runat="server">
            </asp:ContentPlaceHolder>  <hr />
            <footer>
               <p>&copy;2016 - ČZU v Praze - Bc. Jiří Kejmar - Diplomová práce</p>
            </footer>
        </div>
    </form>
</body>
</html>
```

Zdroj: autor

Každá stránka je tvořena souborem .aspx a tzv. code behing souborem aspx.cs. Příklad 5.7 zobrazuje zdrojový kód stránky install.aspx. Struktura a použité značky vycházejí z HTML. Dále jsou doplněny značky ASP.

Příklad 5.7: Kód install.aspx

```
<%@ Page Title="Nová instalace" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeFile="Install.aspx.cs" Inherits="Install" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
    <div id="c1" class="container-fluid bg-white">
        <h3><%: Title %>.</h3>
        <h4>Instaluje zvolený software na zvolený počítač. </h4>
        <h3></h3>
        <div class="row">
            <div class="col-sm-4 col-md-3">
                    <asp:Label Text="Software:" runat="server"    Width="100"
                    Font-  Bold="true"></asp:Label>
                    <asp:DropDownList ID="DropDownSW" runat="server" Height="30"
                    Width="225" AutoPostBack="true"
                    OnInit="DropDownSW_Init"></asp:DropDownList>
            </div>
            <div class="col-sm-4 col-md-3">
                    <asp:Label Text="Počítač:" runat="server" Width="100" Font-
                    Bold="true"></asp:Label>
                    <asp:DropDownList ID="DropDownHW" runat="server" Height="30"
                    Width="225" AutoPostBack="true"
                    OnInit="DropDownHW_Init"></asp:DropDownList>
            </div>
            <div class="col-sm-3 col-md-3">
                    <asp:Label Text="" runat="server" Width="100"></asp:Label>
                    <asp:button ID="buttonInstall" Height="30" Width="225"
                    text="Instalovat" runat="server" OnClick="buttonInstall_Click"
                    />
            </div>
             <div class="col-sm-1 col-md-3">
            </div>
        </div>
        <div class="row">
            <div class="col-xs-9 col-sm-11 col-md-9">
                <h3></h3>
            </div>
        </div>
        <div class="row">
            <div class="col-xs-9 col-sm-11 col-md-9">
                    <asp:Label ID="lblState" Text="" runat="server" Height="30"
                    Font-Bold="true"></asp:Label>
            </div>
        </div>
    </div>
</asp:Content>
```

Zdroj: autor

Výše uvedený příklad definuje, společně s kaskádovými styly, vzhled stránky. Aplikační logika je oddělena do code behing souboru install.aspx.cs. Příklad 5.8 zobrazuje

pouze strukturu tříd a metod. Tato třída může přistupovat k objektům webových stránek díky dědění z objektu Page.

Příklad 5.8: Kód install.aspx.cs – obsah metod skryt

```
using System;
using System.Web.UI;
using System.Data;

public partial class Install : Page
{
    protected void Page_Load(object sender, EventArgs e)
    {…}

    protected void ButtonInstall_Click(object sender, EventArgs e)
    {…}

    protected void DropDownSW_Init(object sender, EventArgs e)
    {…}

    protected void DropDownHW_Init(object sender, EventArgs e)
    {…}
}
```

Zdroj: autor

Ostatní webové stránky jsou vytvořeny analogicky k výše uvedeným příkladům.

5.2.2.2 Komunikace s databází

Komunikace s databází je realizována pomocí třídy `Database`. Třída `Database` obsahuje metody:

- `AddNewQueue` – Provede zápis požadavku do fronty (queue). Nejprve zkontroluje, zda tam již není řádek obsahující stejnou kombinaci názvu počítače a aplikace (viz Příklad 5.9). Výjimkou je status `finished` u takové shody. Metoda vrací hodnotu `true` nebo `false`.

Příklad 5.9: Metoda AddNewQueue

```
public bool AddNewQueue(string sw, string hw, string ac)
{
sql = "SELECT * FROM queue WHERE hw_name ='" + hw + "' AND sw_name ='" + sw + "'
AND state is not 'finished'";

    using (SQLiteConnection sqlite_conn = new SQLiteConnection(dbLocation))
    {
        sqlite_conn.Open();
        using (SQLiteCommand sqlite_cmd = new SQLiteCommand(sql, sqlite_conn))
        {
              using (SQLiteDataReader sqlite_datareader =
              sqlite_cmd.ExecuteReader())
             {
                if (sqlite_datareader.HasRows == false)
                {
                     sql = "INSERT INTO queue('sw_name',
                     'hw_name','state','action') VALUES ('" + sw + "', '" + hw +
                     "', 'new', '" + ac + "' );";

                     using(SQLiteCommand cmd = new SQLiteCommand(sql, sqlite_conn))
                     {
                         cmd.ExecuteNonQuery();
                     }
                     returnValue = true;
                }
                else
                {
                     returnValue = false;
    }   }   }   }
    return returnValue;
}
```

Zdroj: autor

- `LoadRecords` – Velmi komplexní metoda. Provede připojení k databázi, a načtení záznamů dle požadovaných kritérií z tabulky records (viz Příklad 5.10). Metoda vrací již zpracovanou kolekci dle potřeby.

Příklad 5.10: Metoda LoadRecords – obsah zjednodušen

```
public DataTable LoadRecords(string findString, int type)
{
    DataTable dtSource = new DataTable();
    DataTable dt = new DataTable();

    switch (type)
    {
        case 0:
              sql = "SELECT * FROM '" + findString + "'";
              using (SQLiteConnection sqlite_conn = new
              SQLiteConnection(dbLocation))
              {
                     sqlite_conn.Open();
                     using (SQLiteCommand sqlite_cmd = new SQLiteCommand(sql,
                     sqlite_conn))
                     {
                            SQLiteDataAdapter adapter = new
                            SQLiteDataAdapter(sqlite_cmd);
                            adapter.Fill(dt);
              }      }
              break;

        …
        default:
              throw new NullReferenceException();
    }
    return dt;
}
```

Zdroj: autor

- `CheckData` – Provede kontrolu, zda se již nevyskytuje ve frontě nevyřízená stejná kombinace počítače a aplikace (viz Příklad 5.11). Metoda vrací hodnotu `true` nebo `false`.

Příklad 5.11: Metoda CheckData

```
public bool CheckData(string sw, string hw)
{
    DataTable dt = new DataTable();
    dt = LoadRecords("queue",0);
    foreach (DataRow dr in dt.Rows)
    {
        if (hw == dr[2].ToString())
        {
            if (sw == dr[1].ToString())
            {
                if (dr[3].ToString() != "finished")
                {
                    return false;
    }   }   }   }
    return true;
}
```

Zdroj: autor

- `DeleteQueue` – Provede okamžité vymazání požadavku z fronty (queue). Metoda vrací hodnotu `true` nebo `false` (viz Příklad 5.12).

Příklad 5.12: Metoda DeleteQueue

```
public bool DeleteQueue(string sw, string hw, string ac)
{
    sql = "DELETE FROM queue WHERE sw_name='" + sw + "' AND hw_name='" + hw + "'
    AND action='" + ac + "';";
    try
    {
        using (sqlite_conn = new SQLiteConnection(dbLocation))
        {
            sqlite_conn.Open();
            using (SQLiteCommand cmd = new SQLiteCommand(sql, sqlite_conn))
            {
                cmd.ExecuteNonQuery();
            }
        }
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}
```

Zdroj: autor

5.3 Služba

V této kapitole je popisováno zpracování návrhu služby a její implementace.

5.3.1 Návrh

5.3.1.1 Zpracování požadavků

Služba bude zajišťovat zpracování nových požadavků z databáze. Tok požadavků bude tvořit různě dlouhou frontu, kterou je potřeba v relativně krátkém čase zpracovat. Tento požadavek ukazuje na použití vícevláknového zpracování.

.NET Framework podporuje paralelní spouštění kódu pomocí tzv. multithreadingu. Jinak řečeno, lze spouštět několik částí kódu najednou a každou část na samostatném vlákně. .NET Framework aplikace nebo služba využívá ve výchozím stavu pouze jedno vlákno, které vytváří běhové prostředí CLR. Označuje se jako primární vlákno. Pro použití dalšího vlákna se musí používat direktiva `System.Threading` a nové vlákno si vytvořit. [11]

5.3.1.2 Přístup ke vzdálené stanici

Služba bude muset také přistupovat ke vzdálené stanici v lokální síti firmy. Dle požadavku je nutné použít stabilní řešení nahrazující PsExec. Jako schopná a použitelná náhrada je služba WMI neboli Windows Management Instrumentation.

WMI je implementace správy WBEM (Web-Based Enterprise Management) od společnosti Microsoft. WBEM je iniciativa ke stanovení standardů pro přístup a sdílení informací správy v rozlehlé síti. [23]

Pomocí služby WMI lze v nástrojích pro správu počítače spravovat vzdálené počítače. Dále je možné pomocí WMI a skriptovacích nebo programovacích jazyků získat podrobné údaje o konfiguraci vzdálení stanice nebo lze s její pomocí provést změny konfigurace. Pro využití služeb WMI se musí používat direktiva `System.Management`. [23]

Pro tuto službu je podstatnými schopnostmi spouštění vzdálených procesů na počítače v síti a monitorovat jejich stav.

5.3.1.3 Logování

Je nutné zajistit logování jak běhu služby, tak i aktivity spojené s jednotlivými instalacemi, aktualizace a odinstalacemi.

5.3.2 Implemenatace

Podkapitola popisuje konkrétní metody implementace jednotlivých částí služby.

5.3.2.1 Vícevláknové zpracování

Po zvážení dostupných možností vícevláknového zpracování a požadavků na škálovatelnost řešení byla vybrána metoda vytváření nových vláken pomocí příkazu `Timer` z direktivy `System.Threading`. `System.Threading.Timer` umožňuje provádět určitou metodu pomocí nového vlákna v definovaném intervalu. Objekt `Timer` je efektivní ve využívání systémových prostředků.

Při používání vícevláknové technologie je nutné myslet na možné problémy s přístupem ke sdíleným entitám. Kód, který zajišťuje tuto bezpečnost, lze nazývat thread-safe. Při tvorbě systému byly zajištěny statické prvky pomocí konstrukce `lock` (viz kapitola 5.3.2.2) a přístupy do databáze pomocí `using` (viz kapitola 5.3.2.5).

Příklad 5.13 zobrazuje definice dvou použitých objektů `Timer`:

- `timerLoadDB` – načítá nové požadavky z databáze
- `timerRunInstall` – spouští požadavky z nečtené fronty

Příklad 5.13: Vícevláknové zpracování

```
static class Program
{
   private static Timer timerLoadDB;
   private static AutoResetEvent autoEventDB;
   private static StatusCheckerTickDB statusCheckerTickDB;
   private static TimerCallback timerDbDelegate;
   private static Timer timerRunInstall;
   private static AutoResetEvent autoEventInstall;
   private static StatusCheckerTickInstall statusCheckerTickInstall;
   private static TimerCallback timerInstallDelegate;

      static void Main()
      {
          autoEventDB = new AutoResetEvent(false);
          statusCheckerTickDB = new StatusCheckerTickDB(2520); //reset po týdnu
          timerDbDelegate = new TimerCallback(statusCheckerTickDB.CheckStatus);
          timerLoadDB = new Timer(timerDbDelegate, autoEventDB, 1000,
          intervalLoadDB);

          autoEventInstall = new AutoResetEvent(false);
          statusCheckerTickInstall = new StatusCheckerTickInstall(2880); //reset
          po 12 hodinách
          timerInstallDelegate = new
          TimerCallback(statusCheckerTickInstall.CheckStatus);
          timerRunInstall = new Timer(timerInstallDelegate, autoEventInstall, 2000,
          intervalRunInstall);
      }
}
```

Zdroj: autor

Veškeré komponenty musí být deklarovány nikoli v metodě, ale ve třídě, jinak by po pár minutách došlo k odstranění objektů Garbage Collectorem.

Při uvažované průměrné době instalace aplikace dvacet minut, je při použití padesáti vláken teoreticky maximální počet 240 instalací za hodinu nebo 2880 za 12 hodin. Při takto definovaných podmínkách se uvažuje následující nastavení parametrů:

- Interval načtení dat z DB: 4 minuty.
- Interval spuštění vlákna: 15 vteřin.
- Vyřazení požadavku (`RetryIndex`): 300 iterací (přibližně 24 hodin).
- Načtená fronta: 30 požadavků.

`RetryIndex` je implementován pomocí třídy `StatusLimitation`, ve které se udržuje aktuální hodnota indexu pro jedinečnou kombinaci hardware a software (například

ComputernameSoftware). Jako datová struktura je použita kolekce System.Collections.Concurrent.ConcurrentDictionary. Obsahuje dvojice klíčů a jejich hodnoty, ke kterým lze přistupovat pomocí více vláken současně.

5.3.2.2 Zamykání objektů

Pro zajištění thread-safe kódu je potřeba vyřešit přístup více vláken ke sdílenému objektu v čase. Použití objektu typu lock je naznačeno na zjednodušeném kódu zápisu do souboru (viz Příklad 5.14).

Příklad 5.14: Zamykání objektů – obsah zjednodušen

```
        private static object myLockerLock = new object();

        public static void Log(string message)
        {
            lock (myLockerLock)
            {
                    File.AppendAllText(AppDomain.CurrentDomain.BaseDirectory +
                    "logFile.txt", message);
            }
        }
```

Zdroj: autor

Tato možnost zamykání funguje na principu, že první vlákno, které vstoupí do konstrukce lock, tak tímto vstoupením za sebou uzamkne. Další vlákno již bude čekat na uvolnění, které provede automaticky první vlákno při opuštění konstrukce zámku.

5.3.2.3 Logování

Implementace logování je realizována pomocí statické třídy Logger (viz Příklad 5.15). Třída obsahuje přetížený konstruktor. Varianta s jedním vstupním atributem (řetězec) slouží pro zaznamenávání běhu samotné služby. Výsledný výpis je uveden v příloze B. Druhá varianta obsahuje řetězcové atributy software, hardware a zprávu. Tato varianta slouží pro zaznamenávání aktualizace konkrétní aplikace na konkrétní počítač. Výhodou je, že lze v souboru zobrazit i historii. Příklady záznamů jsou uvedeny v příloze A.

Příklad 5.15: Logování

```
static class Logger
{
    private static object myLockerLock = new object();

    public static void Log(string message)
    {
        lock (myLockerLock)
        {
            try
            {
              string _message = String.Format("{0} {1}",
              message, Environment.NewLine);
               File.AppendAllText(AppDomain.CurrentDomain.BaseDirectory +
              "logFile.txt", Date.Now() + " - " + _message);
            }
            catch (Exception)
            {
                //Odchycení vyjímky - nepodařilo se zapsat do souboru
            }
        }
    }
    public static void Log(string sw, string hw, string message)
    {
        lock (myLockerLock)
        {
            try
            {
              string _message = String.Format("{0} {1}", message,
              Environment.NewLine);
               File.AppendAllText(Program.FilesLocation + sw.ToLower() + "\\logs\\"
              + h.ToLower) + ".log", Date.Now() + " - " + _message);
            }
            catch (Exception)
            {
                //Odchycení vyjímky - nepodařilo se zapsat do souboru
            }
        }
}   }
```

Zdroj: autor

5.3.2.4 Požadavky

Pro správu požadavků se používá objekt `System.Collections.Queue`, který zajišťuje přidání požadavků na konec fronty a odebrání zezačátku (metoda FIFO).

Příklad 5.16: Požadavky

```
        //Definice
              public static Queue<string> swQ = new Queue<string>();
        //Přidání do fronty
              Program.swQ.Enqueue(Convert.ToString(sqlite_datareader["sw_name"]));
        //Odebrání z fronty
              string sw = Program.swQ.Dequeue();
```

Zdroj: autor

5.3.2.5 Komunikace s databází

Třída `Database` implementuje kompletní komunikaci s databází. Třída je velmi obsáhlá a obsahuje metody:

- `LoadState` – Provede připojení k databázi a načtení záznamů dle požadovaných stavů z tabulky queue. Zároveň načteným požadavkům změní status z `new` nebo `retry` na `load`.
- `LoadStateFailed` – Provede připojení k databázi a načtení požadavků se stavem `running`. Zajistí změnu stavu nedokončených požadavků.
- `LoadQueue` – Definuje logiku načítání požadavků do fronty. Při prvním načtení jsou načteny požadavky ve stavu `new`, `retry` a `load`. Požadavky, které nebyly v době spuštění dokončené, se nechají nastavit na `failed`. Tím se předchází zaseknutí požadavků v systému navždy. Dále se načítají jen požadavky `new` a `retry`.
- `ChangeState` – Provede okamžitou změna stavu v databázi. Například z `running` na `finished`.
- `AddNewRecord` – Provede zápis do evidence (records), po úspěšném vyřízení požadavku.
- `CheckandBlockRunningHardware` – Slouží k ověření, zda již v aktuální době neprobíhá na stanici jiná instalace. Pokud ano, nastaví požadavku status `retry`.

Příklad 5.17 zobrazuje zjednodušenou ukázku komunikace s databází. Při použití `using`, není potřeba spojení explicitně uzavírat spojení s databází pomocí příkazu `Close`. `SQLiteConnection` se vždy uzavře po ukončení činnosti vlákna v této metodě. Pokud by nebyla použita konstrukce `using`, mohlo by dojít k vyskočení výjimky (například zamčená databáze), ještě před ukončením spojení. Takové chování by mělo za následek nestabilní aplikaci.

Příklad 5.17: Ukázka komunikace s databází – obsah zjednodušen

```
private string sql;

public void LoadState(string odkud, string co, string kde)
{
   sql = "SELECT * FROM " + odkud + " WHERE " + co + "='" + kde + "'";

   using (SQLiteConnection sqlite_conn = new SQLiteConnection(Program.DbLocation))
       {
          sqlite_conn.Open();
          using (SQLiteCommand sqlite_cmd = new SQLiteCommand(sql, sqlite_conn))
              {
                     using (SQLiteDataReader sqlite_datareader =
                     sqlite_cmd.ExecuteReader())
                     {
                         while (sqlite_datareader.Read())
                         {
                             if (Program.hwQ.Count <= 30)
                             {
                                    Program.swQ.Enqueue(Convert.ToString(sqlite_data
                                    reader["sw_name"]));
                                    Program.hwQ.Enqueue(Convert.ToString(sqlite_data
                                    reader["hw_name"]));
                                    Program.acQ.Enqueue(Convert.ToString(sqlite_data
                                    reader["action"]));
                             } else {
                                    Logger.Log(String.Format("Příliš mnoho požadavků
                                    ve frontě - ({0})", Program.hwQ.Count));
                                    break;
                             }
                         }
                     }
              }
       }
}
```

Zdroj: autor

5.3.2.6 Přístup ke vzdálené stanici

Přístup ke vzdálené stanici je realizován pomocí třídy `WMI`. Třída obsahuje jeden konstruktor a metodu `ExecuteRemoteProcessWMI`, která zajišťuje požadovanou činnost.

Pro spuštění nového procesu na vzdálené stanicí se musí vytvořit instance třídy `WMI` a poté použít její výše zmíněnou metodu `ExecuteRemoteProcessWMI`. Tato metoda vyžaduje parametry - název počítače, cesta ke spouštěcímu skriptu na cílovém počítači a maximální časový limit procesu (viz Příklad 5.18).

Příklad 5.18: Spuštění vzdáleného procesu

```
switch (ac)
{
    case "delete":
       WMI DeleteRemoteWMIProcess = new WMI();
       DeleteRemoteWMIProcess.ExecuteRemoteProcessWMI(Workstation.WorkstationName,
       "c:\\" + configuration["scriptDestination"], 1800000);
        break;

    case "install":
       WMI InstallRemoteWMIProcess = new WMI();
       InstallRemoteWMIProcess.ExecuteRemoteProcessWMI(Workstation.WorkstationName
       , "c:\\" + configuration["scriptDestination"], 7200000);
        break;

    case "update":
       WMI UpdateRemoteWMIProcess = new WMI();
       UpdateRemoteWMIProcess.ExecuteRemoteProcessWMI(Workstation.WorkstationName,
       "c:\\" + configuration["scriptDestination"], 7200000);
        break;

    default:
        break;
}
```

Zdroj: autor

Časový limit jedné odinstalace je 30 minut, instalace a aktualizace je 2 hodiny.

Program vytvoří na vzdálené stanici proces. Tento proces lze sledovat pomocí tzv. ProcesID. Proces spouští instalační skript, který následně provede akci za použití nakopírovaných zdrojových dat. Tyto data jsou po dokončení odstraněna.

5.3.2.7 Informace o konfiguraci stanice

Díky třídě je `WMI_Info` je možné zjišťovat detailní konfiguraci o vzdáleném počítači. Tyto informace lze použít, jak pro samotnou administraci stanic, tak pro případné analyzování příčiny problému. Tato data se zapisují do záznamů o instalaci. Ukázka zápisu v příloze A. Jde například o data - výrobce a model počítače, operační systém, bitová verze operačního systému, IP adresa, velikost paměti RAM anebo volné místo na disku.

5.3.2.8 Informace o stanici v síti

Počítače se v sítí identifikuji pomocí IP adresy. Tato adresa není pevně přiřazena každému počítači, ale je přidělována pomocí protokolu DHCP. Počítače v evidenci jsou ale evidované pomocí názvu počítače (computername). Tento překlad zajišťuje Domain Name Server (DNS). Pokud má firma rozdělenu interní síť do více domén, je nutné počítač najít v jedné z nich. [24]

Třída `NetInfo` zajišťuje právě nalezení počítače v jedné z domén datové sítě společnosti. Prostředkem je metoda `GetPingInfo` dotazovaná postupně na existenci počítače v konkrétní doméně z kolekce (viz Příklad 5.19).

Příklad 5.19: Zjištění dostupnosti stanici v doméně

```
private void GetPingInfo(string WorkstationName, string location)
{
    try
    {
       Ping WorkstationPing = new Ping();
       PingReply Workstationreply = WorkstationPing.Send(WorkstationName +
       netLocation[location], 500);
       workstationIPStatus = Workstationreply.Status;
       this.workstationLocation = location;
       this.workstationIPAdressDNS = Workstationreply.Address;
       this.workstationSufix = netLocation[location];
       workstationIPHostEntry = Dns.GetHostEntry(Workstationreply.Address);
       this.netInfoError = "Success";
    }
    catch (PingException)
    {
       this.netInfoError = "PingError";
    }
    catch (System.Net.Sockets.SocketException)
    {
       this.netInfoError = "DNSSocketError";
    }
    catch (ArgumentNullException)
    {
       this.netInfoError = "IPError";
    }
}
```

Zdroj: autor

Ve výsledku dochází ke stanovení plně specifikovaného doménového jméno počítače. Taktéž se označuje jako FQDN (například skdambwg1672.mb.skoda.vwg). Tyto informace se taktéž zapisují do záznamů počítače. Ukázka zápisu v příloze A.

5.3.2.9 Kopírování dat na stanici

Pro kopírování zdrojových dat a skriptů na stanici se používá třída `CopyDir`. Pro kopírování využívá metodu `CopyAll`. Tato metoda používá rekurzivní algoritmus neboli volání sama sebe (viz Příklad 5.20).

Příklad 5.20: Kopírování souborů

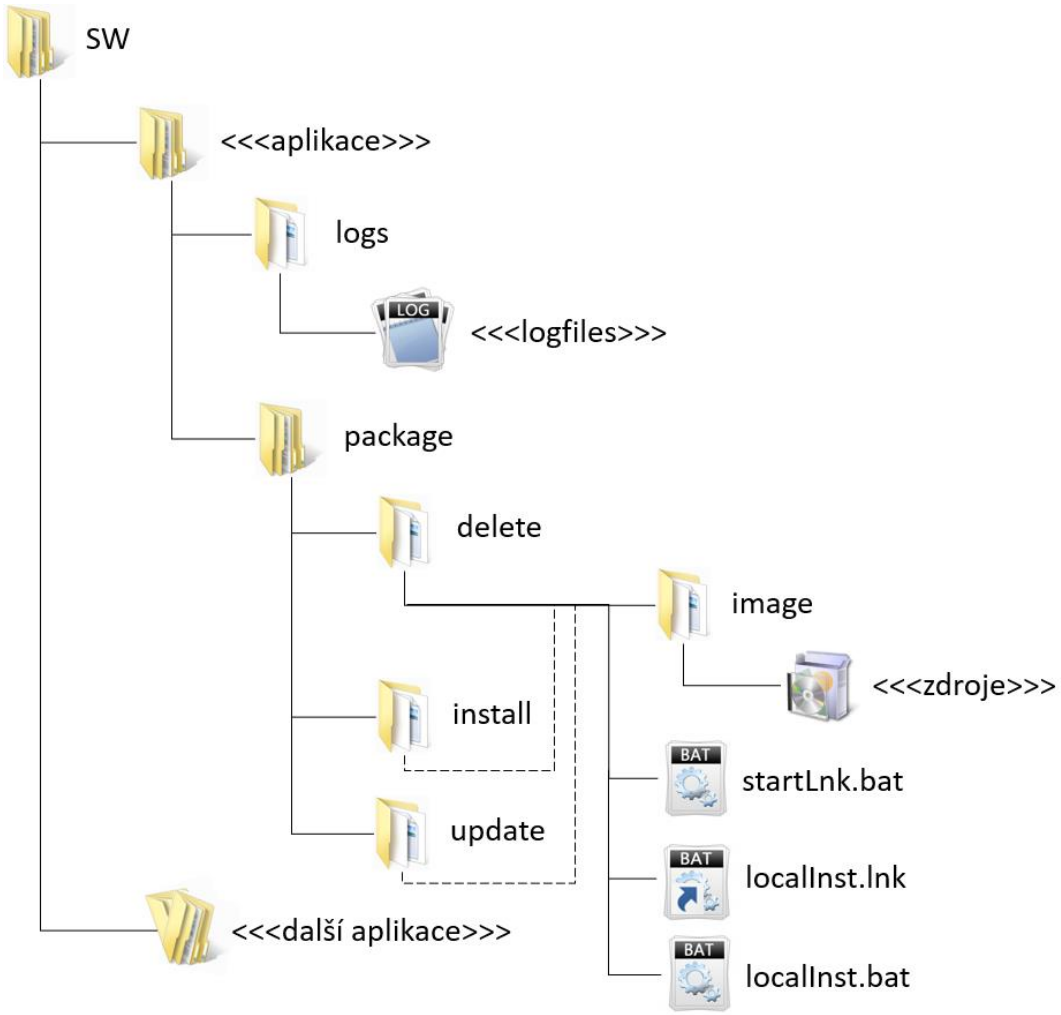
```
class CopyDir
{
    public void Copy(string sourceDirectory, string targetDirectory)
    {
        DirectoryInfo source = new DirectoryInfo(sourceDirectory);
        DirectoryInfo target = new DirectoryInfo(targetDirectory);
        CopyAll(source, target);
    }
    public void CopyAll(DirectoryInfo source, DirectoryInfo target)
    {
        if (Directory.Exists(target.FullName) == false)
        {
            Directory.CreateDirectory(target.FullName);
        }
        foreach (FileInfo fi in source.GetFiles())
        {
            fi.CopyTo(Path.Combine(target.ToString(), fi.Name), true);
        }
        foreach (DirectoryInfo diSourceSubDir in source.GetDirectories())
        {
             DirectoryInfo nextTargetSubDir =
             target.CreateSubdirectory(diSourceSubDir.Name);
             CopyAll(diSourceSubDir, nextTargetSubDir);
        }
    }
}
```

Zdroj: autor

5.4 Organizace dat

Pro centralizovanou správu zdrojů, instalačních skriptů a logů byl zvolen File server. Pro správnou funkci systému musí být, před spuštěním, připravena adresářová struktura. Obrázek 5.32 zobrazuje hierarchickou strukturu složek a souborů.

Obrázek 5.32: Organizace dat



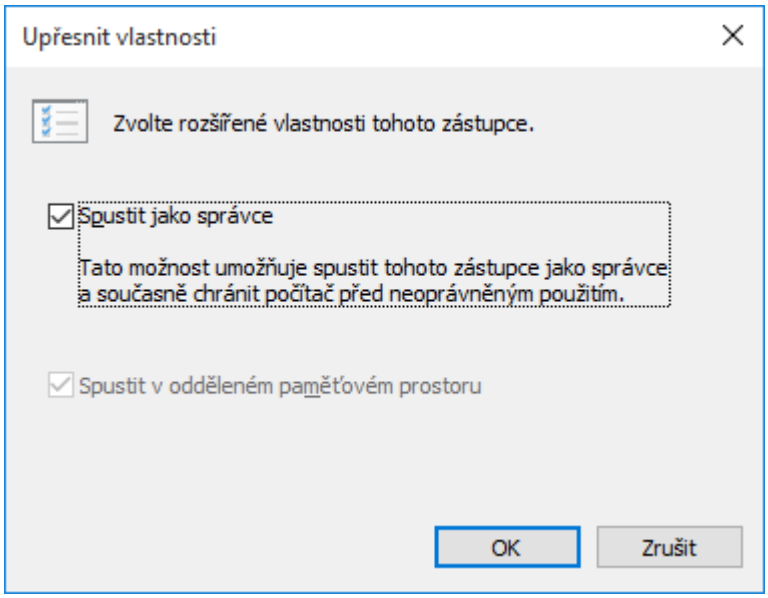
Zdroj: autor

Kořenový adresář SW musí být dostupný na definovaném místě. Názvy všech jeho podsložek se musí značit malými písmeny. Názvy aplikací musí odpovídat uvedeným názvům v databázi. Každý software obsahuje složky logs a package.

Ve složce logs jsou umístěny všechny záznamy o běhu aplikace. V názvu mají název počítače. Složka package obsahuje složky delete, install a update. Všechny tři složky mají analogicky shodnou strukturu obsahu a to složku image, která obsahuje potřebné zdroje pro provedení akce (například soubory typu msi nebo exe). Volitelně lze zdrojová data také umístit na jiný File server v síti, ale je nutné těmto zdrojům zajistit přidělení přístupových oprávnění pro účet, pod kterým se spouští výše popsaná služba. Složky dále obsahují 3 řídící soubory:

- **startLnk.bat** – Vstupní skript na stanici. Slouží pro spuštění zástupce localInst.lnk. Dále obsahuje příkaz timeout pro pětivteřinové zpoždění. To je nutné, aby stihl WMI proces začít sledovat stav spuštěného procesu na vzdálené stanici.
- **localInst.lnk** – Zástupce má za úkol přiřadit, dále spuštěnému řídícímu skriptu localInst.bat, administrátorská oprávnění na vzdálené stanici. Nastavení se provádí pomocí kontextové nabídky vyvolané stisknutím pravého tlačítka myši. Dále se zvolí položka Vlastnosti → Upřesnit vlastnosti. Zde zaškrtnout volbu Spustit jako správce (viz Obrázek 5.33). Tato práva je nutné přiřadit z důvodu zapnuté ochrany UAC (User Access Control).
- **localInst.bat** – Řídící skript. Vytváří správce aplikace a obsahuje pouze specifické příkazy pro jednotlivé aplikace (například kontrola běhového prostředí Java, případně jeho aktualizace). Minimálně musí skript obsahovat volání zdrojového souboru ve složce image nebo příkazy, které zařídí odinstalaci.

Obrázek 5.33: Nastavení administrátorských práv pro skript



Zdroj: autor

5.5 Testování

Pro úspěšné vytvoření systému je nutné myslet v průběhu vývoje i před dokončením na důkladné testování.

5.5.1 Metodika

Testování navrženého systému lze rozdělit do více fází a to:

- **Funkční testy** – Založené na jednotlivých požadavcích systému a uživatele.
- **Systémové testy** – Testování celého systému pomocí tzv. UseCase.
- **Zátěžové testy** – Testování stability systému v případě vysoké zátěže.
- **Stabilita systému** – Zajištění bezpečnosti dat v případech výpadku některé systémové komponenty.

5.5.2 Výsledky

Vytvořený prototyp byl nejprve otestován ve virtuálním prostředí a na závěr byl také otestován v IT oddělení firmy Škoda Auto a.s.

Prototyp aplikace byl průběžně při vývoji podroben funkčním testům, zda splňuje stanovené požadavky. V této fázi testování probíhalo pouze ve virtuálním prostředí.

Ve fázi systémových testů byly zvoleny komplexní scénáře (UseCase viz příloha C) dle výše aplikované analýzy procesu. Jedná o scénáře instalace, odinstalace, aktualizace, aktualizace software, aktualizace hardware. Další části systému byly otestovány postupem jednotlivých funkcí, tak jako ve fázi funkčních testů. V produktivním prostředí firmy byla možnost přistupovat k testovacímu vzorku 14 pracovních stanic umístěných napříč firmou ve více doménách.

Při zátěžových testech bylo do systému zadáno 200 požadavků na existující i neexistující stanice. Systém nevykazoval žádné zpomalení nebo nestabilitu.

Posledním typem testu bylo zajištění konzistenci systému i po výpadku jednotlivých komponent systému. Jako klíčová se v tomto případě ukázala služba. Při jejím pádu mohou zůstat spuštěné instalace a také má načtenou frontu požadavků. Testy prokázaly, že služba dokáže při opětovném spuštění adekvátně zpracovat i tyto požadavky.

Veškeré stanice byly díky systému úspěšně spravovány. Docházelo ke správnému vyhodnocení vstupních podmínek, přípravě zdrojových dat a následné akci. Na závěr došlo k aktualizaci databáze. Webové rozhraní odpovídá požadavkům a bylo plně funkční. Evidence byla také zcela funkční.

6 Výsledky a diskuse

Důležitým hlediskem pro verifikaci vytvoření odpovídajícího systému je kontrola naplněnosti identifikovaných požadavků:

- **Jednotná evidence počítačů** – realizována pomocí centrální databáze SQLite.
- **Jednotná evidence aplikací** – realizována pomocí centrální databáze SQLite.
- **Jednotná evidence aplikací instalovaných na počítače** – realizována pomocí centrální databáze SQLite.
- **Možnost vyhledávání v evidenci** – implementace pomocí ASP.NET.
- **Přehledné a intuitivní webové rozhraní** – implementace pomocí ASP.NET.
- **Rozhraní plnohodnotně dostupné bez ohledu na koncové zařízení** – implementace pomocí ASP.NET a Bootstrap.
- **Rozhraní umožňující řešit neúspěšné instalace** – implementace pomocí ASP.NET.
- **Náhrada duplicitních skriptů zajištujících základní přístupové kontroly** – implementace pomocí Windows služby.
- **Potřeba zajištění automatického zpracování požadavků** – implementace pomocí Windows služby.
- **Potřeba zajištění automatické aktualizace evidence** – implementace pomocí Windows služby.
- **Odstranění duplicitních činností administrátorů při správě software a hardware** – úprava procesu a implementace celého systému.
- **Centrální správa instalačních zdrojů** – realizováno pomocí souborové struktury na File serveru.
- **Nalezení stabilní metody přístupu ke vzdáleným stanicím** – řešení pomocí WMI.
- **Řešení využitelné pro počítače s operačním systémem Windows 7, Windows 8.1 a Windows 10** – řešení založené na .NET Framework.

Veškeré požadavky byly ve fázi návrhu zohledněny a dále byly implementovány pomocí výše uvedených technologií. Takto navržený, implementovaný a otestovaný systém naplňuje nově definovaný proces centrální správy software ve firmě.

7 Závěr

Záměrem diplomové práce bylo představit a popsat problematiku vývoje aplikačního software na platformě Windows s použitím .NET Framework, programovacího jazyka C# a vývojového prostředí Microsoft Visual Studio. Problematika teoretických východisek byla zpracována v kapitole 3.

Hlavním cílem diplomové práce bylo analyzovat problematiku správy software ve firemním prostředí a navrhnout systém pro centrální správu software ve firemním prostředí.

V kapitole 4 byl analyzován, jak proces správy firemního software obecně, tak konkrétně v IT oddělení firmy Škoda Auto a.s. zabývajícím se systémovou integraci a podporou konstrukčních a datových systémů. Na základě analýzy byly identifikovány přesné požadavky pro návrh a implementaci prototypu systému. Dále byl navrhnout nový optimalizovaný proces.

V kapitole 5 byla rozpracována implementace vlastního řešení. V první části byla navržena datová struktura a její implementace pomocí databáze SQLite. Další část obsahuje návrh a implementaci webové aplikace na platformě ASP.NET. Taktéž obsahuje návrh a implementaci celého jádra systému realizovaného pomocí služby systému Windows a napsané pro .NET Framework. V další části byla definována struktura zdrojových dat realizovaná pomocí hierarchické struktury složek na File serveru. Poslední část této kapitoly byla věnována důkladnému testování vytvořeného prototypu systému. Výsledky prokázaly funkčnost vytvořeného řešení a stabilitu běhu celého systému.

Kapitola 6 se zaměřuje na dosažené výsledky a jejich zhodnocení. V praxi bylo ověřeno, že vytvoření sjednoceného procesu a prototypu systému na základě znalostí, zpracované teoretické analýzy prostředí .NET a aktuálního procesu používaného ve firmě, je správnou cestou pro optimalizované řešení centrální správy software.

Na závěr je potřeba říct, že stanovené cíle této diplomové práce byly zcela naplněny a finální systém je funkční a odpovídá stanoveným požadavkům.

8 Seznam použitých zdrojů

1. **HERCEG, Tomáš.** Úvod do .NET Frameworku. *dotnetportal.* [Online] 3. Duben 2009. [Citace: 3. 02 2016.] Dostupný z: http://www.dotnetportal.cz/clanek/125/Uvod-do-NET-Frameworku.

2. **MASSI, Beth.** Understanding .NET 2015. *MSDN.* [Online] 25. 02 2015. [Citace: 15. 02 2016.] Dostupný z: https://blogs.msdn.microsoft.com/bethmassi/2015/02/25/understanding-net-2015/.

3. **MICROSOFT.** Začínáme s rozhraním .NET Framework. *MSDN.* [Online] 2016. [Citace: 11. 02 2016.] Dostupný z: https://msdn.microsoft.com/library/hh425099.aspx.

4. **BĚHÁLEK, Marek.** Programovací jazyk C#. *Katedra Informatiky - VŠB-TU Ostrava.* [Online] 2007. [Citace: 13. 02 2016.] Dostupný z: www.cs.vsb.cz/behalek/vyuka/pcsharp/text/.

5. **MICROSOFT.** .NET Framework Versions and Dependencies. *MSDN.* [Online] 2015. [Citace: 15. 02 2016.] Dostupný z: https://msdn.microsoft.com/en-us/library/bb822049(v=vs.110).aspx.

6. —. Přehled rozhraní .NET Framework. *MSDN.* [Online] 2016. [Citace: 2016. 02 18.] Dostupný z: https://msdn.microsoft.com/cs-cz/library/zw4w595w.aspx.

7. **TROELSEN, Andrew.** *C# a .NET 2.0 profesionálně.* Brno : Zoner press, 2006. str. 1200. ISBN 80-86815-42-0.

8. **GUTHRIE, Scott.** .NET Core. *.NETfoundation.* [Online] 2015. [Citace: 02. 16 2016.] Dostupný z: https://www.dotnetfoundation.org/netcore.

9. **MICROSOFT.** .NET – požadavky na systém. *MSDN.* [Online] 2016. [Citace: 16. 02 2016.] Dostupný z: https://msdn.microsoft.com/cs-cz/library/8z6watww(v=vs.110).aspx.

10. —. Nástroje pro každého vývojáře a každou aplikaci. *Visual Studio.* [Online] 2016. [Citace: 17. 02 2016.] Dostupný z: https://www.visualstudio.com/.

11. **ALBAHARI, Joseph, ALBAHARI, Ben a DRAYTON, Peter.** *Microsoft Visual C# 5.0 in a nuttshell.* Vyd. 5. Sebastopol : O'Reilly, 2012. ISBN 978-144-9320-102.

12. **SHARP, John.** *Microsoft Visual C# 2010.* Vyd. 1. Brno : Computer Press, 2010. ISBN 978-80-251-3147-3.

13. **NASH, Trey.** *C# 2010 Rychlý průvodce novinkami a nejlepšími postupy.* Brno : Computer Press, 2010. ISBN 978-80-251-3034-6.

14. **SPAANJAARS, Imar.** *Beginning ASP.NET 4.5 in C# and VB.* Indianapolis, Ind. : J. Wiley & Sons Inc., 2013. 978-1-118-31180-6.

15. **HANÁK, Ján .** *Základy paralelného programovania v jazyku C# 3.0* . Praha : Artax a.s, 2009. 978-80-87017-03-6.

16. **RUSSINOVICH, Mark.** PsExec v2.11. *Windows Sysinternals.* [Online] 02. 05 2014. [Citace: 21. 02 2016.] Dostupný z: http://technet.microsoft.com/en-us/sysinternals/bb897553.aspx.

17. **HIPP, Richard.** About SQLite. *SQLite.* [Online] 2016. [Citace: 20. 02 2016.] Dostupný z: www.sqlite.org/about.html.

18. —. SQLite Is Public Domain. *SQLite.* [Online] 2016. [Citace: 20. 02 2016.] Dostupný z: https://www.sqlite.org/copyright.html.

19. The Official home of the DB Browser for SQLite. *DB Browser for SQLite.* [Online] 2016. [Citace: 20. 02 2016.] Dostupný z: http://sqlitebrowser.org/.

20. About System.Data.SQLite. *SQLite.* [Online] 2016. [Citace: 24. 02 2016.] https://system.data.sqlite.org/.

21. **ČÁPKA, David.** Úvod do ASP.NET. *itnetwork.cz.* [Online] 2013. [Citace: 27. 02 2016.] http://www.itnetwork.cz/csharp/asp-net/tutorial-uvod-do-asp-dot-net/.

22. Bootstrap. [Online] 2016. [Citace: 23. 02 2016.] http://getbootstrap.com/.

23. **MICROSOFT.** Služba WMI - přehled. *Technet.* [Online] 2016. [Citace: 22. 02 2016.] Dostupný z: http://technet.microsoft.com/cs-cz/library/cc736575.

24. **DOSTÁLEK, Libor.** *Velký průvodce protokoly TCP/IP a systémem DNS.* Vyd. 2. Brno : Computer Press, 2000. ISBN 80-722-6323-4.

25. **KEJMAR, Jiří.** Návrh a implementace systému pro hromadnou správu SW. *Bakalářská práce (Bc.).* Praha : Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta, katedra informačního inženýrství, 2014.

9 Seznam zkratek

CIL – Common Intermediate Language

CLR – Common Language Runtime

CLS – Common Language Specification

CSS – Cascading Style Sheets

CTS – Common Type System

DHCP – Dynamic Host Configuration Protocol

DNS – Domain Name Server

DoS – Denial of Service

FIFO – First In - First Out

FQDN – Fully Qualified Domain Name

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

IIS – Internet Information Services

IP – Internet Protocol

JIT – Just In Time

LIFO – Last In - First Out

MSIL – Microsoft Intermediate Language

SLA – Service Level Agreement

UAC – User Access Control

WBEM – Web-Based Enterprise Management

WMI – Windows Management Instrumentation

XML – eXtensible Markup Language

10 Přílohy

A. LogFile stanice

- Stanice je nedostupná

```
2016-02-26 16:36:40 - >>>INSTALL>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2016-02-26 16:36:40 - Stanice není dostupná
```

- Na stanici probíhá jiná instalace

```
2016-02-26 16:38:25 - >>>INSTALL>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2016-02-26 16:38:25 - Na stanici již běží jiná instalace
```

- Úspěšná instalace

```
2016-02-26 16:40:40 - >>>INSTALL>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2016-02-26 16:40:41 - Workstation Network Information:
2016-02-26 16:40:41 - ---IP          : 10.217.160.83
2016-02-26 16:40:41 - ---DNS Fullname : SKDAMBLG1108.mb.skoda.vwg
2016-02-26 16:40:41 - ---IP Status    : Success
2016-02-26 16:40:41 - ---NetInfoError : Success
2016-02-26 16:40:41 - ---Workstation WMI Information:
2016-02-26 16:40:41 - ---Manufacturer & model             : Hewlett-Packard - HP ZBook 15
2016-02-26 16:40:41 - OS                                  : Microsoft Windows 7 Enterprise
2016-02-26 16:40:41 - ---OS Type                          : x64-based PC
2016-02-26 16:40:41 - ---HDD volne misto [GB] - RAM [GB]  : 95 - 15
2016-02-26 16:40:41 - ---WMI Error                        : Success
2016-02-26 16:40:41 - Kopirovani zdroje C:\DP\FILE\SW\connect\package\install\
2016-02-26 16:40:43 - Spusteni instalace
2016-02-26 16:42:24 - Odstranení zdroje
2016-02-26 16:42:24 - Instalace byla dokončena
```

- Úspěšná aktualizace

```
2016-02-26 16:51:10 - >>>UPDATE>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2016-02-26 16:51:11 - Workstation Network Information:
2016-02-26 16:51:11 - ---IP          : 10.217.160.83
2016-02-26 16:51:11 - ---DNS Fullname : SKDAMBLG1108.mb.skoda.vwg
2016-02-26 16:51:11 - ---IP Status    : Success
2016-02-26 16:51:11 - ---NetInfoError : Success
2016-02-26 16:51:11 - ---Workstation WMI Information:
2016-02-26 16:51:11 - ---Manufacturer & model             : Hewlett-Packard - HP ZBook 15
2016-02-26 16:51:11 - OS                                  : Microsoft Windows 7 Enterprise
2016-02-26 16:51:11 - ---OS Type                          : x64-based PC
2016-02-26 16:51:11 - ---HDD volne misto [GB] - RAM [GB]  : 95 - 15
2016-02-26 16:51:11 - ---WMI Error                        : Success
2016-02-26 16:51:11 - Kopirovani zdroje C:\DP\FILE\SW\connect\package\update\
2016-02-26 16:51:13 - Spusteni instalace
2016-02-26 16:53:04 - Odstranení zdroje
2016-02-26 16:53:05 - Instalace byla dokončena
```

- Uživatel nemá právo zápisu

```
2016-02-26 16:36:10 - >>>INSTALL>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2016-02-26 16:36:10 - Workstation Network Information:
2016-02-26 16:36:10 - ---IP          : 10.217.25.249
2016-02-26 16:36:10 - ---DNS Fullname : SKDAMBWG1678.mb.skoda.vwg
2016-02-26 16:36:10 - ---IP Status    : Success
2016-02-26 16:36:10 - ---NetInfoError : Success
2016-02-26 16:36:10 - ---Workstation WMI Information:
```

```
2016-02-26 16:36:10 - ---Manufacturer & model          : Hewlett-Packard - HP Z420 Workstation
2016-02-26 16:36:10 - OS                               : Microsoft Windows 7 Enterprise
2016-02-26 16:36:10 - ---OS Type                          : x64-based PC
2016-02-26 16:36:10 - ---HDD volne misto [GB] - RAM [GB]  : 130 - 16
2016-02-26 16:36:10 - ---WMI Error                        : Success
2016-02-26 16:36:10 - Kopirovani zdroje C:\DP\FILE\SW\connect\package\install\
2016-02-26 16:36:10 - Přístup k cestě \\SKDAMBWG1678\c$\temp\package\image\ExampleData byl
odepřen.
2016-02-26 16:36:10 - Uživatel nemá právo zápisu
```

B. LogFile služby

```
2016-02-26 16:30:23 - <<<<< START service >>>>>
2016-02-26 16:30:24 - queue state before load: 0 | 5
2016-02-26 16:30:24 - queue state after load: 0 | 5
2016-02-26 16:34:24 - queue state before load: 0 | 6
2016-02-26 16:34:24 - queue state after load: 22 | 6
2016-02-26 16:34:25 - SKDAMBWG1672 | CONNECT | install | 6
2016-02-26 16:34:25 - SKDAMBWG1672 - change state to running
2016-02-26 16:34:40 - SKDAMBWG1674 | CATIA | install | 7
2016-02-26 16:34:40 - SKDAMBWG1674 - change state to running
2016-02-26 16:34:55 - SKDAMBWG1678 | CREO | install | 8
2016-02-26 16:34:55 - SKDAMBWG1678 - change state to running
2016-02-26 16:34:58 - SKDAMBWG1678 - Uživatel nemá právo zápisu
2016-02-26 16:34:58 - SKDAMBWG1678 - change state to failed
2016-02-26 16:35:10 - SKDAMBWG1671 | ICEM | install | 8
2016-02-26 16:35:10 - SKDAMBWG1671 - change state to running
2016-02-26 16:35:25 - SKDAMBLG1108 | KVS | install | 5
2016-02-26 16:35:25 - SKDAMBLG1108 - change state to running
2016-02-26 16:35:28 - SKDAMBLG1108 - Stanice není dostupná
2016-02-26 16:35:28 - SKDAMBLG1108 - SKDAMBLG1108KVS - 1 (RetryIndex)
2016-02-26 16:35:28 - SKDAMBLG1108 - change state to retry
2016-02-26 16:35:40 - SKDATVWG0476 | CONNECT | install | 5
2016-02-26 16:35:40 - SKDATVWG0476 - change state to running
2016-02-26 16:35:40 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:35:40 - SKDATVWG0476 - SKDATVWG0476CONNECT - 1 (RetryIndex)
2016-02-26 16:35:40 - SKDATVWG0476 - change state to retry
2016-02-26 16:35:55 - SKDAMBWG1674 | CONNECT | install | 5
2016-02-26 16:35:55 - SKDAMBWG1674 - Na stanici již běží jiná instalace
2016-02-26 16:35:55 - SKDAMBWG1674 - SKDAMBWG1674CONNECT - 1 (RetryIndex)
2016-02-26 16:35:55 - SKDAMBWG1674 - change state to retry
2016-02-26 16:36:05 - SKDAMBWG1674 - Instalace byla dokončena
2016-02-26 16:36:05 - SKDAMBWG1674 - change state to finished
2016-02-26 16:36:05 - SKDAMBWG1674 - CATIA - Zaevidováno do DB
2016-02-26 16:36:10 - SKDAMBWG1678 | CONNECT | install | 5
2016-02-26 16:36:10 - SKDAMBWG1678 - change state to running
2016-02-26 16:36:10 - SKDAMBWG1678 - Uživatel nemá právo zápisu
2016-02-26 16:36:10 - SKDAMBWG1678 - change state to failed
2016-02-26 16:36:24 - SKDAMBWG1671 - Instalace byla dokončena
2016-02-26 16:36:24 - SKDAMBWG1671 - change state to finished
2016-02-26 16:36:24 - SKDAMBWG1671 - ICEM - Zaevidováno do DB
2016-02-26 16:36:25 - SKDAMBWG1671 | CONNECT | install | 5
2016-02-26 16:36:25 - SKDAMBWG1671 - change state to running
2016-02-26 16:36:38 - SKDAMBWG1672 - Instalace byla dokončena
2016-02-26 16:36:38 - SKDAMBWG1672 - change state to finished
2016-02-26 16:36:38 - SKDAMBWG1672 - CONNECT - Zaevidováno do DB
2016-02-26 16:36:40 - SKDAMBLG1108 | CONNECT | install | 8
2016-02-26 16:36:40 - SKDAMBLG1108 - change state to running
2016-02-26 16:36:40 - SKDAMBLG1108 - Stanice není dostupná
2016-02-26 16:36:40 - SKDAMBLG1108 - SKDAMBLG1108CONNECT - 1 (RetryIndex)
2016-02-26 16:36:40 - SKDAMBLG1108 - change state to retry
2016-02-26 16:36:55 - SKDATVWG0221 | CONNECT | install | 6
2016-02-26 16:36:55 - SKDATVWG0221 - change state to running
2016-02-26 16:36:55 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:36:55 - SKDATVWG0221 - SKDATVWG0221CONNECT - 1 (RetryIndex)
```

```
2016-02-26 16:36:55 - SKDATVWG0221 - change state to retry
2016-02-26 16:37:10 - SKDATVWG0136 | CONNECT | install | 6
2016-02-26 16:37:10 - SKDATVWG0136 - change state to running
2016-02-26 16:37:10 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:37:10 - SKDATVWG0136 - SKDATVWG0136CONNECT - 1 (RetryIndex)
2016-02-26 16:37:10 - SKDATVWG0136 - change state to retry
2016-02-26 16:37:25 - SKDATVWG0221 | CATIA | install | 6
2016-02-26 16:37:25 - SKDATVWG0221 - change state to running
2016-02-26 16:37:25 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:37:25 - SKDATVWG0221 - SKDATVWG0221CATIA - 1 (RetryIndex)
2016-02-26 16:37:25 - SKDATVWG0221 - change state to retry
2016-02-26 16:37:40 - SKDAMBWG1672 | CATIA | install | 6
2016-02-26 16:37:40 - SKDAMBWG1672 - change state to running
2016-02-26 16:37:55 - SKDAMBWG1672 | CREO | install | 9
2016-02-26 16:37:55 - SKDAMBWG1672 - Na stanici již běží jiná instalace
2016-02-26 16:37:55 - SKDAMBWG1672 - SKDAMBWG1672CREO - 1 (RetryIndex)
2016-02-26 16:37:55 - SKDAMBWG1672 - change state to retry
2016-02-26 16:38:08 - SKDAMBWG1671 - Instalace byla dokončena
2016-02-26 16:38:08 - SKDAMBWG1671 - change state to finished
2016-02-26 16:38:08 - SKDAMBWG1671 - CONNECT - Zaevidováno do DB
2016-02-26 16:38:10 - SKDAMBWG1672 | ICEM | install | 10
2016-02-26 16:38:10 - SKDAMBWG1672 - Na stanici již běží jiná instalace
2016-02-26 16:38:10 - SKDAMBWG1672 - SKDAMBWG1672ICEM - 1 (RetryIndex)
2016-02-26 16:38:10 - SKDAMBWG1672 - change state to retry
2016-02-26 16:38:24 - queue state before load: 6 | 5
2016-02-26 16:38:24 - queue state after load: 15 | 5
2016-02-26 16:38:25 - SKDAMBWG1672 | KVS | install | 9
2016-02-26 16:38:25 - SKDAMBWG1672 - Na stanici již běží jiná instalace
2016-02-26 16:38:25 - SKDAMBWG1672 - SKDAMBWG1672KVS - 1 (RetryIndex)
2016-02-26 16:38:25 - SKDAMBWG1672 - change state to retry
2016-02-26 16:38:40 - SKDAMBWG1678 | KVS | install | 5
2016-02-26 16:38:40 - SKDAMBWG1678 - change state to running
2016-02-26 16:38:40 - SKDAMBWG1678 - Uživatel nemá právo zápisu
2016-02-26 16:38:40 - SKDAMBWG1678 - change state to failed
2016-02-26 16:38:45 - SKDAMBWG1672 - Instalace byla dokončena
2016-02-26 16:38:45 - SKDAMBWG1672 - change state to finished
2016-02-26 16:38:45 - SKDAMBWG1672 - CATIA - Zaevidováno do DB
2016-02-26 16:38:55 - SKDATVWG0476 | CATIA | install | 9
2016-02-26 16:38:55 - SKDATVWG0476 - change state to running
2016-02-26 16:38:55 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:38:55 - SKDATVWG0476 - SKDATVWG0476CATIA - 1 (RetryIndex)
2016-02-26 16:38:55 - SKDATVWG0476 - change state to retry
2016-02-26 16:39:10 - SKDAMBLG1108 | CREO | install | 9
2016-02-26 16:39:10 - SKDAMBLG1108 - change state to running
2016-02-26 16:39:10 - SKDAMBLG1108 - Stanice není dostupná
2016-02-26 16:39:10 - SKDAMBLG1108 - SKDAMBLG1108CREO - 1 (RetryIndex)
2016-02-26 16:39:10 - SKDAMBLG1108 - change state to retry
2016-02-26 16:39:25 - SKDAMBWG1674 | ICEM | install | 9
2016-02-26 16:39:25 - SKDAMBWG1674 - change state to running
2016-02-26 16:39:40 - SKDAMBWG1678 | CATIA | install | 7
2016-02-26 16:39:40 - SKDAMBWG1678 - change state to running
2016-02-26 16:39:41 - SKDAMBWG1678 - Uživatel nemá právo zápisu
2016-02-26 16:39:41 - SKDAMBWG1678 - change state to failed
2016-02-26 16:39:55 - SKDAMBLG1108 | KVS | install | 7
2016-02-26 16:39:55 - SKDAMBLG1108 - change state to running
2016-02-26 16:40:10 - SKDATVWG0476 | CONNECT | install | 4
2016-02-26 16:40:10 - SKDATVWG0476 - change state to running
2016-02-26 16:40:10 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:40:10 - SKDATVWG0476 - SKDATVWG0476CONNECT - 2 (RetryIndex)
2016-02-26 16:40:10 - SKDATVWG0476 - change state to retry
2016-02-26 16:40:17 - SKDAMBWG1674 - Instalace byla dokončena
2016-02-26 16:40:17 - SKDAMBWG1674 - change state to finished
2016-02-26 16:40:17 - SKDAMBWG1674 - ICEM - Zaevidováno do DB
2016-02-26 16:40:25 - SKDAMBWG1674 | CONNECT | install | 4
2016-02-26 16:40:25 - SKDAMBWG1674 - change state to running
2016-02-26 16:40:38 - SKDAMBLG1108 - Instalace byla dokončena
2016-02-26 16:40:38 - SKDAMBLG1108 - SKDAMBLG1108KVS --> limitace odstraněna (RetryIndex)
```

```
2016-02-26 16:40:38 - SKDAMBLG1108 - change state to finished
2016-02-26 16:40:38 - SKDAMBLG1108 - KVS - Zaevidováno do DB
2016-02-26 16:40:40 - SKDAMBLG1108 | CONNECT | install | 11
2016-02-26 16:40:40 - SKDAMBLG1108 - change state to running
2016-02-26 16:40:55 - SKDATVWG0221 | CONNECT | install | 7
2016-02-26 16:40:55 - SKDATVWG0221 - change state to running
2016-02-26 16:40:55 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:40:55 - SKDATVWG0221 - SKDATVWG0221CONNECT - 2 (RetryIndex)
2016-02-26 16:40:55 - SKDATVWG0221 - change state to retry
2016-02-26 16:41:10 - SKDATVWG0136 | CONNECT | install | 7
2016-02-26 16:41:10 - SKDATVWG0136 - change state to running
2016-02-26 16:41:10 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:41:10 - SKDATVWG0136 - SKDATVWG0136CONNECT - 2 (RetryIndex)
2016-02-26 16:41:10 - SKDATVWG0136 - change state to retry
2016-02-26 16:41:25 - SKDATVWG0221 | CATIA | install | 7
2016-02-26 16:41:25 - SKDATVWG0221 - change state to running
2016-02-26 16:41:25 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:41:25 - SKDATVWG0221 - SKDATVWG0221CATIA - 2 (RetryIndex)
2016-02-26 16:41:25 - SKDATVWG0221 - change state to retry
2016-02-26 16:41:40 - SKDAMBWG1672 | CREO | install | 7
2016-02-26 16:41:40 - SKDAMBWG1672 - change state to running
2016-02-26 16:41:55 - SKDAMBWG1672 | ICEM | install | 13
2016-02-26 16:41:55 - SKDAMBWG1672 - Na stanici již běží jiná instalace
2016-02-26 16:41:55 - SKDAMBWG1672 - SKDAMBWG1672ICEM - 2 (RetryIndex)
2016-02-26 16:41:55 - SKDAMBWG1672 - change state to retry
2016-02-26 16:42:09 - SKDAMBWG1674 - Instalace byla dokončena
2016-02-26 16:42:09 - SKDAMBWG1674 - SKDAMBWG1674CONNECT --> limitace odstraněna (RetryIndex)
2016-02-26 16:42:09 - SKDAMBWG1674 - change state to finished
2016-02-26 16:42:09 - SKDAMBWG1674 - CONNECT - Zaevidováno do DB
2016-02-26 16:42:24 - queue state before load: 0 | 4
2016-02-26 16:42:24 - queue state after load: 8 | 4
2016-02-26 16:42:24 - SKDAMBLG1108 - Instalace byla dokončena
2016-02-26 16:42:24 - SKDAMBLG1108 - SKDAMBLG1108CONNECT --> limitace odstraněna (RetryIndex)
2016-02-26 16:42:24 - SKDAMBLG1108 - change state to finished
2016-02-26 16:42:24 - SKDAMBLG1108 - CONNECT - Zaevidováno do DB
2016-02-26 16:42:25 - SKDATVWG0476 | CONNECT | install | 13
2016-02-26 16:42:25 - SKDATVWG0476 - change state to running
2016-02-26 16:42:25 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:42:25 - SKDATVWG0476 - SKDATVWG0476CONNECT - 3 (RetryIndex)
2016-02-26 16:42:25 - SKDATVWG0476 - change state to retry
2016-02-26 16:42:40 - SKDATVWG0221 | CONNECT | install | 4
2016-02-26 16:42:40 - SKDATVWG0221 - change state to running
2016-02-26 16:42:40 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:42:40 - SKDATVWG0221 - SKDATVWG0221CONNECT - 3 (RetryIndex)
2016-02-26 16:42:40 - SKDATVWG0221 - change state to retry
2016-02-26 16:42:45 - SKDAMBWG1672 - Instalace byla dokončena
2016-02-26 16:42:45 - SKDAMBWG1672 - SKDAMBWG1672CREO --> limitace odstraněna (RetryIndex)
2016-02-26 16:42:45 - SKDAMBWG1672 - change state to finished
2016-02-26 16:42:45 - SKDAMBWG1672 - CREO - Zaevidováno do DB
2016-02-26 16:42:55 - SKDATVWG0136 | CONNECT | install | 4
2016-02-26 16:42:55 - SKDATVWG0136 - change state to running
2016-02-26 16:42:55 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:42:55 - SKDATVWG0136 - SKDATVWG0136CONNECT - 3 (RetryIndex)
2016-02-26 16:42:55 - SKDATVWG0136 - change state to retry
2016-02-26 16:43:10 - SKDATVWG0221 | CATIA | install | 4
2016-02-26 16:43:10 - SKDATVWG0221 - change state to running
2016-02-26 16:43:10 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:43:10 - SKDATVWG0221 - SKDATVWG0221CATIA - 3 (RetryIndex)
2016-02-26 16:43:10 - SKDATVWG0221 - change state to retry
2016-02-26 16:43:25 - SKDAMBWG1672 | ICEM | install | 13
2016-02-26 16:43:25 - SKDAMBWG1672 - change state to running
2016-02-26 16:43:40 - SKDAMBWG1672 | KVS | install | 8
2016-02-26 16:43:40 - SKDAMBWG1672 - Na stanici již běží jiná instalace
2016-02-26 16:43:40 - SKDAMBWG1672 - SKDAMBWG1672KVS - 2 (RetryIndex)
2016-02-26 16:43:40 - SKDAMBWG1672 - change state to retry
2016-02-26 16:43:55 - SKDATVWG0476 | CATIA | install | 8
2016-02-26 16:43:55 - SKDATVWG0476 - change state to running
```

```
2016-02-26 16:43:55 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:43:55 - SKDATVWG0476 - SKDATVWG0476CATIA - 2 (RetryIndex)
2016-02-26 16:43:55 - SKDATVWG0476 - change state to retry
2016-02-26 16:44:10 - SKDAMBLG1108 | CREO | install | 8
2016-02-26 16:44:10 - SKDAMBLG1108 - change state to running
2016-02-26 16:44:19 - SKDAMBWG1672 - Instalace byla dokončena
2016-02-26 16:44:19 - SKDAMBWG1672 - SKDAMBWG1672ICEM --> limitace odstraněna (RetryIndex)
2016-02-26 16:44:19 - SKDAMBWG1672 - change state to finished
2016-02-26 16:44:19 - SKDAMBWG1672 - ICEM - Zaevidováno do DB
2016-02-26 16:45:15 - SKDAMBLG1108 - Instalace byla dokončena
2016-02-26 16:45:15 - SKDAMBLG1108 - SKDAMBLG1108CREO --> limitace odstraněna (RetryIndex)
2016-02-26 16:45:15 - SKDAMBLG1108 - change state to finished
2016-02-26 16:45:15 - SKDAMBLG1108 - CREO - Zaevidováno do DB
2016-02-26 16:46:24 - queue state before load: 0 | 13
2016-02-26 16:46:24 - queue state after load: 6 | 13
2016-02-26 16:46:25 - SKDATVWG0476 | CONNECT | install | 13
2016-02-26 16:46:25 - SKDATVWG0476 - change state to running
2016-02-26 16:46:25 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:46:25 - SKDATVWG0476 - SKDATVWG0476CONNECT - 4 (RetryIndex)
2016-02-26 16:46:25 - SKDATVWG0476 - change state to retry
2016-02-26 16:46:40 - SKDATVWG0221 | CONNECT | install | 13
2016-02-26 16:46:40 - SKDATVWG0221 - change state to running
2016-02-26 16:46:40 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:46:40 - SKDATVWG0221 - SKDATVWG0221CONNECT - 4 (RetryIndex)
2016-02-26 16:46:40 - SKDATVWG0221 - change state to retry
2016-02-26 16:46:55 - SKDATVWG0136 | CONNECT | install | 13
2016-02-26 16:46:55 - SKDATVWG0136 - change state to running
2016-02-26 16:46:55 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:46:55 - SKDATVWG0136 - SKDATVWG0136CONNECT - 4 (RetryIndex)
2016-02-26 16:46:55 - SKDATVWG0136 - change state to retry
2016-02-26 16:47:10 - SKDATVWG0221 | CATIA | install | 13
2016-02-26 16:47:10 - SKDATVWG0221 - change state to running
2016-02-26 16:47:10 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:47:10 - SKDATVWG0221 - SKDATVWG0221CATIA - 4 (RetryIndex)
2016-02-26 16:47:10 - SKDATVWG0221 - change state to retry
2016-02-26 16:47:25 - SKDAMBWG1672 | KVS | install | 13
2016-02-26 16:47:25 - SKDAMBWG1672 - change state to running
2016-02-26 16:47:40 - SKDATVWG0476 | CATIA | install | 12
2016-02-26 16:47:40 - SKDATVWG0476 - change state to running
2016-02-26 16:47:40 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:47:40 - SKDATVWG0476 - SKDATVWG0476CATIA - 3 (RetryIndex)
2016-02-26 16:47:40 - SKDATVWG0476 - change state to retry
2016-02-26 16:48:07 - SKDAMBWG1672 - Instalace byla dokončena
2016-02-26 16:48:07 - SKDAMBWG1672 - SKDAMBWG1672KVS --> limitace odstraněna (RetryIndex)
2016-02-26 16:48:07 - SKDAMBWG1672 - change state to finished
2016-02-26 16:48:07 - SKDAMBWG1672 - KVS - Zaevidováno do DB
2016-02-26 16:50:24 - queue state before load: 0 | 12
2016-02-26 16:50:24 - queue state after load: 9 | 12
2016-02-26 16:50:25 - SKDAMBWG1672 | CONNECT | update | 12
2016-02-26 16:50:25 - SKDAMBWG1672 - change state to running
2016-02-26 16:50:40 - SKDAMBWG1671 | CONNECT | update | 8
2016-02-26 16:50:40 - SKDAMBWG1671 - change state to running
2016-02-26 16:50:55 - SKDAMBWG1674 | CONNECT | update | 15
2016-02-26 16:50:55 - SKDAMBWG1674 - change state to running
2016-02-26 16:51:10 - SKDAMBLG1108 | CONNECT | update | 16
2016-02-26 16:51:10 - SKDAMBLG1108 - change state to running
2016-02-26 16:51:26 - SKDATVWG0476 | CONNECT | install | 17
2016-02-26 16:51:26 - SKDATVWG0476 - change state to running
2016-02-26 16:51:26 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:51:26 - SKDATVWG0476 - SKDATVWG0476CONNECT - 5 (RetryIndex)
2016-02-26 16:51:26 - SKDATVWG0476 - change state to retry
2016-02-26 16:51:41 - SKDATVWG0221 | CONNECT | install | 17
2016-02-26 16:51:41 - SKDATVWG0221 - change state to running
2016-02-26 16:51:41 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:51:41 - SKDATVWG0221 - SKDATVWG0221CONNECT - 5 (RetryIndex)
2016-02-26 16:51:41 - SKDATVWG0221 - change state to retry
2016-02-26 16:51:56 - SKDATVWG0136 | CONNECT | install | 17
```

```
2016-02-26 16:51:56 - SKDATVWG0136 - change state to running
2016-02-26 16:51:56 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:51:56 - SKDATVWG0136 - SKDATVWG0136CONNECT - 5 (RetryIndex)
2016-02-26 16:51:56 - SKDATVWG0136 - change state to retry
2016-02-26 16:52:11 - SKDATVWG0221 | CATIA | install | 17
2016-02-26 16:52:11 - SKDATVWG0221 - change state to running
2016-02-26 16:52:11 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:52:11 - SKDATVWG0221 - SKDATVWG0221CATIA - 5 (RetryIndex)
2016-02-26 16:52:11 - SKDATVWG0221 - change state to retry
2016-02-26 16:52:19 - SKDAMBWG1672 - Instalace byla dokončena
2016-02-26 16:52:19 - SKDAMBWG1672 - change state to finished
2016-02-26 16:52:19 - SKDAMBWG1672 - CONNECT - Kombilace již evidována v DB
2016-02-26 16:52:26 - SKDATVWG0476 | CATIA | install | 17
2016-02-26 16:52:26 - SKDATVWG0476 - change state to running
2016-02-26 16:52:26 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:52:26 - SKDATVWG0476 - SKDATVWG0476CATIA - 4 (RetryIndex)
2016-02-26 16:52:26 - SKDATVWG0476 - change state to retry
2016-02-26 16:52:34 - SKDAMBWG1671 - Instalace byla dokončena
2016-02-26 16:52:34 - SKDAMBWG1671 - change state to finished
2016-02-26 16:52:34 - SKDAMBWG1671 - CONNECT - Kombilace již evidována v DB
2016-02-26 16:52:50 - SKDAMBWG1674 - Instalace byla dokončena
2016-02-26 16:52:50 - SKDAMBWG1674 - change state to finished
2016-02-26 16:52:50 - SKDAMBWG1674 - CONNECT - Kombilace již evidována v DB
2016-02-26 16:53:05 - SKDAMBLG1108 - Instalace byla dokončena
2016-02-26 16:53:05 - SKDAMBLG1108 - change state to finished
2016-02-26 16:53:05 - SKDAMBLG1108 - CONNECT - Kombilace již evidována v DB
2016-02-26 16:54:24 - queue state before load: 0 | 12
2016-02-26 16:54:24 - queue state after load: 5 | 12
2016-02-26 16:54:26 - SKDATVWG0476 | CONNECT | install | 12
2016-02-26 16:54:26 - SKDATVWG0476 - change state to running
2016-02-26 16:54:26 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:54:26 - SKDATVWG0476 - SKDATVWG0476CONNECT - 6 (RetryIndex)
2016-02-26 16:54:26 - SKDATVWG0476 - change state to retry
2016-02-26 16:54:41 - SKDATVWG0221 | CONNECT | install | 13
2016-02-26 16:54:41 - SKDATVWG0221 - change state to running
2016-02-26 16:54:41 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:54:41 - SKDATVWG0221 - SKDATVWG0221CONNECT - 6 (RetryIndex)
2016-02-26 16:54:41 - SKDATVWG0221 - change state to retry
2016-02-26 16:54:56 - SKDATVWG0136 | CONNECT | install | 12
2016-02-26 16:54:56 - SKDATVWG0136 - change state to running
2016-02-26 16:54:56 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:54:56 - SKDATVWG0136 - SKDATVWG0136CONNECT - 6 (RetryIndex)
2016-02-26 16:54:56 - SKDATVWG0136 - change state to retry
2016-02-26 16:55:11 - SKDATVWG0221 | CATIA | install | 13
2016-02-26 16:55:11 - SKDATVWG0221 - change state to running
2016-02-26 16:55:11 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:55:11 - SKDATVWG0221 - SKDATVWG0221CATIA - 6 (RetryIndex)
2016-02-26 16:55:11 - SKDATVWG0221 - change state to retry
2016-02-26 16:55:26 - SKDATVWG0476 | CATIA | install | 12
2016-02-26 16:55:26 - SKDATVWG0476 - change state to running
2016-02-26 16:55:26 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:55:26 - SKDATVWG0476 - SKDATVWG0476CATIA - 5 (RetryIndex)
2016-02-26 16:55:26 - SKDATVWG0476 - change state to retry
2016-02-26 16:58:24 - queue state before load: 0 | 12
2016-02-26 16:58:24 - queue state after load: 5 | 12
2016-02-26 16:58:26 - SKDATVWG0476 | CONNECT | install | 12
2016-02-26 16:58:26 - SKDATVWG0476 - change state to running
2016-02-26 16:58:27 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:58:27 - SKDATVWG0476 - SKDATVWG0476CONNECT - 7 (RetryIndex)
2016-02-26 16:58:27 - SKDATVWG0476 - change state to retry
2016-02-26 16:58:41 - SKDATVWG0221 | CONNECT | install | 12
2016-02-26 16:58:41 - SKDATVWG0221 - change state to running
2016-02-26 16:58:41 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:58:42 - SKDATVWG0221 - SKDATVWG0221CONNECT - 7 (RetryIndex)
2016-02-26 16:58:42 - SKDATVWG0221 - change state to retry
2016-02-26 16:58:56 - SKDATVWG0136 | CONNECT | install | 12
2016-02-26 16:58:56 - SKDATVWG0136 - change state to running
```

```
2016-02-26 16:58:57 - SKDATVWG0136 - Stanice není dostupná
2016-02-26 16:58:57 - SKDATVWG0136 - SKDATVWG0136CONNECT - 7 (RetryIndex)
2016-02-26 16:58:57 - SKDATVWG0136 - change state to retry
2016-02-26 16:59:11 - SKDATVWG0221 | CATIA | install | 14
2016-02-26 16:59:11 - SKDATVWG0221 - change state to running
2016-02-26 16:59:11 - SKDATVWG0221 - Stanice není dostupná
2016-02-26 16:59:11 - SKDATVWG0221 - SKDATVWG0221CATIA - 7 (RetryIndex)
2016-02-26 16:59:11 - SKDATVWG0221 - change state to retry
2016-02-26 16:59:26 - SKDATVWG0476 | CATIA | install | 14
2016-02-26 16:59:26 - SKDATVWG0476 - change state to running
2016-02-26 16:59:26 - SKDATVWG0476 - Stanice není dostupná
2016-02-26 16:59:27 - SKDATVWG0476 - SKDATVWG0476CATIA - 6 (RetryIndex)
2016-02-26 16:59:27 - SKDATVWG0476 - change state to retry
2016-02-26 17:00:11 - <<<<< STOP service >>>>>
```

C. UseCase

Níže jsou uvedeny testovací scénáře z pohledu uživatele využívané ve fázi systémového testování:

- Instalace

TYP	AKTIVITA	OČEKÁVANÝ VÝSLEDEK
PŘÍPRAVA	role uživatel, dostupný webový portál, dostupný počítač	
KROK 01	zobrazit stránku instalace	stránka instalace zobrazena
KROK 02	zvolit název počítače	název počítače zvolen
KROK 03	zvolit název aplikace	název aplikace zvolen
KROK 04	potvrdit požadavek	požadavek je odeslán, pole počítač a aplikace jsou prázdná
KROK 05	zobrazit stránku požadavky	stránka požadavky zobrazena
KROK 06	zkontrolovat existenci požadavku	v tabulce je uveden požadavek s názvy software, hardware, typ install a stav new nebo load
KROK 07	zkontrolovat existenci požadavku	změna stavu na running
KROK 08	zkontrolovat existenci požadavku	zmizení požadavku z fronty
VÝSLEDEK	zkontrolovat, zda se požadovaná aplikace skutečně nainstalovala	aplikace je na počítači

- Odinstalace

TYP	AKTIVITA	OČEKÁVANÝ VÝSLEDEK
PŘÍPRAVA	role uživatel, dostupný webový portál, evidovaný hw a sw, nainstalovaný počítač	
KROK 01	zobrazit stránku odinstalace	stránka odinstalace zobrazena
KROK 02	zvolit název počítače	název počítače zvolen
KROK 03	zvolit název aplikace	název aplikace zvolen
KROK 04	potvrdit požadavek	požadavek je odeslán, pole počítač a aplikace jsou prázdná
KROK 05	zobrazit stránku požadavky	stránka požadavky zobrazena
KROK 06	zkontrolovat existenci požadavku	v tabulce je uveden požadavek s názvy software, hardware, typ delete a stav new nebo load
KROK 07	zkontrolovat existenci požadavku	změna stavu na running
KROK 08	zkontrolovat existenci požadavku	zmizení požadavku z fronty
VÝSLEDEK	zkontrolovat, zda se požadovaná aplikace skutečně odinstalovala	aplikace již není na počítači

- Aktualizace

TYP	AKTIVITA	OČEKÁVANÝ VÝSLEDEK
PŘÍPRAVA	role uživatel, webový portál, evidovaný hw a sw, neaktualizovaný počítač	
KROK 01	zobrazit stránku aktualizace	stránka aktualizace zobrazena
KROK 02	zvolit název počítače	název počítače zvolen
KROK 03	zvolit název aplikace	název aplikace zvolen
KROK 04	potvrdit požadavek	požadavek je odeslán, pole počítač a aplikace jsou prázdná
KROK 05	zobrazit stránku požadavky	stránka požadavky zobrazena
KROK 06	zkontrolovat existenci požadavku	v tabulce je uveden požadavek s názvy software, hardware, typ update a stav new nebo load
KROK 07	zkontrolovat existenci požadavku	změna stavu na running

KROK 08	zkontrolovat existenci požadavku	zmizení požadavku z fronty
VÝSLEDEK	zkontrolovat, zda se požadovaná aplikace skutečně aktualizovala	aplikace je na počítači aktuální

- Aktualizace software

TYP	AKTIVITA	OČEKÁVANÝ VÝSLEDEK
PŘÍPRAVA	role uživatel, webový portál, evidovaný hw a sw, neaktualizované počítače	
KROK 01	zobrazit stránku aktualizace aplikace	stránka aktualizace aplikace zobrazena
KROK 02	zvolit název aplikace	název aplikace zvolen
KROK 03	zkontrolovat tabulku počítačů	tabulka zobrazena a obsahuje počítače
KROK 04	potvrdit požadavek	požadavek je odeslán, pole aplikace je prázdná, tabulka počítačů je skryta
KROK 05	zkontrolovat tabulku přidaných počítačů	tabulka zobrazena a obsahuje přidané počítače
KROK 06	zobrazit stránku požadavky	stránka požadavky zobrazena
KROK 07	zkontrolovat existenci zadaných požadavků	v tabulce je uvedeny požadavky s názvy software, hardware, typ update a stav new nebo load
KROK 08	zkontrolovat existenci požadavků	změna stavu na running
KROK 09	zkontrolovat existenci požadavků	zmizení požadavků z fronty
VÝSLEDEK	zkontrolovat, zda se požadovaná aplikace na počítačích skutečně aktualizovala	aplikace je na počítačích aktuální

- Aktualizace hardware

TYP	AKTIVITA	OČEKÁVANÝ VÝSLEDEK
PŘÍPRAVA	role uživatel, webový portál, evidovaný hw a sw, neaktualizované nebo nenainstalované aplikace na počítači	
KROK 01	zobrazit stránku aktualizace počítače	stránka aktualizace počítače zobrazena
KROK 02	zvolit název počítače	název počítače zvolen
KROK 03	zkontrolovat tabulku aplikací	tabulka zobrazena a obsahuje aplikace

KROK 04	potvrdit požadavek	požadavek je odeslán, pole počítač je prázdné, tabulka aplikací je skryta
KROK 05	zkontrolovat tabulku přidaných aplikací	tabulka zobrazena a obsahuje přidané aplikace
KROK 06	zobrazit stránku požadavky	stránka požadavky zobrazena
KROK 07	zkontrolovat existenci zadaných požadavků	v tabulce je uvedeny požadavky s názvy software, hardware, typ update a stav new nebo load
KROK 08	zkontrolovat existenci požadavků	změna stavu na running
KROK 09	zkontrolovat existenci požadavků	zmizení požadavků z fronty
VÝSLEDEK	zkontrolovat, zda se požadované aplikace na počítači skutečně aktualizovaly	aplikace jsou na počítači aktuální