Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Zpracování dat domácnostního šetření

Bc. Tomáš Klapka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Tomáš Klapka

Systémové inženýrství a informatika
Informatika

Název práce

Zpracování dat domácnostního šetření

Název anglicky

Processing of population questionnaire data

Cíle práce

Cílem práce je analýza, návrh a implementace aplikace pro Český statistický úřad.
Aplikace bude sloužit k aktualizaci kontaktních údajů respondentů, čistění a importu audit trail dat tazatelů a garantů kraje nebo ČR z výsledků domácnostního šetření do existující databáze, a jejich záloze na firemním serveru. Tato činnost bude prováděna automatickým procházením a hledáním požadovanách dat, jejich transformací a o činnostech bude uživatel informován v logu.

Metodika

Metodika diplomové práce je založena na studiu odborné literatury a alternativních informačních zdrojů. Zpracování informací je zaměřeno na popis použité technologie k vývoji vlastní aplikace.

Bude provedena analýza problematiky a návrh aplikace k řešení daného úkolu.

Vlastní práce bude realizována ve vývojovém prostředí Visual Studio v jazyce C# s využitím Entity Frameworku. Při vývoji praktické části bude využito standardů softwarového inženýrství.

Na základě teoretických poznatků a testování aplikace budou formulovány závěry.

Doporučený rozsah práce

50 – 60 stran

Klíčová slova

C#, WPF, Entity Framework, XAML, Visual Studio, .NET

Doporučené zdroje informací

MACDONALD, M. – Pro WPF in C# 2010: Windows presentation foundation in.NET 4. New York: N.Y.: Distributed to the book trade worldwide by Springer-Verlag, 2010. ISBN 1430272058.

SKEET, J. – C# in depth. Third editions. Shelter Island: NY: Manning, 2013. ISBN 978-16-17-29134-0.

VIRIUS, M. Cvičení k programování pro .NET. Praha: České vysoké učení technické v Praze, 2019. ISBN 978-80-01-06546-4.

VIRIUS, M. Programování pro .NET. V Praze: České vysoké učení technické, 2011. ISBN 978-80-01-04864-1.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 30. 03. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Zpracování dat domácnostního šetření" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 4. 4. 2020 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu mé práce Ing. Marku Píckovi Ph.D. za cenné rady a odbornou pomoc. Také bych rád poděkoval svému bývalému kolegovy Bc. Janovi Pospíchalovi za jeho odborné připomínky, rady a vedení ve zpracování této diplomové práce. V neposlední řadě bych poděkoval také mé rodině a přátelům, kteří mě v průběhu celého studia podporovali.

Zpracování dat domácnostního šetření

Abstrakt

Cílem práce je návrh a vývoj desktopové aplikace určené ke zpracování určitých dat tazatelského souboru výběrových šetření v domácnostech pro Český statistický úřad. Konkrétněji má zpracovat audittraily z tazatelských souborů, kontrolovat jejich formát pro chyby, nahrání těchto dat a aktualizování seznamu pagin a kontaktů na centrální databázi Oracle. Aplikace bude umožňovat toto zpracování dělat automaticky, aby zajistil možný chod na pozadí jinak využívané pracovní stanice. Veškerý záznam o chodu aplikace bude uchováván v logu aplikace a bude umět tento log exportovat do textového nebo Rich Text File (RTF) souboru.

Práce je rozdělena na dvě hlavní části, teoretickou a praktickou. Teoretická část se zabývá obecně technologiemi, které jsou využívané v praktické části. Praktická část se poté zaměřuje na vlastní analýzu, návrh a implementaci práce. Vývoj aplikace bude realizován pomocí technologie Windows Presentation Foundation (WPF) v programovacím jazyce C#. K vývoji bylo využito vývojové prostředí Visual Studio a PL/SQL Developer client Oraclu pro uložené procedury.

Klíčová slova: .NET, C#, SQL, desktopová aplikace, vývoj

Processing of population questionnaire data

Abstract

The goal of this thesis is design and development of desktop application designed to process specific data from files send by questioners of the population questionnaire. The application is being developed for Czech statistical office. More specificaly the data it's processing is audittail files, which are checked for possible errors in the format of the files, uploading the data and updating pagination and list of contacts onto the central Oracle database. Application will be able to do this processing automaticaly, reducing the involvement of user, so the work station can be used for other tasks. All the program's activity is recorded in a log, which can be exported to a Text File or Rich Text File.

The thesis is split into two parts, theoretical and practical. The theoretical part is delving into the general technologies used in the practical part. The practical part then focuses on analysis, design and implementation of the application. The develompent of which will be realized using Windows Presentation Foundation technology and C# programming language. For the development, Visual Studio and Oracle PL/SQL Developer client is used.

Keywords: .NET, C#, SQL, desktop application, development

Obsah

Seznam obrázků

Seznam použitých zkratek

IŠD – Výběrové integrované šetření v domácnosti
EHIS – European Health Interview Survey
SILC - Statistics on Income and Living Conditions

1 Úvod

Hlavním přínosem této práce je vytvoření aplikace založené na technologii Windows Presentation Foundation (WPF), která je vyvíjena pro Český statistický úřad. Jedná se o náhradu stávajícího řešení za nové, jehož důvodem je požadavek na zefektivnění, zrychlení a zautomatizování stávajících operací. Těmito šetřeními jsou například Výběrové integrované šetření v domácnostech (IŠD), Výběrové šetření o zdraví (EHIS) nebo Výběrové šetření příjmů a životních podmínek (SILC). Navrhovaná aplikace bude sloužit ke zpracování nových dat, aktualizace stávajících dat, čištění audittrail souborů jednotlivých tazatelských souborů výběrových šetření a jejich ukládání do centrální databáze.

Windows Presentation Foundation (WPF) bylo zvoleno, neboť je to moderní a dnes běžně využívaná technologie pro vývoj desktopových aplikací. Tento framework přišel s .NET Frameworkem verze 3.0 a Microsoft jej uvedl jako náhradu za starší Windows Forms, které není možné využít pro tvorbu aplikací například na tabletech nebo mobilních zařízeních.

S aplikací bude pracovat přímo zaměstnanec Českého statistického úřadu. Aplikace je uzpůsobena tak, aby vyžadovala co nejmenší možnou odezvu od uživatele a mohla fungovat na pozadí uživatelovi stanice, zatímco uživatel se bude věnovat jiné práci. Zároveň je tu však možnost si zvolit specifický soubor ke zpracování.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je analýza, návrh a implementace aplikace pro Český Statistický Úřad. Aplikace bude sloužit k aktualizaci kontaktních údajů respondentů, čištění a import audittrail dat tazatelů a garantů krajů nebo české republiky získané z domácnostního šetření. Import bude provádět do Oracle databáze a bude prováděna záloha na firemním diskovém serveru. Tato činnost bude prováděna automatickým procházením a hledáním požadovaných dat v souborech tazatelů a garantů na pozadí počítače, aktualizací stávajících Oracle tabulek, hledáním chyb v textu, jejich transformací do databázových tabulek a následných nahrání do Oracle databáze. O této činnosti bude uživatel informován v logu.

2.2 Metodika

Metodika diplomové práce je založena na studiu odborné literatury a alternativních informačních zdrojů. Zpracování informací je zaměřeno na popis použitých technologií k vývoji vlastní aplikace.

Bude provedena analýza problematiky a návrh aplikace k řešení daného úkolu. Vlastní práce je realizováno vývojem WPF aplikace ve vývojovém prostředí Visual Studio v jazyce C#. Pro mou práci je využita řada technologií k zajištění správné funkčnosti aplikace. Pro práci s databází využívám OracleDataAdapter a OracleCommand knihoven. Pro bezpečnost a práci s hesly využívám Microsoft knihoven IsolatedStorage a Cryptography.

Při vývoji praktické části bude využito standardů softwarového inženýrství a na základě teoretických poznatků a testování aplikace budou formulovány závěry.

3 Teoretická východiska

Tato kapitola popisuje teoretická východiska, které slouží k následné tvorbě praktické části diplomové práce, která je zaměřena na vytvoření samotné aplikace. Jedná se především o popisu vývojového prostředí Microsoft .NET, týkající se hlavně programovacího jazyka C#, Windows Presentation Foundation, značkovací jazyk XAML a Entity Framework, který využívám pro manipulaci s Oracle databází.

3.1 Microsoft .NET platforma

Mluvíme zde o celé škále produktů společnosti Microsoft. Základem všech produktů je .NET Framework, který spolu s Microsoft Visual Studio tvoří komplexní vývojové prostředí, které lze využít pro vývoj desktopových, webových i mobilních zařízení. Platforma umožňuje vývoj v řadě programovacích jazyků, například C#, C++, Visual Basic nebo F#. [6]

Důležitá vlastnost .NET platformy je standardizovaná specifikace Common Language Infrastructure (CLI), která popisuje prostředí, v kterém lze využít více vysokoúrovňových programovacích jazyků na různých platformách. [7]

3.1.1 .NET Framework

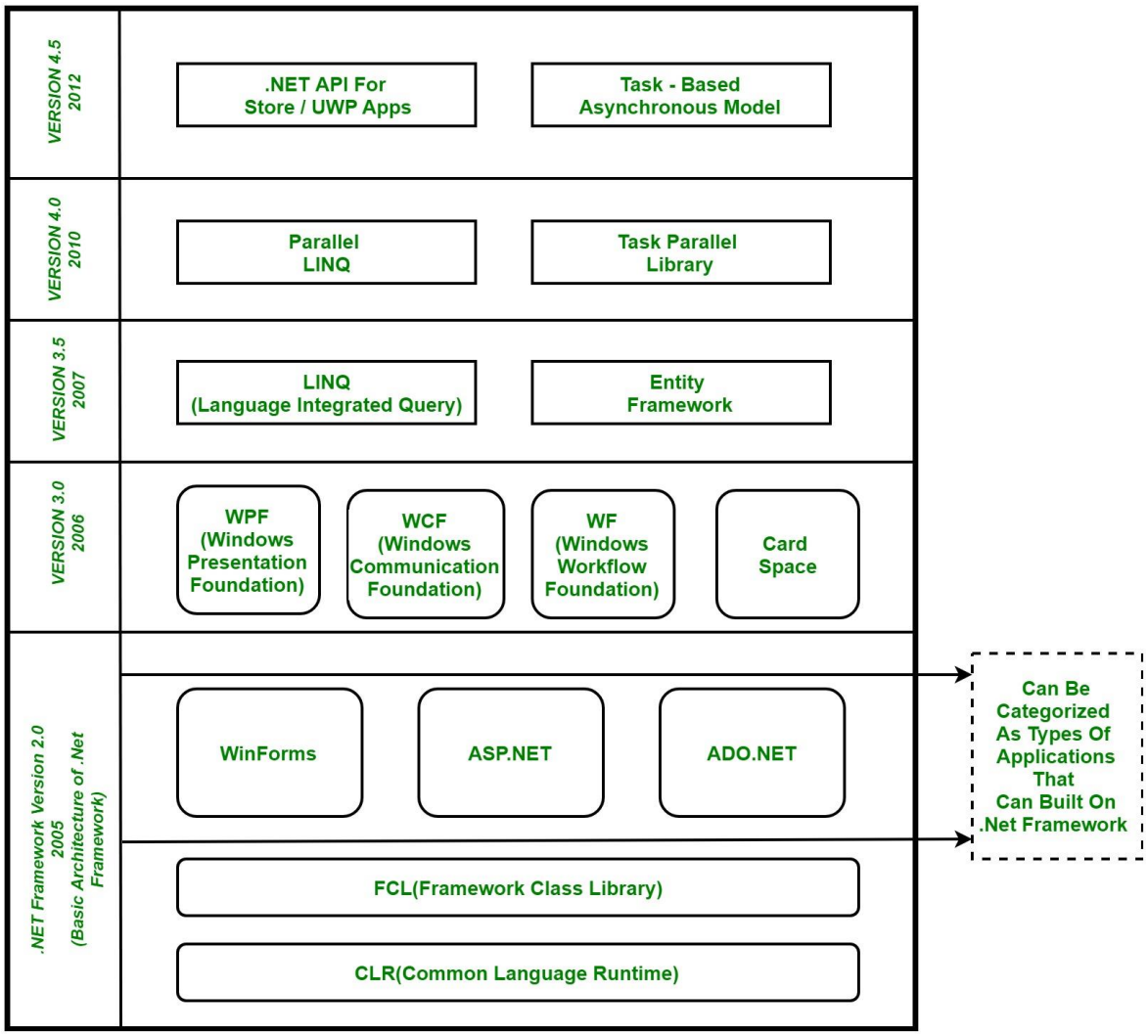
Jedná se o software framework vyvíjený společností Microsoft, který podporuje vytváření a pouštění aplikací a webové služby XML (eXtensible Markup Language).

Skládá se z potřebné sady knihoven Framework Class Library (FCL) a prostředí pro běh aplikací Common Language Runtime (CLR).

FCL je komplexní kolekcí objektově orientovaných a znovu využitelných prvků, které se dají využít při vývoji aplikace. Může se jednat o práci s konzolí, prvky grafického rozhraní (GUI) nebo práci s databázemi. [8]

CLR implementuje část CLI o využitelnosti libovolného podporovaného jazyka. Je to virtuální stroj, kde je kód zkompilován do mezikód přes Common Intermediate Language (CIL), kterému se říká spravovaný kód a je poté relativně rychle interpretován za běhu aplikace díky jeho jednoduchosti. Také poskytuje základní služby jako správu paměti, vynucuje přísné zabezpečení a další formy přesnosti kódu, podporující bezpečnost a robustnost. [9]

Aplikace vyvinuté v .NET vyžadují nainstalování stejné nebo vyšší verze .NET Frameworku, pro který byla aplikace vyvinuta. Operační systémy Windows běžně přicházejí přímo v základní instalaci s .NET Frameworkem, který je poté automaticky aktualizován automatickými systémovými aktualizacemi. [8]



Obrázek 1 – C# | .NET Framework (Basic Architecture and Component Stack)

3.1.2 Common Language Runtime

Modul Common Language Runtime vychází ze specifikace CLI, tedy standardu ECMA-335. Na jeho vývoji se podílelo mnoho významných firem, například Intel, Novell nebo Microsoft. Jedná se o virtuální stroj, který umožňuje použití více vysokoúrovňových jazyků na různých platformách, aniž by bylo nutno upravovat kompilátory danému prostředí. Kód je zde kompilován pomocí Common Intermediate Language na takzvaný spravovaný kód, který běží v modulu Runtime.

Spravovaný kód není nikdy interpretován, k tomu slouží kompilátor Just-in-time. Kompilátor také vytváří ze spravovaného kódu meta data, které popisují typy proměnných a odkazy na jiné objekty. Meta data jsou ukládána spolu se spravovaným kódem, ty jsou využívána k vyvolávání metod, generování nativního kódu, vyšší bezpečnosti a vymezení kontextu.

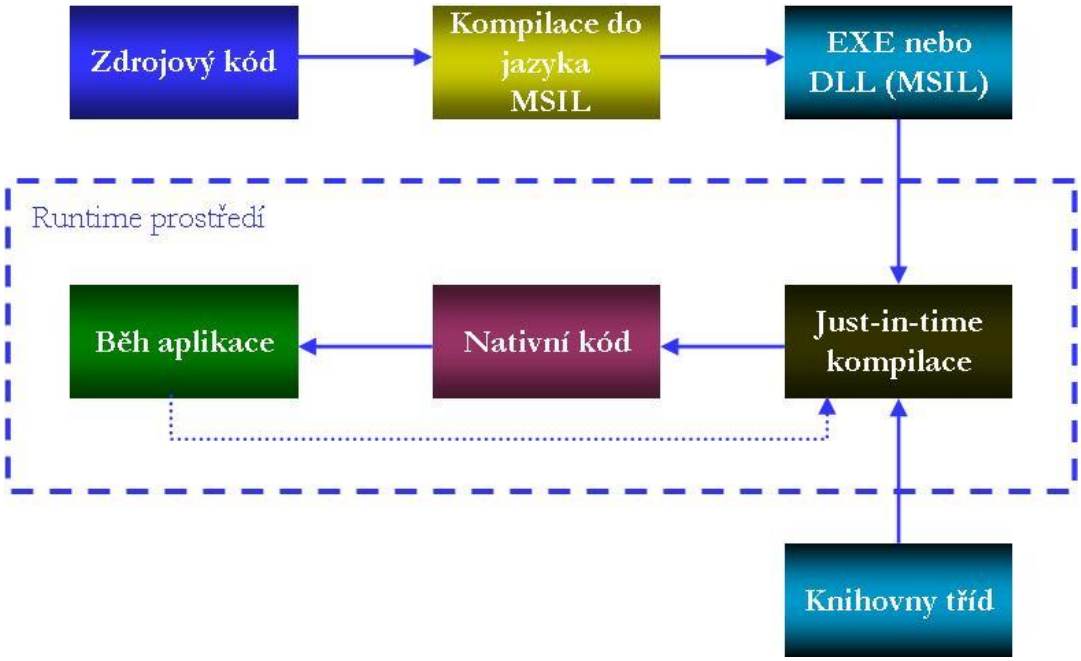
Runtime dále také automaticky spravuje rozložení objektů v paměti, jejich reference a uvolnění z paměti v případě nevyužívaných objektů. Eliminuje tak nedostatek paměti a běžné programovací chyby jako například úniky paměti nebo neplatné odkazy paměti.

Tento modul zvyšuje kompatibilitu a efektivitu vývojářů, protože lze vyvíjet v libovolném podporovaném jazyce a zároveň využívat knihovny či komponenty, psané v jakémkoliv podporovaném jazyce. Jde například definovat třída v jednom jazyce a následně ji odvodit nebo zavolat její metodu v jazyce jiném.

Je vynucována robustnost kódu, která je zajištěna implementací Common Type Systém (CTS), což je infrastruktura type-and-code ověření. To zajišťuje sebepopisovatelnost spravovaného kódu. Kompilátory na základě specifikace CTS tento kód generují, spravovaný kód tedy může využívat jiné spravované typy a instance. Zároveň to vynucuje typovou bezpečnost.

Součást všech meta dat je také informace o zdrojích, ze kterých byly spravované komponenty vytvořeny. Runtime tyto informace využívá k zajištění, zda aplikace má k dispozici všechny vyžadované verze zdrojů potřebných ke svému běhu. To snižuje náchylnost kódu k chybě vzniklé z nespecifikovaných závislostí.

Vzájemná kooperace spravovaného a nespravovaného kódu umožňuje používat komponenty COM a knihovny DLL. [9]



Obrázek 2 – .NET Framework | Common Language Runtime

3.1.3 C#

Jedná se o vysokoúrovňový objektově orientovaný programovací jazyk, který byl vyvinut společností Microsoft spolu s platformou .NET Framework. Je založen na svém předchůdci C++ z kterého čerpá syntaxe a na jazyku Java. C# byl navržen, aby co nejvíce bral v úvahu strukturu Common Language Runtime. Většina základních typů odpovídá základním typům platformy CLI. Lze ho mimo jiné využít právě pro databázové programy a formulářové aplikace na operačním systému Windows. C# má hierarchický přístup k dědění avšak neumožňuje mnohonásobné dědění. [34]

Mezi některé významné funkce, kterými se C# liší od ostatních jsou například [26]:

- Přenosnost, C# implicitně bere v úvahu standardy CLI, avšak nevyžaduje nutně použití CLR. Teoreticky je možné pomocí C# kompilátoru generovat strojový kód, avšak v praxi všechny překladače jazyka C# generují CLI.
- Deklarace proměnných, C# například umožňuje deklaraci implicitní variabilní proměnné *var nebo* implicitního pole díky klíčovému slovu new[]. Také má vyšší bezpečnost proti chybám způsobených rozdílnými datovými typy.
- Language Integrated Query (LINQ), umožňuje provádět dotazovací příkazy na libovolném XML dokumentu, IEnumerable<T> objektu, ADO.NET data setu či SQL databázi.

- Polymorfismus, C# neumožňuje dědění z několika předků, avšak třída může dědit libovolný počet rozhraní. C# však umožňuje přetížení libovolných operátorů.

3.1.3.1 Definice současného designu C#

Podle standardu ECMA je současný design C# definován [21]:

- C# je jednoduchý, moderní, mnohoúčelový a objektově orientovaný programovací jazyk.
- Jazyk a jeho implementace poskytuje podporu pro principy softwarového inženýrství jako jsou: hlídání hranic polí, detekce použití neinicializovaných proměnných a automatický garbage collector. Důležitost je kladena také na robustnost, trvanlivost a programátorskou produktivitu.
- Jazyk je vhodný pro vývoj softwarových komponent distribuovaných v různých prostředích.
- Přenositelnost zdrojového kódu je nesmírně důležitá, převážně pro programátory se znalostmi jazyků C a C++.
- Taktéž velmi důležitá je mezinárodní podpora.
- C# je navržen pro vývoj aplikací , pro zařízení se sofistikovanými operačními systémy tak i pro malá zařízení s omezenými a specifickými funkcemi.
- Přestože by aplikace psané v jazyce C# jsou zamýšlené k tomu být ekonomické s přidělenou pamětí a procesorovým časem, jazyk nebyl tvořen k tomu, aby konkuroval co se týče výkonu a velikostí s jazyky C a Assembler.

3.1.3.2 Vznik

Během vývoje .NET Frameworku byl původně využit spravovaný kód za použití systému kompilace nazývaný Simple Managed C. V roce 1999 Anders Hejlsberg, který se mimo jiné podílel na vývoji jazyků Turbo Pascal, Delphi a J++, však sestavil tým za účelem vývoje nového programovacího jazyka, v té době zvaný Cool (C-like Object Oriented Language). Ten byl v roce 2000 při jeho představení přejmenován na C# z důvodu autorských práv. Veškeré knihovny a ASP.NET Runtime modul byly přeloženy do jazyka C#. [34]

3.1.3.3 Historie verzí C#

V této kapitole chci popsat jednotlivé verze C# a nové služky a featury, s kterými tyto verze přišli [26]:

- C# 1.0 byla vydaná v roce 2002 společně s .NET Framework 1.0. Obsahovala základní podporu objektového programování vycházejícího z jazyka C++ a jazyku Java.
- C# 2.0 vyšla v roce 2005. Byla přidávana nativní podpora generik na úrovni CLI, parciální a statické třídy, iterátory, anonymní metody pro užívání delegátů a nulovatelné hodnotové typy.
- C# 3.0 vyšla v roce 2007 spolu s .NET Framework 3.5 a Visual Studio 2008. Přišla s řadu razantními změnami, které ustanovili C# jako impozantní jazyk. Mezi ně se řadí hlavně:
    - LINQ (Language Integrated Query) zavedl několik klíčových slov z dotazovacího jazyka SQL.
    - Zjednodušenní anonymních metod pomocí Lambda výrazů.
    - Inicializátory objektů a kolekcí.
    - Anonymní třídy umožňující rychlé tvoření objektů přenášející informace vyžádané z LINQ.
    - Klíčové slovo var, nutné pro využití anonymních tříd.
    - Rozšiřující metody.
    - Výrazové stromy, které umožňují za určitých podmínek kompilátoru místo vyhodnocení výrazu vytvořit jeho objektovou reprezentaci.
- C# 4.0 bylo vydáno spolu s Visual Studio 2010. Tato verze přinesla dynamické bindování, pojmenované a volitelné argumenty, kovariance a kontravariance v obecných typech a ekvivalenci typů.
- C# 5.0 byla vypuštěna spolu s Visual Studio 2012. Tato verze sebou přinesla především asynchronní metody s klíčovými slovy async a await.
- C# 6.0 přišela spolu s Visual Studio 2015. Tato verze byla především zaměřena na malé změny posilující produktivitu programátorů. Jedná se například o filtry vyjímek, možnost používají await v blocích catch a finally, auto-properties, zlepšení konstruktorů a možnost importování statických metod pomocí klíčového slova using.
- C# 7.0 vyšla v roce 2017. Přinesla s sebou například klíčové slovo throw pro vyvolávání vyjímek kdekoliv v kódu, zobecnění návratových typů asynchronních metod, proměnné a návratové hodnoty mohou být referenční a klíčové slovo out pro argumenty metody.

- C# 8.0 vyšla spolu s Visual Studio 2019. Přidány byly například readonly modifikátory proměnných, using proměnných (ty jsou na konci bloku zahozeny) nebo asynchronní streamy.

3.1.3.4 Syntaxe

Následující je jednoduchá C# konzolová aplikace Hello World, na které popíši základní syntaxi C#:

```
using Systém;

namespace UkazkaKonzole
{
    class Program
    {
        static void Main(string[] args)
        {
            // Toto je komentář, také lze využít /* a */ pro víceřádkový.
            Console.WriteLine(„Ahoj, světe!“);
        }
    }
}
```

Na začátku zdrojového kódu jsou uvedeny namespace s klíčovým slovem using, které budeme využívat pro sovu práci. Namespace nám potom udává jmenný prostor našeho vlastního projektu. Další řádek nám definuje třídu Program. Třídy jsou základní jednotky objektového programování a jsou rozděleny pro lepší orientaci a jednoznačnost názvů do jmenných prostorů (namespace). Zajímavostí C# oproti jiným jazykům může být to, že vše je uvnitř nějaké třídy. Veškeré bloky příkazů se utvářejí pomocí složených závorek {}.

Třída program obsahuje metodu s názvem Main. Klíčové slovo void uvádí, že metoda nemá žádnou návratovou hodnotu. Tato metoda také slouží jako spustitelný vstupní bod aplikace. Klíčové slovo static poté uvádí, že metoda je statická, tedy ji lze volat bez vytváření instance třídy Program.

Argumenty metody se udávají v jednoduchých závorkách (). Stejně jako při deklarování proměnných se uvádí typ proměnné, v tomto případě se jedná o pole řetězců (stringů) s názvem args, neboť pole značíme hranatými závorkami za typem proměnné [].

V těle metody na prvním řádku předvádím komentáře. To je část kódu, která nebude překládána, je často využitá například k popisu kódu. Komentáře na jeden řádek začínají dvěma lomítky // nebo jeden blok komentáře na více řádků začíná /* a končí */. Na druhém řádku předvádím volání statické metody třídy Console, která sídlí v namespace System. Jedná se o metodu WriteLine, která má parametr typu string, který vypíše uživateli do konzolového

pole při spuštění aplikace. V tomto případě místo proměnné vkládám argument přímo řetězec, který se ohraničuje uvozovkami.

Většina příkazů je ukončena středníkem ;. Výhoda takového ukončování je, že příkaz může být na libovolný počet řádků a překladač vždy pozná, kdy příkaz končí. [2]

3.1.3.5 Async/Await

Klíčová slova async a await, které přišli spolu s C# verze 5.0 se často využívají pro oddělení výpočtu business logiky aplikace od uživatelského prostředí. To nám zamezí uzamčení celého uživatelského rozhraní z důvodu probíhajícího výpočtu a nebo nám umožní provádět více výpočtů paralelně. Tyto syntaxe se nejčastěji využívají právě pro aplikace s grafickým rozhraním založené na XAML. Tedy WPF a .NET Framework, Silverlight nebo Windows Phone. Asynchronní metodu poznáme tak, že její návratová hodnota je Task nebo Task<typ>. To je datový typ reprezentující asynchronní operace a všechny dřívější reprezentace asynchronních operací, jako například BackgroundWorker, jsou na tuto reprezentaci principiálně převeditelné. [29]

Taková metoda vypadá takto:

```
Private Task<string> AsyncString()
 {
     return Task.Run(() =>
     {
         return "Návratový řetězec";
     });
 }
```

V tomto případě se jedná o úlohu (task), která po svém ukončení vrací řetězec „Návratový řetězec“ typu string. Synchronní blok kódu uvnitř této metody obalíme pomocí klíčového slova Task.Run(() => …), které způsobí vykonávání tohoto kódu na threadpoolu. Threadpool je skupina před-inicializovaných vláken, které čekají na nějaké potřebné operace a zamezuje tak zbytečnému tvoření nových vláken. [31]

Klíčové slova async a await poté slouží jako alternativní způsob zápisu vlastní implementace těla asynchronních metod. Taková implementace by vypadala následujícím způsobem:

```
private async void Button1_Click(object sender, RoutedEventArgs e)
 {
     this.IsEnabled = false;
     IsBusy = true;
     try
     {
         await Zpracovani();
     }
     finally
     {
         this.IsEnabled = true;
         IsBusy = false;
     }
 }
 private async Task Zpracovani()
 {
     textBlock.Text = "Probíhá výpočet...";
     //Asynchronní výpočet
     int vysledek = await AsyncString();
     //Aktualizace uživatelského rozhraní
     textBlock.Text = vysledek.ToString();
 }
```

Jedná se pouze o jednoduchou ukázku užití asynchronních metod. Tlačítkem spouštíme v příkladu uvedený asynchronní výpočet, po jehož dokončení je jeho výsledek zobrazen do textového pole uživatelského rozhraní. Během provádění tohoto výpočtu vidíme v textovém poli pouze informaci o provádějícím se výpočtu. Samozřejmě v tomto případě by výsledek byl zobrazen okamžitě, ale pokud by výpočet byl složitější nebo na okamžik pozastavoval práci vlákna, byla by tato posloupnost na první pohled viditelná. [31]

Nejdříve bych ještě rád shrnul hlavní vlastnosti těchto dvou klíčových slov [31]:

- Async – Tímto klíčovým slovem označujeme v deklaraci metodu, v jejímž těle budeme využívat alespoň jednou klíčové slovo await. Tím budeme tuto metodu implementovat jako kompozici běhu asynchronního workflow sestavenou z volání jiných existujících asynchronních metod. To zapříčiní změnu způsobu předkladu této metody do stavového automatu, který dále zajišťuje spouštění těch částí kódů, které jsou za nebo mezi jednotlivými klíčovými slovy await jako jednotlivé asynchronní volání metody. Metody, které jsou označené slovem async musí mít návratový typ Task nebo Task<T>, případě void. Příkaz return v asynchronní metodě určuje okamžik, ve kterém je asynchronní operace dokončena a v případě návratového typu Task<T> je na výstupu

k dispozici právě hodnota typu T, která se dostane z výstupu asynchronní metody.

- Await – toto klíčové slovo uvádíme před volání asynchronní metody, pokud chceme, aby se běh dalšího kódu umístěného za klíčovým slovem await prováděl až ve chvíli, kdy je asynchronní operace dokončena. Díky tomu jsou v kódu za klíčovým slovem await k dispozici výsledky z prováděného asynchronního volání metody, tedy pokud návratová hodnota metody je Task<T> a je k dispozici hodnota typu T. Klíčové slovo await lze uvést pouze před výraz typu Task nebo Task<T>.

Metoda Zpracovani z příkladu slouží jako pomocná asynchronní metoda, která vrací typ Task, jelikož představuje úlohu, která po svém dokončení nevrací žádnou hodnotu. Chceme však umět reagovat na její dokončení, k tomu slouží implementování metody pomocí nových klíčových slov async/await. Klíčové slovo await, kterou uvádíme před samotným voláním metody AsyncString, vyčká na samotný výpočet této metody a teprve po jeho dokončení provede aktualizaci uživatelského rozhraní výpisem výsledné hodnoty.

Metoda Button1_Click je event handler WPF komponenty buttonu. Ta využívá způsob implementace async/await, avšak nemá návratovou hodnotu. Proto se nejedná přímo o asynchronní metodu, na kterou by šlo nějakým způsobem reagovat. Zde je uveden async v hlavičce metody za účelem umožnit metodě využít klíčové slovo await. Toto je však zcela validní postup využití. [31]

Průběh jednotlivých akcí v tomto příkladu je tedy:

1. Vyvolání procedury event. Handleru Button1_Click.
2. Synchronní provedení kódu před první await a zavolá se metoda Zpracovani.
3. Synchronní provedení kódu metody Zpracovani do prvního await, zobrazí se v textovém poli „Probíhá výpočet…“.
4. Metoda AsyncString se spustí na pomocném vlákně threadpoolu a ihned vrátí Task<string> představující výsledek této úlohy.
5. Metoda Zpracovani převezme výstupní objekt metody AsyncString, spustí kód následující za awaitem. Navrací objekt typu Task.
6. Metoda Button1_Click převezme objekt Task a spustí kód následující za klíčovým slovem await.
7. Metoda Button1_Click ukončí svůj běh.

3.2 Windows Presentation Foundation (WPF)

Technologie WPF je rozhraní pro návrh a zobrazování uživatelského prostředí. Je logickým nástupce starší technologie Windows Forms nebo také nazývané WinForms, avšak WinForms je stále možné používat. Jedná se tedy o knihovnu tříd pro tvorbu grafického rozhraní a nástupce za předchozí Windows Forms. Původně vydán jako součást .NET frameworku verze 3.0. WPF využívá DirectX pro vykreslování složitějších grafických úkonů. To využívá zdrojů GPU, tedy sníží zatížení CPU a umožní mu provádět jinou práci, čímž se zrychluje výkon aplikace. [1]

Za vznikem WPF stojí především potřeba psát stále graficky složitější aplikace a potřeba unifikovat způsob návrh uživatelského rozhraní napříč všemi platformami. Dalším důvodem bylo to, že starší WinForms začala již narážet na různá omezení, které vycházeli z podstaty jejího návrhu. WinForms totiž pracuje s elementy uživatelského rozhraní vestavěného přímo do systému Windows, s kterými však není vůbec snadné pracovat a upravovat je po vzhledové stránce nebo v chování. Především v případě kdy chceme vytvořit nějaké vlastní složitější data gridy narazíme na problém, neboť tato technologie s takovými úpravy vůbec nepočítá.

WPF se oproti tomu snaží poskytnout co nejvíce rozšiřitelný objektový model. Programátor tedy není vázán na pevnou sadu komponent s fixním vzhledem, ale může si plně graficky upravovat jakoukoliv část jakékoliv komponenty nebo si ji vytvořit novou od základu. Zároveň však WPF poskytuje sadu základních komponent s základním vzhledem operačního systému. To znamená, že se s ním dá pracovat stejně jako s WinForms. [25]

Výhody technologie WPF oproti starším WinForms [10]:

- Komplexní objektový model, v kterém existuje řada nových událostí, parametrů a vlastností. Ty jde snadněji upravovat, přidávat a reagovat na chování elementů.
- Bindingy a triggery, bindingy můžeme navázat vlastnosti datových objektů na uživatelské prostředí nebo zajišťovat jednodušší logiku uživatelského prostředí bez nutnosti psát kód na pozadí.
- Styly a templaty, celé WPF je koncipované k tomu, aby se mohla měnit funkčnost i vzhled existujících prvků. Pomocí stylů lze toho velmi efektivně dosáhnout.
- Dispatchers, je to typ objektu, který zajišťuje synchronizaci úkolů do hlavního aplikačního vlákna.

- Jazyk XAML, v kterém se provádí samotný návrh uživatelského prostředí, může být využit i pro mobilní telefony Windows Phone nebo pro webové aplikace Silverlight. Vše co je definované v jazyce XAML lze také zapsat pomocí jiného jazyka, například C# nebo Visual Basic.
- Layout, správné rozložení ovládacích prvků při změně celého formuláře nebo při změnách velikosti vnitřních elementů komponent uživatelského prostředí lze vyřešit pomocí celou sadou komponent.
- Vektorové transformace, celé WPF je vektorové a je proto možné elementy zmenšovat, zvětšovat, otáčet či jinak upravovat. Výsledná aplikace je také díky tomu nezávislá na rozlišení zobrazovacího rozlišení, ať už se jedná o monitor nebo telefonní displej.
- Využívá DirectX, tedy v případě potřeby může provádět složitější grafické úkony na GPU a ulevit tím zátěž CPU. To umožní CPU dělat jinou práci a zvyšuje tím výkon celé aplikace.

3.2.1 XAML

Extensible Application Markup Language je značkovací jazyk založen na principech XML (Extensible Markup Language), vyvíjen společností Microsoft k inicializování a konstruování .NET objektů. Je ve značně využíván v .NET Frameworku verze 3.0 a 4.0, převážně ve Windows Presentation Foundation, Silverlight, Windows Workflow Foundation (WF), Windows Runtime XAML Frameworku a Windows Store aplikacích. XAML utváří ve WPF základní značkovací jazyk k utváření grafického prostředí a definuje elementy uživatelského prostředí, data binding, události a obsluhuje další služby. Ve WF lze definovat workflow pomocí XAML a také může být využit v aplikacích Silverlightu, Windows 10 Mobile nebo Univerzálních Windows Platform aplikacích.

XAML elementy se mapují přímo na Common Language Runtime instance objektů, zatímco XAML atributy se mapují na Common Language Runtime vlastnosti (properties). Soubory XAML lze upravovat nebo vytvářet pomocí design editorů jako Microsoft Expression Blend nebo Microsoft Visual Studio. Avšak dají se vytvořit i editovat pouhým textovým editorem, více specializovaným XAMLPadem nebo grafickým editorem Vector Architect.

Jakýkoliv kód XAML lze vyjádřit pomocí běžných programovacích jazyků jako C# nebo Visual Basic, avšak klíčový aspekt XAML technologie je snížení komplexity nástrojů pro zpracování XAML kódu. Díky tomu, že vychází ze značkovacího jazyka XML, vývojáři a designéři jsou schopný si sdílet a upravovat navzájem kódy bez nutnosti kompilace. XAML navíc preferuje využití deklarativních definicí uživatelského rozhraní oproti procedurálnímu k jeho vygenerování.

XAML kód může být zkompilován do .BAML souboru (Binary Application Markup Language), který poté může být vložen jako zdroj do libovolné sestavy .NET Frameworku. Během jejíchž spuštění engine frameworku extrahuje BAML soubor ze zdrojů, zparsuje jej a vytvoří odpovídající WPF visual tree nebo WF workflow. V případě užití WPF je XAML používán k popisu uživatelského rozhraní, který umožňuje definování jak 2D tak i 3D objektů, rotace, animace a další efekty. [23]

3.3 Relační databáze

Databází můžeme rozumět úložné místo pro potřebná data, podobná například knihovně, kde jsou knihy uložené podle určitého systému. Databázové systémy poté slouží k definici dat a jejich pravidel, podle kterých mají být ukládána. Dále také řeší vztahy mezi uloženými daty, jak k nim může být přistoupeno a také jaké operace s nimi můžeme provádět. V neposlední řadě také slouží jako systém pro zprávu přístupu k těmto datům, tedy spravuje práva uživatelů a jejich oprávnění k manipulaci s daty. Aplikace velmi často využívají relačních databází díky jejich jednoduchosti a pochopitelnosti. Základ relačních databází jsou tabulky, nazývané také entity, které poté obsahují sloupce jinak nazývané atributy. Mezi entitami může existovat logická vazba, které říkáme relace. Řádky tabulek poté vyjadřují jednotlivé záznamy v tabulkách a měl by mít vlastní primární klíč. Každá vlastnost entity má přiřazený název a datový typ, který definuje, co se v něm bude ukládat za data. [32]

3.3.1 SQL

Jazyk SQL (Structured Query Language) je standardizovaný dotazovací jazyk pro správu, organizaci a manipulaci s databází a je integrovanou součástí systému řízení dat. Jazyk SQL můžeme rozdělit na tři jeho hlavní části, které se zabývají rozdílnými úkony uvnitř databáze. Jazyk DDL (Data Definition Language) tvoří schémata databáze a katalogy. Jazyk SDL (Storage Definition Language) potom tvoří strukturu tabulek, plní je daty, maže potřebné záznamy a podobné. Jazyk VDL (View Definition Language) umožňuje vytvářet náhledy tabulek podle specifikovaných definic. Tyto náhledy jsou tvořeny za účelem umožnit uživatelům vyhledání pouze těch dat, které jsou mu potřeba. Náhledy jsou dynamické a svůj obsah mění podle úpravy dat původní tabulky v databázi. Aktuální náhledy mohou být také využity k přidávání a mazání dat.

Dalším jazykem je DML (Data Manipulation Language), který základní příkazy. SQL také umožňuje nastavovat rozdílná přístupová práva a sdílené využívání mezi více klienty.

SQL je však pouze dotazovací jazyk, to znamená, že nedokáže přímo deklarovat proměnné a ani nemá žádné prvky a funkce typické pro programovací jazyk. Můžeme říci, že se jedná o deklarativní jazyk, a proto jeho kód nepíšeme v samostatném programu, ale vkládáme jej do jiného programovacího jazyka, který je procedurální. Jazyk SQL využíváme v případě, že se připojujeme terminálem přímo k SQL serveru.

Mezi nejvyužívanější systémy relačních databází dnes patří například: Microsoft SQL Server, Access, Sybase, Oracle, PostgreSQL, atd. Aplikace, kterou se budu zabývat ve vlastní práci, bude pracovat s firemní databází běžící na Oracle. Všechny tyto databáze využívají jazyka SQL, avšak většina jich má vlastní nadstavby a rozdílné syntaxe, které jsou unikátní jejich systému. [32]

3.3.1.1 Základní SQL dotazy

Tuto podkapitolu chci věnovat příkladům SQL příkazů, které ve vlastní práci mohou být využity. Jelikož nevytvářím rozsáhlou databázi, ale pouze vytvářím tabulky a záznamy, uvedu zde pouze základní dotazy. [33]

- **Select** – slouží k výběru řádků a sloupců pro výpis. Jeden z nejdůležitějších a nejzákladnější příkazů SQL.
  Příklad příkazu:

  ```
  SELECT Zkratka_ulohy, Prefix_ulohy FROM Uloha;
  ```

  Tento příkaz vypíše sloupce Zkratka_ulohy a Prefix_ulohy pro všechny záznamy tabulky uloha.

- **Insert** – slouží k vložení nových záznamů (tedy řádků) do tabulky.
  Příklad příkazu:

  ```
  INSERT INTO Uloha (Zkratka_ulohy, Prefix_Ulohy)
  VALUES ('FSD', 'FS');
  ```

  Tento příkaz vloží do tabulky Uloha záznam o 2 sloupcích. Primární klíč je psán automaticky.

- **Update** – slouží k úpravě existujícího záznamu v tabulce. K identifikaci přesného záznamu určeného k modifikaci slouží klíčové slovo *WHERE*.
  Příklad příkazu:

  ```
  UPDATE Uloha
  SET Zkratka_ulohy = 'ISD', Prefix_ulohy = 'IS'
  WHERE UlohaID = 1;
  ```

  Tento příkaz aktualizuje záznam s ID 1, konkrétně přepíše jeho hodnoty ve sloupcích Zkratka_ulohy a Prefix_ulohy.

- **Delete** – slouží k vymazání konkrétního existujícího záznamu z tabulky. Podobně jako tomu bylo u příkazu *INSERT*, i příkaz *DELETE* ke specifikaci požadovaného záznamu využívá klíčové slovo *WHERE*.

  Příklad příkazu:

  ```
  DELETE FROM Uloha
  WHERE Zkratka_ulohy = 'ISD';
  ```

  Tento příkaz vymaže z tabulky Uloha záznamy, který ve sloupci Zkratka_ulohy obsahuje ‚ISD‘.

3.3.1.2 Nadstavba PL/SQL

Procedural Language/Structured Query Language je nadstavba jazyka SQL firmy Oracle. Je založení na programovacím jazyku Ada. Podobné Používá se k rozšíření SQL jazyka o konstrukce procedurálního programování jako například podmíněné příkazy nebo cykly. Dále také umožňuje deklarování proměnných, procedur a funkcí. Listy jsou také podporované za využití PL/SQL kolekcí.

Od Oracle databáze verze 8 obsahují také prvky objektového programování, jako například uložení procedur či funkcí do samotné databáze, které mohou být následně volány aplikací využívající tuto databázi. Díky tomu máme strukturální jazyk, který dokáže mnohem více věcí než samotný SQL.

V mém případě se bude jednat o uložené procedury, které budou usnadňovat chod aplikace a zefektivní psaní zdrojového kódu aplikace. Ty velice připomínají funkce v tom, že se jedná o kusy kódu, které se dají opakovaně volat. Hlavním rozdílem mezi PL/SQL procedurou a funkcí je ten, že funkce se dají zavolat uvnitř SQL dotazu, procedury však nejdou. Dalším rozdílem je i to, že procedury mohou vracet několik hodnot, funkce však pouze jednu. [11]

Procedura obsahuje vstupní (IN) a výstupní (OUT) parametry. Základní syntaxe PL/SQL procedury vypadá takto:

```
CREATE [OR REPLACE ] PROCEDURE procedure_name (parameter_list)
IS
    [declaration statements]
BEGIN
    [execution statements]
```

```
        EXCEPTION
            [exception handler]
    END [procedure_name ];
```

Procedura začíná hlavičkou specifikující její název a seznam parametrů. Každý z těchto parametrů je buď v *IN*, *OUT* nebo *INOUT*. To specifikuje, zda lze tyto parametry číst nebo do nich zapisovat.

- **IN** – *IN* parametr je pouze pro čtení. Slouží jako vstupní parametr, na který můžeme uvnitř procedury odkazovat ale nemůžeme měnit jeho hodnotu. Oracle využívá parametr *IN* jako defaultní, pokud se tedy nespecifikuje, parametr bude *IN*.
- **OUT** – *OUT* parametru lze upravovat jeho hodnotu. Může sloužit jako návratová hodnota pro program, který proceduru zavolal.
- **INOUT** – *INOUT* parametr je vstupní parametr, kterému může procedura měnit hodnotu.

OR REPLACE pak umožní přepsat proceduru.

Tělo procedury se poté skládá ze tří částí. Spustitelná část (Executable) je povinná, zatímco části deklarativní (Declarative) a výjimkové (Exception-handling) části jsou nepovinné.

- **Declaration** – Obsahuje deklarace proměnných, konstant, ukazatelů atd.
- **Execution** – Tato část obsahuje business logiku procedury.
- **Exception** – Tato část řeší vyjímky.

Taková procedura jde poté zkompilovat a uložit v Oracle databázi. Následně ji lze možné zavolat příkazem EXECUTE název_procedury (parametry); [11]

4 Vlastní práce

Cílem vlastní práce bylo vytvořit aplikaci v prostředí .NET, založené na technologii Windows Presentation Foundation a s využitím Entity Frameworku. Tato část práce se zaměřuje na praktickou demonstraci užitých postupů a na ukázky užití zdrojového kódu.

V úvodu jsem definoval účel užití aplikace. Aplikace je vyvíjena pro Český statistický úřad za úkolem zpracování, čištění a zálohování audittrail dat tazatelských souborů z domácnostních šetření. Aplikace vzniká jako náhrada za stávající řešení, které by měla nahradit za účelem zrychlení, zautomatizování a zefektivnění operace.

4.1 Aktuální stav

Český statistický úřad pro zpracování audittrail dat a seznamu kontaktů a pagin z domácnostních šetření využíval do současnosti vnitřně vytvořenou aplikaci psanou v jazyce Visual Basic.

Audittrail daty se rozumí textový soubor, který představuje chronologický záznam toho, co tazatel na formuláři dělal a se specifickým formátováním. Tedy zaznamenává například to, kdy byl formulář dotazníku otevřen nebo kdy bylo vybráno určité políčko otázky dotazníku, z jaké a na jakou hodnotu bylo políčko změněno a kdy tazatel z políčka pokračovat na další.

Tyto data jsou brané z tazatelských souborů, které jsou odesílané a ukládané na počítači uživatele. Takový soubor může být z různých zdrojů a týká se vždy jedné úlohy v daném roce a období. Nejprve může být od jednotlivých tazatelů, kteří pracují v terénu. V takovém případě bude soubor obsahovat audittrail data za dosud vyplněné dotazníky tohoto jednoho tazatele, jeho seznam kontaktů a pagin bytů/domů kontaktů. Takový zdroj má příponu slo. Dalšími zdroji může být Garant kraje, kde v takovém souboru bude více audittrail souborů za jednotlivé tazatele a celkový seznam jejich kontaktů a pagin bytů/domů. Tento soubor má příponu slog. Posledním zdrojem je poté Garant ČR, který sjednocuje soubory jednotlivých Garantů za kraje a má příponu slogcr.

4.2 Zadání

Aplikace s názvem CO_Zpracování, která je předmětem této diplomové práce, by měla umožňovat efektivnější a automatické zpracování, kontrolu a aktualizace dat tazatelských souborů a jako taková kompletně nahradí stávající řešení. Uživatelem aplikace bude programátor či administrátor z oddělení koordinace domácnostního šetření a bude hlavně využívána na pozadí. Aplikace tedy nemusí být zcela uživatelsky přívětivá, priorita je její funkčnost.

Základní činnost aplikace je tedy vytváření tabulek pro kontakty, paginy jejich bytů/domů a jejich aktualizace v případě že již existují. Další základní činností je konvertovat audittrail data tazatelských souborů do tabulek Oracle databáze. Během této konverze aplikace nalezne chyby v audittrail souborech, tyto záznamy vynechá z původní tabulky a vytvoří pro ně novou tabulku s chybami. To může být zapříčiněno například chybou v separaci řádků, kdy nebyl rozpoznán separující znak, nebo neodpovídající posloupností parametrů, kdy například byla ukončena sekvence, aniž by předtím začala.

Zpracování souboru může být provedeno pomocí dialogového okna volbou požadovaného tazatelského souboru se správnými přípony. Toto zpracování však jde také zautomatizovat, umožní to tak uživateli pracovat na jiné práci, zatímco se soubory budou zpracovávat na pozadí. Automatické zpracování bude procházet složku uvedenou v aplikaci. Mimo složky obsahující soubory lze také nastavit složky pro uchování nezpracovaných souborů, již zpracovaných souborů a dočasnou složku pro práci se soubory. Složky nemohou být pevné kvůli potenciální změně souborového systému.

Procházení bude prováděno prioritně podle zdroje souboru, tedy Garant ČR má přednost před Garantem kraje a ten má přednost před tazatelem. Protože každý soubor může obsahovat více audittrail souborů, uživatel si také může nastavit kolik těchto souborů bude chtít paralelně zpracovávat.

Uživatel může zpracování přerušit v průběhu jeho zpracovávání. V případě že se zpracovává soubor z automatického zpracovávání, může aplikaci říct, aby nejprve dodělalo momentálně zpracovávaný soubor a poté přerušilo vyhledávání dalších souborů.

Veškeré kroky zpracování aplikace uchovává včetně data a času v logu, z kterého uživatel může vysportovat obsah do textového nebo RTF souboru. Po dokončení zpracování

souboru a vytvoření záznamů v Oracle databázi je soubor seřazen podle úlohy a období mezi zpracované soubory. V případě přerušení nebo nastalé chyby v průběhu zpracování bude soubor nahrán do složky pro nezpracované soubory spolu s textovým souborem s příponou slot, která bude obsahovat chybovou hlášku.

4.2.1 Shrnutí požadovaných funkcionalit aplikace

V této kapitole shrnuji veškeré funkcionální požadavky, které jsou od aplikace očekávány:

- Aplikace umožňuje v první řadě ruční zpracování jednoho zadaného tazatelského souboru, které si uživatel vybere přes dialog okno.
- Aplikace dále bude umět automatické zpracování, které bude moci jet na pozadí počítače uživatele. Zde je požadavkem, aby aplikace vyhledávala ve složce soubory prioritně dle zdroje tazatelského souboru. Jestli-že to žádný soubor nenajde, bude složku každých 5 minut kontrolovat pro nové soubory do zastavení automatického zpracování.
- Uživatel může nastavit výchozí složky pro zpracování.
- Uživatel má možnost zpracování souboru zastavit během jeho zpracovávání. V případě automatického zpracování také může nechat dopracovat momentální zpracování souboru.
- Uživatel má možnost nastavit, kolik audittrail souborů bude zpracováváno paralelně najednou.
- Aplikace bude veškeré své činnosti uchovávat v logu, včetně data a času provedení, který uživatel může vyexportovat jako textový nebo RTF soubor.
- Aplikace bude řadit zpracované a nezpracované soubory, k nezpracovaným přidá chybovou hlášku v textovém dokumentu pro informaci.

4.3 Analýza

V této podkapitole rozeberu analýzu požadavků a funkcionalit na aplikaci, zpracovávaný soubor, spojení s databází a návrh tříd business logiky aplikace.

4.3.1 Tazatelský soubor

Tazatelské soubory se vztahují vždy na specifické vyšetřované úlohy (například Výběrové integrované šetření v domácnostech IŠD) a na konkrétní vyšetřované období (například rok 2020 kvartál 1). Tyto soubory poté obsahují buďto soubory jednoho tazatele, či více v případě souborů garanta nebo garanta ČR. Dále také obsahují záznam chyb, které se tazatelům vyskytly během používání formuláře. V neposlední řadě obsahují soubory samotného formuláře, který tazatel využívá pro svou práci. Soubory, s kterými moje aplikace bude pracovat jsou soubory audittrailu, které obsahují přesný log aktivity na formuláři, a soubory MDB obsahující tazatelovy přidělené kontakty k vyšetření.

Zde uvedu jak může vypadat takový audittrail soubor, informace jsou náhodně vygenerována a data nemusí nutně odpovídat skutečnému času provádění dotazníku:

"2.2.2020 14:28:14","Start Session"
"2.2.2020 14:28:16","Enter Form:89","Key:669892311"
"2.2.2020 14:28:16","Enter Field:Byt.idRefObd","Status:Normal","Value:1"
"2.2.2020 14:28:17","Leave Field:Byt.idRefObd","Cause:Next Field","Status:Normal","Value:1"
"2.2.2020 14:28:17","Enter Field:Byt.aaPocOsB","Status:Normal","Value:"
"2.2.2020 14:28:43","Action:Store Field Data","Field:Byt.aaPocOsB"
"2.2.2020 14:28:43","Leave Field:Byt.aaPocOsB","Cause:Next Field","Status:Normal","Value:1"
"2.2.2020 14:33:16","Leave Form:89","Key:669892311"
"2.2.2020 14:33:16","End Session"

Tyto soubory obsahují datum a čas začátku každé rekordované akce, o jaký typ akce se jedná, parametry akce a vložené hodnoty.

4.3.2 Databáze

Mým původním záměrem při specifikaci zadání této diplomové práce bylo využít Entity Frameworku s kterým jsem mírně pracoval ve své bakalářské práci, ale vzhledem k minimálním požadavkům struktury databáze mě toto řešení přišlo zbytečně komplexní. Veškeré mnou ukládaná data jsou vkládána do centrální databáze bez jakékoliv vazby na jiné data uložené v databázi a použití SQL příkazů k docílení takového úkonu mě přišlo zcela postačující. Ke spojení s centrální databází a spouštění příkazů jsem se proto rozhodl namísto Entity Frameworku implementovat pouze použití knihoven OracleCommand a OracleDataAdapter. Tyto knihovny mě dále také umožní bez problému spouštět uložené procedury v centrální databázi a budou tak redukovat potřebu specifikovat SQL kód uvnitř zdrojového kódu aplikace.

4.3.2.1 Normalizace Dat

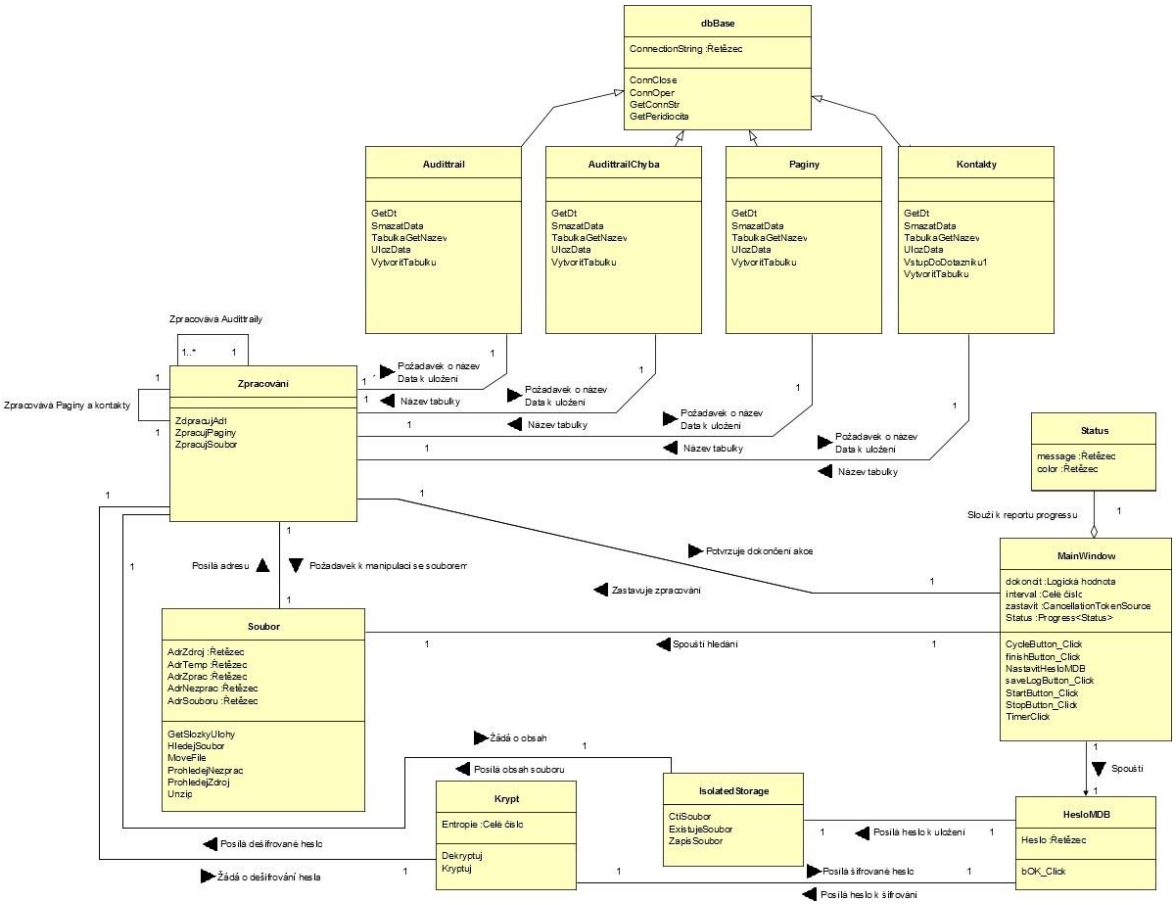
Moje práce neobsahuje žádnou rozlehlou databázi a vytvářím v Oracle pouze tabulky pro zpracované data, postačí mě tedy pouze první dva stupně normalizace databáze:

1. **Normální forma** – Nám dává pouze pravidlo, že každý sloupec v tabulce bude představovat pouze nějaký jedinečný typ informace. Tedy že všechny atributy tabulky by měli být atomické, nedělitelné a obsahovat pouze jednu hodnotu. Je prvotním a velice důležitým pravidlem, které nám usnadňuje a zefektivňuje vyhledávání záznamů.
2. **Normální forma** – Poté udává to, že každý atribut musí být závislí na primárním klíči. Tedy v tabulce by neměl být atribut, který nijak nesouvisí s primárním klíčem tabulky.

4.4 Návrh

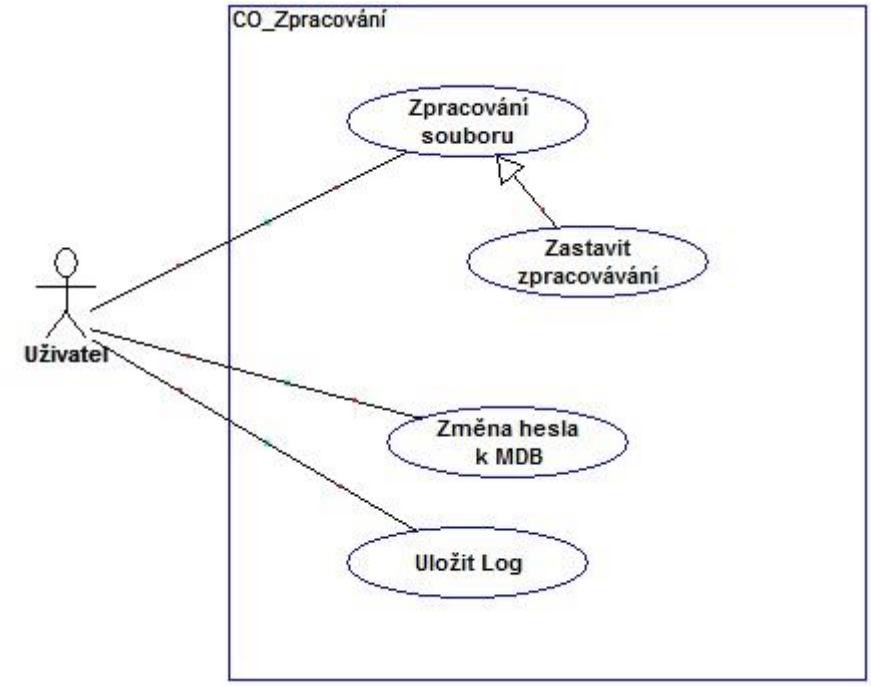
V první řadě před samotnou implementací je důležité si rozvrhnout a navrhnout jednotlivé funkční požadavky na aplikaci. Takového návrhu dosahuji nejdříve diagramem tříd, v kterém si rozvrhnu třídy aplikace, jejich vlastnosti, metody a jejich architekturu. Dále je také důležité si být vědom, kdo bude aplikaci využívat. K tomu slouží Use Case diagram. Nakonec jsem vytvořil diagram aktivity pro hlavní funkci aplikace, tedy automatické zpracování, který má utvářet představu o ovládání, jak se bude aplikace chovat během chodu a jaké inputy od uživatele budou třeba.

4.4.1 Model tříd aplikace CO_Zpracování



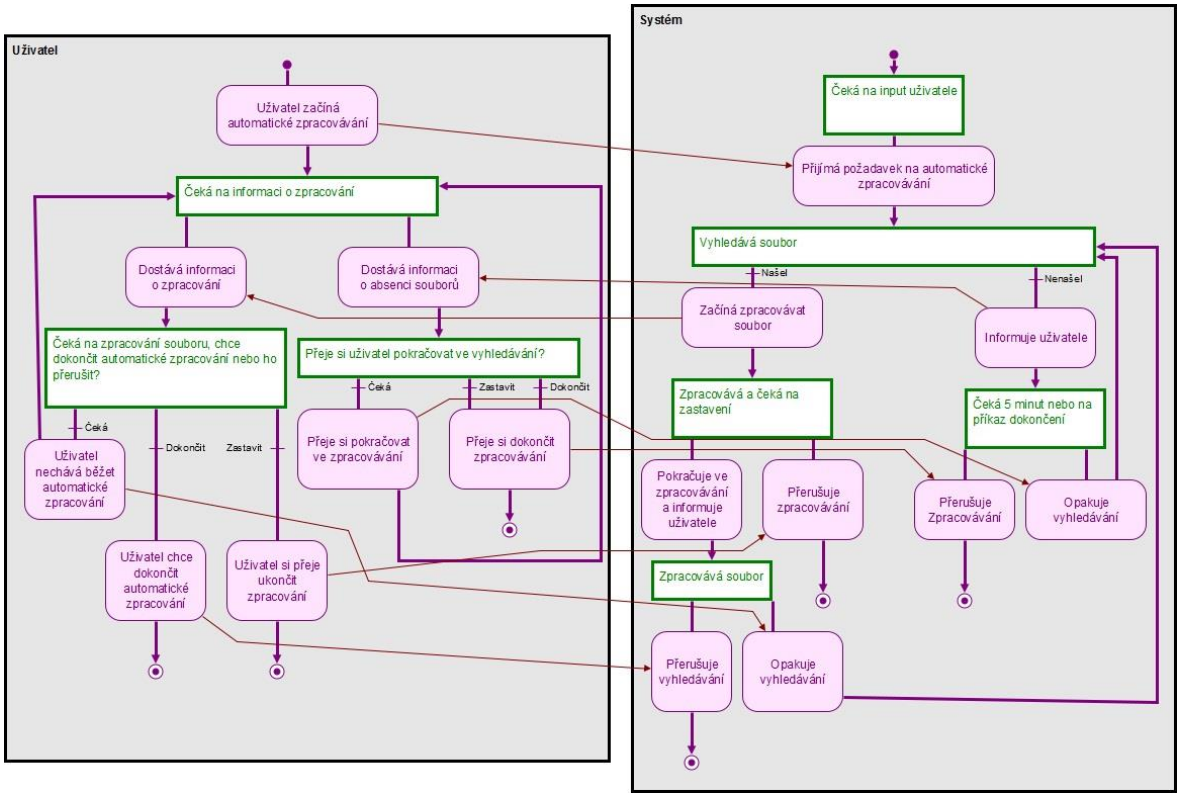
Obrázek 3 – Diagram tříd

4.4.2 Use Case diagram



Obrázek 4 – Use Case diagram

4.4.3 Model aktivit



Obrázek 5 – Activity diagram

4.5 Implementace aplikace

V této kapitole popisuji implementaci vlastní práce. Kapitolu rozděluji na 3 hlavní podkapitoly, které se zabývají modelovou částí aplikace, uživatelským rozhraním a nakonec business logikou aplikace.

4.5.1 Databáze

Aplikace CO_Zpracování využívá k úchově dat firemní centrální databázi Oracle. Pro navázání spojení s databází využívám Oracle Managed Data Access, což je oficiální Oracle Data Provider pro .NET. Nevýhodou je nutnost instalace data provideru na počítač uživatele. Jedná se však o podnikovou aplikaci a koncovým uživatelem bude programátor, který Oracle data provider má již nainstalován pro svou práci, nevýhoda je zde proto anulována.

Kromě centrální databáze ještě pracuji s instancemi Microsoft SQL databáze, které jsou přítomné v tazatelských souborech. S těmi pracuji pomocí OLE DB (Object Linking and Embedding, Database), oficiálním API vyvíjeným Microsoftem.

4.5.1.1 Connection String

Connection Stringy nebo-li připojovací řetězce jsou objekty, které dědí z třídy DbConnection a také vlastnosti ConnectionString. Ten se může měnit podle konkrétního zprostředkovatele. V případě připojování na centrální Oracle databázi využívám standardní Oracle connection stringu:

```
this._ConnStr = string.Format("data Source=apolo.CSU;User Id={0};
Password={1};", Id, Psw);
this.conn = new OracleConnection(_ConnStr);
```

V případě připojení se k instancím Microsoft SQL databáze využívám OLE DB Connection string:

```
string connStr = String.Format("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=\"{0}\";Jet OLEDB:Database Password={1};", dbKontaktCesta,
Krypt.Dekryptuj(IsolatedStorage.CtiSoubor("MDB")));
```

Třída Krypt obsahuje metody pro krytování a dekretování hesel, třída IsolatedStorage poté obsahuje metody pro zápis a čtení souborů z izolovaného úložiště. Více o těchto třídách napíšu v pozdější části vlastní práce.

4.5.1.2 OracleCommand

OracleCommand objekt reprezentuje SQL příkaz, který se má provést na databázi, buď se jedná o SQL řetězec nebo o uloženou proceduru na straně databáze. Pomocí různých přetížení lze použít několik různých variant syntaxe. Objekt také implicitně obsahuje metody určené k provádění příkazového řetězce a nastavení vstupních parametrů.

Dvě mnou užívané metody, které zde uvedu, jsou ExecuteScalar a ExecuteNonQuery. ExecuteScalar vrací první sloupec prvního řádku v sadě výsledků dotazu, který je vrácen jako datový typ .NET Frameworku.

Na příkladu ExecuteScalar metody OracleCommandu ze zdrojového kódu metoda vrací nalezený prefix podle zadané úlohy z databáze:

```
string sql = "Select prefix_ulohy from setdom.uloha
where zkratka_ulohy=:uloha";
OracleCommand oc = new OracleCommand(sql, conn);
oc.Parameters.Add(new OracleParameter("uloha", uloha));
string prefix = oc.ExecuteScalar().ToString();
```

ExecuteNonQuery provede požadovaný příkaz a vrací počet ovlivněných řádků. Ve svém kódu tuto variantu využívám ke spouštění uložených PL/SQL procedur, příkladem může být metoda vytvoření nové tabulky kontaktů:

```
public void VytvoritTabulku(string TabNazev)
{
    using (OracleCommand cmd = new OracleCommand())
    {
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "SETDOM.Kontakty_Vytvorit";
        cmd.Parameters.Add(GetParametr_NazevTab(TabNazev));
        cmd.Connection = this.conn;
        this.ConnOpen();
        cmd.ExecuteNonQuery();
        if (!ConnNechatOtevrene)
            this.ConnClose();
    }
}
```

Zde však ve svém kódu nevyužívám návratovou hodnotu metody ExecuteNonQuery. Ta by se případně dala využít například pro kontrolu, zda byl záznam přidán do tabulky. Příkladem toho a alternativní syntaxe inicializace OracleCommandu by mohlo být (nepoužito v mém zdrojovém kódu):

```
string sql = "INSERT INTO SETDOM.ULOHA (zkratka_ulohy, prefix_ulohy)
VALUES ('ISD', 'IS')“;
using(OracleCommand cmd = new OracleCommand(SqlPrikaz, conn))
{
        this.ConnOpen();
        int updated = cmd.ExecuteNonQuery();
        if (updated == 0)
                Vlozeno = "Úlohu se nepodařilo přidat.";
        else
                Vlozeno = "Úloha byla úspěšně přidána.";
        if (!ConnNechatOtevrene)
           this.ConnClose();
}
```

4.5.1.3 DataSet

DataSet představuje mezipaměť dat, získaných ze zdrojových dat databáze, ale bez ohledu na zdroj dat. Je hlavní součást architektury ADO.NET. DataSet se skládá z kolekce objektů DataTable, které budu využívat pro svou práci. Tyto objekty jde případně vzájemně propojit pomocí objektů DataRelation. Integritu dat uvnitř DataSetu můžeme vymáhat pomocí objektů UniqueConstraint a ForeignKeyConstraint.

DataTable pak obsahuje samotná data a pomocí DataRelationCollection můžeme navigovat hierarchií tabulky. Tabulky jsou umístěny v DataTableCollection, který slouží k přístupu k jednotlivým tabulkám. DataTable tabulky jsou podmíněně citlivá na velká či malá písmena, tedy jsou citlivá v případě, že kolekce obsahuje stejné dva či více názvů tabulek jen s jinými kapitalizacemi písmen.

DataSet umí číst a ukládat data a schéma jako XML dokumenty. Ty je poté možno přenášet přes protokol http a použít libovolnou aplikací na libovolné platformě s podporou XML. K tomu slouží metoda WriteXml pro zápis a ReadXml pro čtení dokumentu či schéma v XML.

Ve své práci využívám právě objektů DataTable, do kterých načítám data z centrální databáze pomocí DataAdapteru. DataTable se dále skládá z objektů DataRow, která obsahuje data za jednotlivé řádky tabulky. S těmi se pracuje v případě změň v objektech DataTable pomocí přidání, aktualizace nebo odstraněním objektů DataRow.

DataSety dále mají metody pro povolení či zamítnutí změň DataSetu nebo Merge (spojení) dvou DataSetů.

4.5.1.4 OracleDataAdapter

OracleDataAdapter slouží jako můstek mezi objekty DataSet ze třídy System.Data a databází. K tomu slouží metoda System.Data.Common.DbDataAdapter.Fill, která naplní data ze zdrojové databáze přímo do objektu DataSet. Když OracleDataAdapter naplní DataSet, vytvoří veškeré potřebné tabulky a sloupce, pokud ještě neexistují. Veškerá číselná pole jsou do DataSetu z metody Fill importována namapovaná na OracleNumber objekty. Informace o primárním klíči však nejsou zahrnuté do implicitně vytvořeného schématu, pokud není nastavená vlastnost MissingSchemaAction na AddWithKey. Také je možné vytvořit schéma DataSetu, včetně informace o primárním klíči, ještě před jeho naplněním. Dále také může volat metodu System.Data.DataSet.Update() pro zpětné posílání změň do zdrojové databáze. Je to takové spojení OracleConnection a OracleCommand k zvýšení rychlosti a efektivity připojení k Oracle databázím. Ve svém kódu jej využívám k naplňování DataTable objektů datami z centrální Oracle databáze.

OracleAdapter dále disponuje metodami SelectCommand, InsertCommand, DeleteCommand, UpdateCommand a vlastnostmi TableMappings, kterými může uskutečnit načítání a změnu dat.

Příkladem užití v mé práci může být například naplnění DataSetu seznamem Pagin z Microsoft SQL Server instance, který se nachází v tazatelských souborech:

```
// Poznámka: Right(Poznamka,2000) -
pokud je více znaků, pak nelze zapstat do Oraclu
string sql = string.Format("SELECT VAL(IIF([Pagina]<9999,[Okres] & '0'
& [Pagina], [Okres] & [Pagina])) AS PK, " + " CisBytu, CisObec, ID_Vysl
Setr, ID_VyslSetr_Pred, " + "Modul, NazObec, ObdZar, OsobniCislo, PocCl
enDom, PocDom, PosNav, " + "Right(Poznamka,2000) As Poznamka, RefRok,
RefObd, SO, TazSoubor, Vlna, VyslHD1, VyslHD2 " + "FROM Pagina " +
"WHERE Uloha='{0}';", Uloha);

DataTable dtPagina = Pagina.GetDt();
using (OleDbDataAdapter da = new OleDbDataAdapter(sql, connStr))
{
    da.Fill(dtPagina);
}
```

Metoda GetDt slouží k vytvoření požadované struktury sloupečků tabulky pro daný DataTable.

4.5.1.5 PL/SQL Procedury

Jak jsem již uvedl v teoretické části, PL/SQL (Procedural Language/Structured Query Language) je nadstavba Oracle databáze založená na programovacím jazyku Ada a umožňuje mimo jiné ukládat na centrální databázi procedury, které lze opakovaně volat. Tyto procedury zefektivňuje a usnadňuje psaní kódu. Procedury volám přes objekt OracleCommand.

Procedury používám na vytvoření předem specifikovaných tabulek pro Audittrail, Audittrail chyby, Paginy a Kontakty. Procedury pro mazání audittrailů a audittrail_chyba jsou 2, jedna procedura smaže veškerý obsah dané tabulky, druhá procedura maže z dané tabulky pouze specifické tazatelské soubory. Mazání Kontaktů a Pagin se dále rozděluje na 3 procedury, které mažou všechen obsah tabulek podle názvu tabulky, mažou zadané kraje v určité tabulce nebo mažou podle tazatelského souboru. Paginy a Kontakty mají dodatečně proceduru, která zapíše data do tabulky v případě že neexistují, jinak aktualizuje existující řádky tabulky.

Jako příklad uložené procedury popíši proceduru určenou k mazání Audittrail záznamů podle tazatelského souboru.

```
CREATE OR REPLACE PROCEDURE Smazat_audittrail_soubor (NazevTabulky In
Varchar, pSoubor in Varchar)
AUTHID CURRENT_USER

IS
  data_not_found_1410 exception;
  pragma exception_init(data_not_found_1410,-1410);
      table_not_exist_942 exception;
      pragma exception_init(table_not_exist_942,-942);
      sql_pocet varchar2(1000);
      pocet_radku number;

BEGIN
      sql_pocet:='select count(pk)
      from setdom.'|| NazevTabulky||'
      where soubor = '''|| pSoubor ||''''';
      execute immediate sql_pocet into pocet_radku;

      EXECUTE IMMEDIATE 'delete from setdom.'|| NazevTabulky ||'
      where soubor='''|| pSoubor ||''''';
      commit;
      dbms_output.put_line('V tabulce '|| NazevTabulky||' bylo
      smazáno '|| pocet_radku ||'.');

      exception
            when data_not_found_1410 THEN NULL;
            when table_not_exist_942 then null;
end;
```

V hlavičce procedury se definuje název procedury a jejich parametrů, porodně jako by se definovala metoda ve zdrojovém kódu C#. Klauzule AUTHID pouze udává, že práva pro spouštění kódu procedury budou stejná jako uživatel, který proceduru volá a nikoliv její původní autor. Oracle defaultně při nespecifikování volí práva autora procedury.

V prvním bloku kódu poté definujeme proměnné, do kterých můžeme ukládat data a pracovat s nimi podobně, jako například v C#. Například proměnná pocet_radku slouží ve výstupu pro oznámení, kolik řádků bylo smazáno.

Ve druhém bloku kódu běží samotná logika procedury. Zde je vidět, že můžeme deklarovat Varchar řetězec, který využijeme k uložení sql kódu. Ten můžeme sestavit pomocí vstupních parametrů a následně jej zavolat příkazem EXECUTE IMMEDIATE. Dále klauzulí INTO můžeme výsledek nahrát do proměnné pocet_radku, který bude obsahovat konkrétní počet řádků ke smazání. Dále provedeme další SQL příkaz pro samotné smazání řádků z tabulky. Nakonec zavoláme příkaz dbms_output.put_line, který bude sloužit jako výstup procedury. V tomto případě nás informuje o úspěšnosti provedení procedury a informuje nás o počtu řádků, které se smazali.

Ve třetím bloku poté řešíme vyjímky. Zde pouze v případě, že data či tabulka nebyly nalezeny, je výstup procedury nulový. Samotné výjimkové zprávy v aplikaci řeším přímo kódově na straně aplikace.

4.5.1.6 Třídy pro práci s daty a databází

K obsluze databáze a dat využívám řadu tříd s metodami, které jsou uzpůsobeně datům, které mají zpracovávat. Mimo jiné zde figuruje třída dbBase, kterou jsem vytvořil jako třídu ze které budou ostatní dědit. Ta slouží k tomu aby předávala základní metody, které budou využívat všechny třídy. Jedná například o vytváření Connection Stringu a metody pro jeho získání či otevření a zavření připojení. Dále je velmi důležitá metodu pro získání periodicity zpracovávané úlohy. Tato metoda slouží poté při vytváření názvů tabulek. Periodicita znamená, jak často v roce se šetření koná. To může být buďto čtvrt roční, půl roční nebo roční.

Příklad metody pro získání Periodicity:

```
public string GetPeriodicita(string uloha, int rok, byte kvartal)
{
    string sql = "SELECT periodicita FROM setdom.uloha_obdobi uo join
    setdom.uloha u on uo.id_uloha=u.uloha_id where u.zkratka_ulohy=
   :uloha AND LOWER(obdobi) = :obdobi";
    OracleCommand oc = new OracleCommand(sql, conn);
    oc.Parameters.Add(new OracleParameter("uloha", uloha));
    oc.Parameters.Add(new OracleParameter("obdobi",
    $"{rok}q{kvartal}"));
    try
    {
        return oc.ExecuteScalar().ToString();
    }
    catch
    {
        return null;
    }
}
```

Z této třídy dědí další 4 speciálně vytvořené pro obsluhu dat do tabulek Audittrail, Audittrail_Chyba, Kontakty a Pagina. V těchto třídách sestavuji název tabulky podle typu úlohy, roku a kvartálu zpracování a také podle původu dat, tedy od jaké role je původní zpracovávaný soubor, která mění koncovku názvu tabulky. Formát názvů tabulek dále vysvětlím v kapitole Výsledky a Diskuze, kde uvedu jak program funguje z uživatelova pohledu. Dále zde figuruje metoda GetDt, která slouží k předpřipravení nově deklarovaných objektů DataTable na požadovanou strukturu sloupců a datových typů. Této metodě dále slouží metody pro nastavení a získávání potřebných OracleCommand parametrů. Základní metody pro zpracování tabulek v databázy se mírně liší podle typu dat. Audittrail a Audittrail_Chyba obsahuje metody pouze pro vytvoření tabulky, smazání dat v tabulce podle tazatelského souboru a uložení dat do tabulky.

Kontakty a Pagina mají navíc mazání dat z tabulek navíc řešené podle názvu tabulky, zdroje tazatelského souboru nebo kraje. Třída kontakty pak má navíc ještě metodu, která vrací čas a datum vstupu do dotazníků za jednotlivé šetření.

Příklad spuštění takovéto procedury přes objekt OracleCommand poté vypadá podobně jako spouštění běžného SQL příkazu, jen se upraví parametr CommandType k užití procedury na CommandType.StoredProcedure. Text příkazu je poté název uložené procedury.

```
using (OracleCommand cmd = new OracleCommand())
{
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "SETDOM.Smazat_Audittrail_Soubor";
        cmd.Parameters.Add(this.GetParametr_NazevTab(TabNazev));
        cmd.Parameters.Add(Audittrail.GetParametrSloupec_Soubor(soubor));
        cmd.Connection = this.conn;
        this.ConnOpen();
        cmd.ExecuteNonQuery();
}
```

Dalším zajímavým využití PL/SQL nadstavby může být procedura Pagina_Upsert_p, která využívá podmínek. Z důvodu opakování a šetření místa vynechávám velkou řadu procedury, parametrů a uvedu pouze samotné podmínění:

```
v_sql:= 'merge into ' || pNazevTabulky || ' t
   using dual
   on (t.PK = :pPK)
  when matched then
      update set t.CisBytu = :pCisBytu, …

  when not matched then
      insert
            (t.PK, …)

      values
            (:pPK2, …)';
```

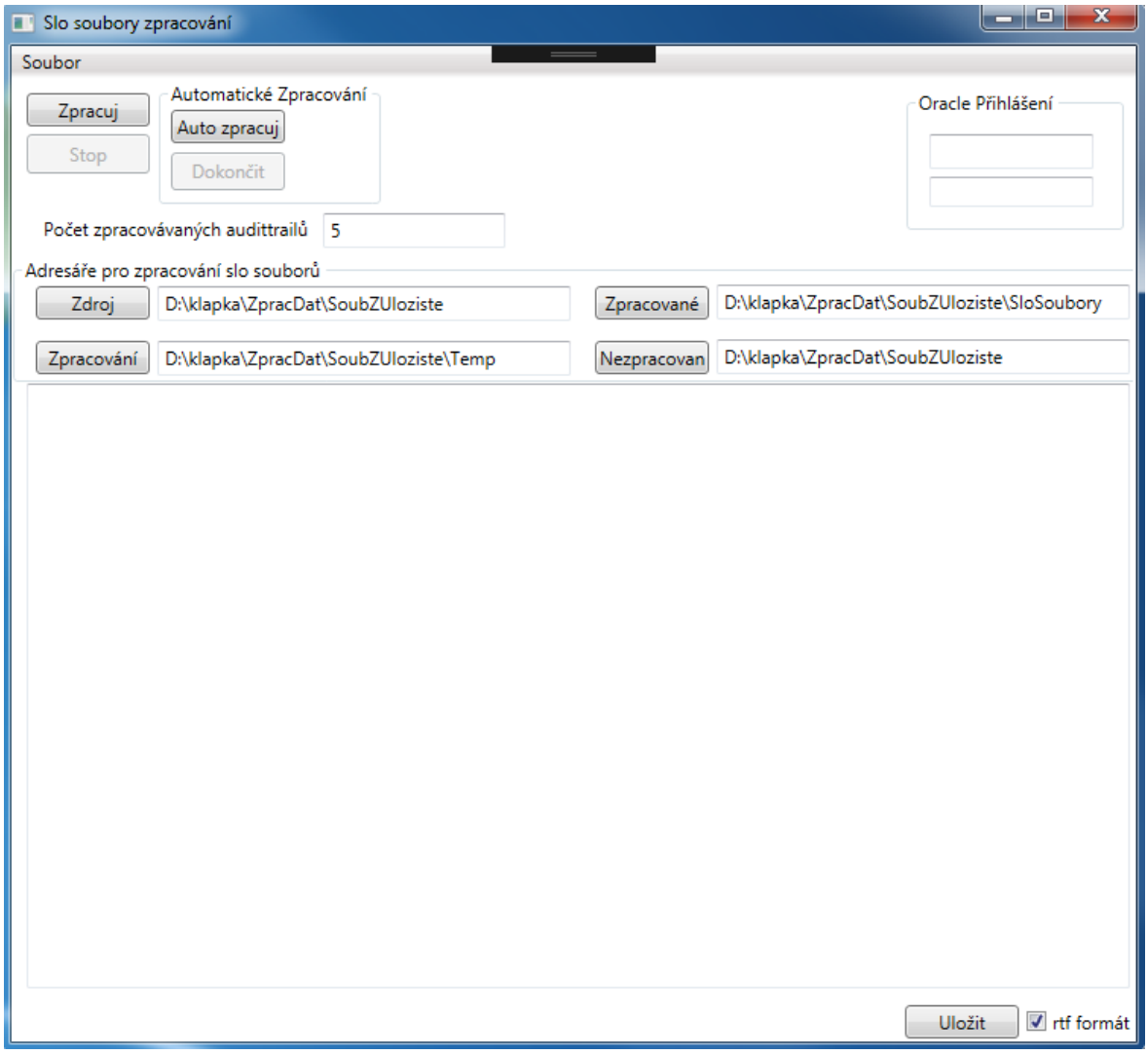
V této proceduře porovnávám primární klíče všech aktuálních pagin. V případě, že tato pagina již existuje v dané tabulce, ji pouze aktualizuji hodnoty sloupečků.
Pokud hodnota v tabulce neexistuje, beru řádek jako zcela nový záznam a použiju příkaz INSERT k nahrání nového záznamu do tabulky.

4.5.2 WPF

Grafické rozhraní WPF je vytvořeno pomocí značkovacího jazyka XAML. Za tímto rozhraním také může dále běžet kód na pozadí, tedy code-behind. Zde můžeme definovat další logiku aplikace nebo reakce na události, které jsou vytvářené formulářem. Většina takového kódu by se měla věnovat samotného grafického rozhraní a chod aplikace nechat na jiných třídách.

Pro aplikaci využívám 2 formulářové okna. Jedno hlavní menu aplikace a druhé pro nastavení a uložení hesla pro instance Microsoft SQL server uložené v tazatelských souborech.

Hlavní menu a form aplikace:

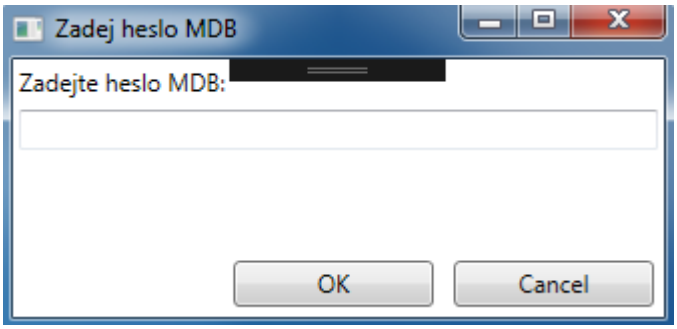


Obrázek 6 – Hlavní okno aplikace

V pravé horní straně okna jsou políčka pro vyplnění přihlašovacích údajů do Oracle databáze, které jsou povinná pro chod aplikace. V levém horním rohu tlačítko Zpracuj slouží

ke zpracování vybraného souboru a tlačítko Stop slouží k okamžitému zastavení zpracování jak jednotlivého tak i automatického zpracování. Tlačítko Auto zpracuj v containeru Automatické Zpracování pak slouží ke spuštění automatického vyhledávání a zpracování souborů podle programově vytvořených specifikací. Tlačítko Dokončit nechá zpracovat momentálně zpracovávaný soubor a poté automatické zpracování ukončí. Text box pod tlačítky ovládající zpracování slouží k předefinování počtu paralelního zpracování audittrail souborů. Tlačítka a textboxy uprostřed okna slouží k definování všech potřebných adresářových cest pro zajištění funkčnosti aplikace. Pod nimi se nachází log aplikace, který zaznamenává probíhání zpracování. Tlačítko a combobox na spodu obrazovky slouží k exportování tohoto logu do zadaného adresáře, combo box ovlivňuje, zda se bude jednat o soubor formátu rtf nebo pouhý textový soubor.

Aplikační okno pro nastavení hesla k .mdb souborům v tazatelských souborech:



Obrázek 7 – Okno pro zadání nového hesla k souboru MDB

Přesné ovládání aplikace bude uvedeno v kapitole Výsledky a diskuze.

Ukázka XAML kódu nastavení hesla k .mdb souborům, které ukazuje sestavení a formátování okna:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <TextBlock Margin="3">Zadejte heslo MDB:</TextBlock>
    <PasswordBox x:Name="pbHeslo" Grid.Row="1" Margin="3" MaxLength="29"/>
    <DockPanel Grid.Row="2" VerticalAlignment="Bottom" HorizontalAlignment="Right">
        <Button x:Name="bOK" Height="23" Width="100" Margin="5" IsDefault="True"
                Click="bOK_Click">OK</Button>
        <Button Height="23" Width="100" Margin="5" IsCancel="True">Cancel</Button>
    </DockPanel>
</Grid>
```

4.5.2.1 Kód za WPF

Kód na pozadí WPF používám primárně k ovládání grafického rozhraní a jako metody tlačítek, které dále volají specifické metody mimo grafické rozhraní. Snažil jsem se takto rozdělit grafickou stránku a rozhraní aplikace od business logiky aplikace.

V samotné inicializaci okna definuji některé vlastnosti, s kterými poté pracuji dále v mých metodách. Jedná se například o boolean dokoncit, který zaznamenává žádost o dokončení právě se zpracovávaného souboru a následně pozastavení automatického vyhledávání nebo vlastnosti WPF ovládacích prvků, které chci mít při spuštění nepovolené. Kromě toho zde také kontroluji existenci zašifrovaného souboru na izolovaném disku obsahující heslo k MDB databázím uvnitř souborů a v případě jeho neexistence nabádám uživatele k jeho vytvoření. Toto heslo jde však dále upravit v nastavení.

Mezi mými vytvořenými metodamy, které zde využívám, jsou například metody pro výpis do logu v okně. K zápisu do logu využívám RichTextBox, který mě umožní zvýraznit barevně důležité zprávy. V parametrech metody udávám objekt RichtextBox, který chci měnit. Dále také text a barvu textu. Poslední funkcí metody je psát nové zprávy vždycky na nový řádek.

```
public static void AppendLine(RichTextBox source, string text, string color)
{
    string str = new TextRange(source.Document.ContentStart,
    source.Document.ContentEnd).Text;
    if (str.Length == 0)
    {
        BrushConverter bc = new BrushConverter();
        TextRange tr = new TextRange(source.Document.ContentEnd,
        source.Document.ContentEnd);
        tr.Text = text;
        try
        {
            tr.ApplyPropertyValue(TextElement.ForegroundProperty,
            bc.ConvertFromString(color));
        }
        catch (FormatException) { }
    }


    else
    {
        BrushConverter bc = new BrushConverter();
        TextRange tr = new TextRange(source.Document.ContentEnd,
        source.Document.ContentEnd);
        tr.Text = "\r"+text;
        try
        {
            tr.ApplyPropertyValue(TextElement.ForegroundProperty,
            bc.ConvertFromString(color));
            source.ScrollToEnd();
        }
        catch (FormatException) { }
    }
}
```

K předávání informací z business logiky využívám proměnné Progress<T>. Ta se využívá obzvláště pro report progresu z asynchronních metod. Za svůj parametr implementuje moji třídu Status. Ta v sobě zahrnuje text, který chci zobrazit a jeho barvu. O výstup se stará právě metoda AppendLine.

```
Progress<Status> Status = new Progress<Status>(stat => AppendLine(Log1,
stat.message, stat.color));
```

Dále využívám například metody pro kontrolu obsahu TextBoxu pro nastavení počtu zpracovávaných audittrailů. Tato metoda povoluje zapsat pouze číslice do tohoto TextBoxu a znemožní vložení mezery.

Mimo jiné zde také ovládám, které z tlačítek jsou momentálně k dispozici. Například Stop a Dokončit tlačítka nejsou k dispozici, dokud se nezpracovává některý tazatelský soubor, či naopak tlačítka pro zpracování během samotného zpracování již nejsou povolené.

V poslední řadě zde obsluhuji Timer, který ovládá automatické zpracování. Ten je nastaven tak, aby při spuštění zkontrolovat soubory, pokud po spuštění či dokončení předchozího zpracování nenalezne žádný nový, tak se pokusí vyhledávat další soubory s intervalem 5 minut.

4.5.3 Třídy business logiky aplikace

V této kapitole dále rozepíši jednotlivé třídy, které zajišťují funkčnost aplikace. Jedná se o můj pokus odloučit business logiku od uživatelského rozhraní aplikace.

4.5.3.1 IsolatedStorage a Krypt

Třídu IsolatedStorage využívám k zápisům a čtení šifrovaných hesel pro čtění souborů Microsoft databáze uvnitř tazatelských souborů na izolovaná úložiště. K tomu využívám namespace IO.IsolatedStorage, který obsahuje typy pro vytváření a využívání izolovaných úložišt. S těmito úložišti, aplikace může číst a zapisovat data, která méně důvěryhodný kód nebude schopný přečíst. Lze tak díky tomu zabránit úniku citlivých informací ze strany uživatele i assembly v kterém kód existuje. Data jsou navíc izolované doménou. Tyto izolovaná úložiště jsou zcela unikátní pro daný program.

Tento přístup však několik nevýhod. Složitější používání API, čtení a zápis do izolovaných úložišt je možný pouze pomocí IsolatedStorageStreamu, s kterými nelze získat cestu k souboru. V poslední řadě také neumožňuje mazat nebo upravovat soubory založené jiným uživatelem.

Hlavní výhodou je však možnost zapisovat na tomto úložišti nezávisle na uživatelových právech čtení/zápisu/úpravy kdekoliv jinde v počítači.

Ukázka metody pro čtení souboru z izolovaného úložiště:

```
internal static string CtiSoubor(string soubor)
{
    IsolatedStorageFile isoStore = IsolatedStorageFile.GetStore(
    IsolatedStorageScope.User | IsolatedStorageScope.Assembly, null, null);

    string[] fileNames = isoStore.GetFileNames(soubor);

    if(fileNames.Length == 1)
    {
        string textSouboru;
        using(System.IO.StreamReader reader = new System.IO.StreamReader(new
        IsolatedStorageFileStream(soubor, System.IO.FileMode.Open, isoStore)))
        {
            textSouboru = reader.ReadToEnd();
            reader.Close();
        }
        return textSouboru;
    }
    else
    {
        throw new System.IO.FileNotFoundException("Soubor" + soubor +
        "nebyl nalezen.");
    }
}
```

Třída Krypt poté slouží k zašifrování hesel na těchto izolovaných úložištích. Využívá ke své práci namespace Systém.Security.Cryptography, která nabízí služby pro šifrování, zabezpečené kódování a dekódování dat, náhodné generování čísel, ověřování zpráv, hash algoritmus a řadu dalších služeb.

Ukázka metody pro krytování textu:

```
internal static string Kryptuj(string text)
{
    byte[] secret = System.Text.Encoding.Unicode.GetBytes(text);

    byte[] kod = ProtectedData.Protect(secret, Entropie,
    DataProtectionScope.CurrentUser);

    return Convert.ToBase64String(kod);
}
```

4.5.3.2 Vyhledávání souborů ke zpracování

Třída soubor slouží pro ukládání cest ke složkám pro zpracovávání nebo uchovávání souborů a metody pro vyhledávání souborů v adresáři. K vyhledávání zde slouží 3 metody, které navzájem pracují pro vyhledávání souborů. Ty vyhledávají soubory podle následujících kritérií a prioritou podle seznamu:

1. Prioritně prohledává přerušené/nezpracované soubory u kterých se nevyskytla žádná chyba
2. Vyhledává prioritně podle zdroje souboru, tedy soubory od správce ČR mají vyšší prioritu než od Garanty. Ty zase mají vyšší prioritu než od tazatele.
3. Dále porovnává datum změny souboru. To zapříčiní že bude vyhledán vždy nejnovější a nejčerstvější možný soubor.

Třída Soubor dále obsahuje metody pro práci se soubory. Zde jsou gettery pro různé složky souboru. Těmi například mohu zjistit úlohu, období, rok, kvartál, původ dat podle role nebo datum. Tyto gettery dále využívám v metodách pro přesun souborů mezi adresáři.
K přesunu souborů využívám metody z důvodu třídění a filtrování souborů. Ty jsou z původní složky přesouvané buďto do složky kde je soubor archivován po jeho zpracování a nebo je přesunut do složky pro nezpracované soubory. Dále zde mám metody pro práci s kompresovanými soubory pro který využívám NuGet balíček DotNetZip.

Příklad metody pro vyhledávání souboru podle 2. a 3. kritéria, 1. kritérium je řešeno jinou metodou:

```
public string ProhledejZdroj()
{
    string posledniNejnovejsi;
    String[] slogcrSoubory = Directory.GetFiles(AdrZdroj, "*.slogcr");
    if (slogcrSoubory.Length > 0)
    {
        posledniNejnovejsi = slogcrSoubory[0];
        foreach (string slogcr in slogcrSoubory)
        {
            if (GetDatum(slogcr) > GetDatum(posledniNejnovejsi))
                posledniNejnovejsi = slogcr;
        }
        return posledniNejnovejsi;
    }
    else
    {
        String[] slogSoubory = Directory.GetFiles(AdrZdroj, "*.slog");
        if (slogSoubory.Length > 0)
        {
            posledniNejnovejsi = slogSoubory[0];
            foreach (string slog in slogSoubory)
            {
                if (GetDatum(slog) > GetDatum(posledniNejnovejsi))
                    posledniNejnovejsi = slog;
            }
            return posledniNejnovejsi;
        }
        else
        {
            String[] sloSoubory = Directory.GetFiles(AdrZdroj, "*.slo");
            if (sloSoubory.Length > 0)
            {
                posledniNejnovejsi = sloSoubory[0];
                foreach (string slo in sloSoubory)
                {
                    if (GetDatum(slo) > GetDatum(posledniNejnovejsi))
                        posledniNejnovejsi = slo;
                }
                return posledniNejnovejsi;
            }
        }
    }
    throw new FileNotFoundException("Nebyl nalezen žádný soubor
    ke zpracování.");
}
```

4.5.3.3 Zpracování

Poslední je statická třída Zpracovani potom řídí celou business logiku hlavního cíle této aplikace, a sice zpracovávání jednotlivých souborů. V této kapitole popíši některé technologie využívané při zpracování a celkový postup kódu zpracovávání souborů.

K oddělení samotné činnosti zpracování souboru od uživatelského prostředí, aby nedošlo k jeho zaseknutí a čekání na výpočet, řeším pomocí asynchronních metod a klíčových slov async a await. Prvotní spouštění těchto procesů však začínají už v WPF code-behind. Tomto bohužel bez využití nějakého návrhového vzoru nešlo předejít.

Figurují zde 3 metody. Metoda, která je volaná z WPF button event handlerů funguje jako hlavní schéma podle kterého je řešeno celé zpracování daného souboru. Další dvě metody pak slouží ke zpracování speciálně pagin a kontaktů ze souboru a pro zpracování jednotlivých audittrail souborů a nalezení chyb mezi nimi.

```
public async static Task ZpracujSoubor(CancellationToken Zastavit, IProgress<Status>
Progress, Soubor Adresar, string id, string password, int pocetAsync)
```

Toto je hlavička hlavní metody, kterou bych zde rád rozebral a popsal. V první části kódu si vytvářím pomocné proměnné, které v sobě uchovávají užitečné informace o vybraném zpracovávaném souboru. Z těchto například poté tvořím názvy tabulek Oracle nebo je využívám jako parametrů pro sestavování samotných SQL dotazů. K oddělení těchto informací slouží právě gettery složek souboru z třídy Soubor. Také si zde sestavím cestu dočasné temp složky, určené k rozzipování souboru.

Ve druhé části figuruje pouze metoda rozzipování z třídy Soubor, která extrahuje soubor do nastavené dočasné složky.

Ve třetí části metody nastupuje zpracovávání pagin a kontaktů, k tomu využívám další metody ZpracujPaginy. Tato metoda si otevře pomocí zašifrovaného hesla na izolovaném disku MDB databáze ze zpracovávaného souboru a připraví si data k nahrání do Oracle databáze do pomocných DataTable proměnných. Vytvoří žádoucí tabulky v případě že neexistují. V poslední řadě v případě existence pagin v tabulce tyto hodnoty pouze aktualizuje, pokud však v tabulce zatím nefigurují, vloží je jako nové řádky.

Ve čtvrté části se zpracovávají Audittraily. Soubor může obsahovat větší množství Audittrail souborů a zpracování bylo naprogramováno, aby umožňovalo jejich paralelní zpracování. Toho jsem docílil využitím listu Tasků, do kterých vkládám metodu pro zpracování Audittrailu neboť má návratovou hodnotu typu Task. U těch v cyklu očekávám jejich zpracování, po kterém vkládám na jejich místo zpracování dalšího souboru. Tím

docílím dynamické paralelní zpracování, které pokaždé zpracovává dle určeného limitu maximální možný počet audittrailů najednou. Toto značně zvyšuje rychlost a efektivitu programu oproti jeho předchůdci a v případě vyšších procesorových zdrojů lze zrychlit aplikaci více zvednutím limitu počtu paralelního zpracování.

Metoda ZpracujPaginy, která je volána v hlavní metodě zpracování, slouží ke zpracování pagin a kontaktů. Ta nejprve maže staré záznamy kontaktů k nahrání nových, aby nedošlo k duplikaci, následně doplní nové kontaktní údaje a nové údaje pagin či stávající aktualizuje.

Metoda ZpracujAudittraily, která je také volána hlavní metodou ZpracujSoubor, slouží nejprve k filtrování audittrail souboru a nalezené chyb. K tomu slouží objekt TextFileParser, který poskytuje metody a vlastnosti pro analýzu strukturovaných textových souborů. Také usnadňuje rozdělování stringu pomocí znaku díky vlastmosti delimiter a nemusí kvůli tomu využívat String.Split. Tato metoda dále profiltruje audittrail soubor podle zadaných kriterií a v případě, že během parsování došlo k nějaké chybě, jsou údaje přesunuty do tabulky Audittrail_Chyba. Nakonec údaje Audittrail i Audittrail_Chyba vloží do databáze Oracle.

Příklad kódu paralelního zpracování audittrailů, kterému předchází cyklus pro naplnění listu Tasků do limitu zadaným uživatelem, jejich prvotním spuštěním a uložení jejich dat do databáze Oracle:

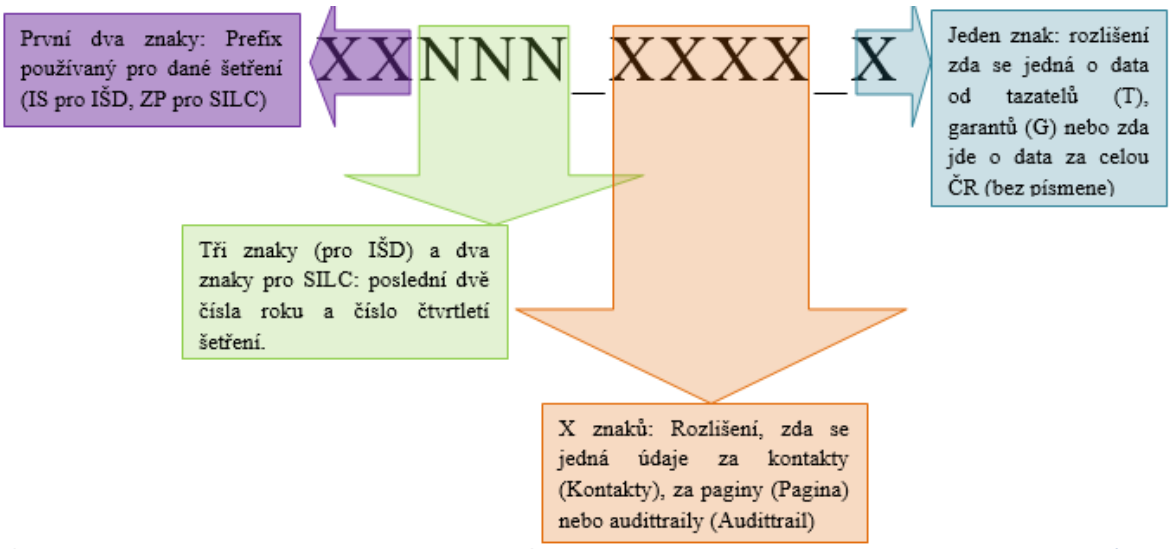
```
for (int i = 0; i<pocetAsync; i++)
{
    if (posledniAdt<Audittraily.Length && tasks[i] != null &&
        tasks[i].IsCompleted)
    {
        await Task.Run(() => tasks[i]);

        if (Zastavit.IsCancellationRequested)
        {
            Adresar.MoveFile(soubor, Progress);
            Zastavit.ThrowIfCancellationRequested();
        }

        string AdtBezPripony = Path.GetFileNameWithoutExtension(Audittraily
        [posledniAdt]);
        if (AdtBezPripony.Contains("_"))
            AdtSoubor = Path.GetFileName(Audittraily[posledniAdt]);
        else
            AdtSoubor = string.Format("{0}_{1}.adt", AdtBezPripony, TazSoubor);

        Progress.Report(new Status(String.Format("{0} –
        zpracování audittrailu {1}.", nazevSouboru, Path.GetFileName(
        Audittraily[posledniAdt])), "black"));

        // Maže data z tabulky Audittrail a Audittail_Chyba podle názvu .adt
        souboru pro případ, že bylo zpracování přerušeno předčasně
        orDbAudittrail.SmazatData(TabAudittrailNazev, AdtSoubor);
        orDbAudittrailChyba.SmazatData(TabAudittrailChybaNazev, AdtSoubor);

        tasks[i] = ZpracujAdt(Audittraily[posledniAdt], AdtSoubor,
        orDbAudittrail, TabAudittrailNazev, orDbAudittrailChyba,
        TabAudittrailChybaNazev, orDbKontakt, TabKontaktNazev, Zastavit,
        dtKontakty, Progress, Adresar, soubor);

        posledniAdt++;
    }
}
if (posledniAdt == Audittraily.Length)
{
    foreach (Task ukol in tasks)
    {
        if (ukol != null)
               await ukol;
    }
}
```

5 Výsledky a diskuse

5.1 Výsledná aplikace

Cíl programu je umožnit a vylepšit průběžné monitorování domácnostních šetření (například Integrovaného šetření u domácností - IŠD, Výběrové šetření o zdraví – EHIS, Výběrové šetření příjmů a životních podmínek - SILC). Hlavní podmínka funkčnosti programu je zajištění, aby tazatelé, garanti a správce ČR zasílali své výstupní soubory (tedy soubory s příponou slo, slog a slogcr). Ty jsou zasílány k dalšímu zpracování přes centrální úložiště v předem stanovených intervalech. Minimální intenzita odesílání těchto výstupních souborů je určována metodiky úloh.

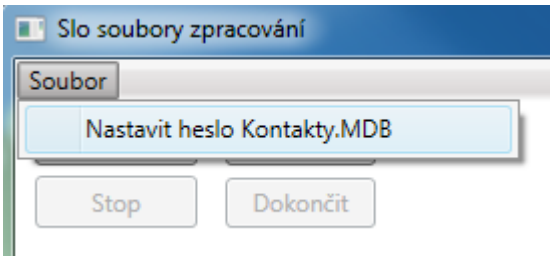
Program CO_Zpracování automaticky zpracovává údaje z nových slo, slog a slogcr souborů, které se dostanou na centrální úložiště od tazatelů, garantů nebo správce ČR. Program je uzpůsoben k nepřetržitému monitorování výskytu nových souborů na úložišti a přenáší údaje o skutečných kontaktech a šetřených paginách z MDB databáze Kontakty. Také přenáší údaje z Audittrailu (soubory s příponou .adt umístěne v zpracovávaných souborech) do tabulek v databázi Oracle. Před vložením nových dat jsou starší data z Oraclu smazána. U tabulek s paginami a kontakty se mažou data podle názvů tazatelských souborů v případě slo souborů, podle čísla kraje v případě slog souborů nebo všechan data v případě slogcr souboru. U audittrailu se mažou pouze data podle názvu právě zpracovaného audittrailu aby se zabránilo duplicitním datům v tabulce.



Obrázek 8 – Pojmenování tabulek

Například údaje pocházející ze slo souborů jednotlivých tazatelů z tabulky Kontakty databáze Kontakty z úlohy IŠD za její 1. čtvrtletí roku 2020 budou uložený v tabulce: *IS201_Kontakty_T*.

Po spuštění programu, v případě že je uživatelem program spuštěn poprvé, je uživatel vyzván k zapsání hesla k přístupu do MDB databáze Kontakty, která je přítomna ve všech tazatelských souborech. V případě, že uživatel heslo zadal špatně nebo se změnilo, může využít menu ve vrchní liště k jeho změně.



Obrázek 9 – Nastavení nového hesla k souboru MDB

Heslo je uloženo v zašifrované podobě na izolovaném úložišti, z kterého si program při své práci heslo automaticky bere. Před samotným zahájením zpracovávání je také nutné v hlavním okně programu vyplnit přihlašovací údaje do databáze Oracle. Dále je možné si nastavit počet zpracovávaných audittrailů, který určuje, kolik adt souborů se bude zpracovávat najednou. To však výrazně spotřebovává zdroje procesoru a mělo by se využívat přiměřeně. Defaultní hodnota je nastavena na 5 adt souborů najednou a lze nastavit maximálně hodnotu 99.

Adresáře pro zpracování souborů by měli být nastaveny defaultně, ale dají se změnit podle potřeby uživatele. Popis adresářů pro zpracování:

- Zdroj – Do tohoto adresáře jsou jinou aplikací automaticky a nepřetržitě aktualizovány všechny soubory z centrálního úložiště. Při automatickém zpracování aplikace na tomto adresáři (v případě že nejsou přípustné soubory ke zpracování v adresáři Nezpracované) zpracovává nejnovější soubory prioritně podle jejich původu a veškeré nové, které se zde objeví.
- Zpracovávání – Toto je dočasný adresář, do kterého je zpracovávaný soubor kopírovaný a rozzipován z adresáře Zdroj. Tento adresář tedy obsahuje pouze rozbalené soubory právě zpracovávaného souboru.
- Zpracované – Tento adresář obsahuje úspěšně zpracované soubory z adresáře Zpracovávání, tedy soubory jejichž data byly úspěšně nahrány do databáze Oracle. Soubory jsou zde tříděny podle úlohy, roku a kvartálu.

- Nezpracované – Do tohoto adresáře se přesunují soubory, při jejichž zpracovávání došlo k chybě a neprošli kontrolou nebo byly přerušeny. V případě, že došlo k chybě, je k souboru vložen textový dokument s příponou slot, který obsahuje veškeré chybové hlášení. V případě že soubor slot chybí, došlo pouze k přerušení a při automatickém zpracovávání mají tyto soubory přednost před Zdrojem.

Okno logu zaznamenává a zobrazuje jednotlivé akce mapující automatické nebo ruční zpracování souboru. Tlačítkem a comboboxem na spodu okna lze celý dosavadní záznam uložit do textového souboru nebo souboru RTF. Tyto zprávy jsou barevně odlišeny:

- Zelená slouží k mapování zahájení a konec zpracování souboru.
- Červená zobrazuje závažné chyby při zpracování.
- Oranžová vkládá varování. Jedná se o oznámení například o tom, že nebyl nalezen soubor s audittraily.
- Modře jsou odlišeny zprávy o přerušení zpracovávání po stisknutí tlačítka Stop.
- Černá slouží k jednotlivým akcím samotného zpracovávání, například informuje o rozzipování, zpracovávání kontaktů a tak dále.

Aplikace se ovládá 4 tlačítkami:

- Tlačítko Zpracuj vyvolá okno s filtrami pro slo, slog a slogcr soubor a při vybrání souboru ho zpracuje. Při ručním zpracování nehrají roli nastavené adresáře, uživatel může soubor vybrat odkudkoliv.
- Auto zpracuj slouží ke spuštění automatického zpracovávání.
- Stop slouží k okamžitému zastavení zpracovávání souboru. Takový soubor bude bez chybové hlášky nahrán do adresáře Nezpracované.
- Dokončit dokončí právě se zpracovávaný soubor, a poté přeruší další vyhledávání automatického zpracovávání. V případě že aplikace pouze čeká na nové soubory s prázným adresářem pozastaví Timer.

5.2 Alternativní technologie, návrhové vzory a testování

Původním záměrem bylo pro navázání spojení s databází využít Entity Frameworku, na jehož základě jsem vytvářel původní zadání diplomové práce. Avšak při návrhu aplikace se Entity Framework ukázal jako zbytečně komplexní řešení pro potřeby mé aplikace. Jelikož nepotřebuji vytvářet žádný komplexní databázový model s relacemi a implementace celého Frameworku mě přišlo zcela zbytečně, zvolil jsem pro práci se vzdálenou databází více primitivní metody v podobě knihoven OracleDataAdapter a OracleCommand. Tyto knihovny jsou sice již trochu zastaralé, ale stále se dají nalézt v řešení aplikací a plně vyhovovali mým požadavkům pro práci s daty.

Dnešním standardem desktopové aplikace je návrhový vzor MVVM (Model-View-ViewModel), který rozděluje aplikaci na modelovou, grafickou a aplikační stránku. Jedná se o obdobu webového MVC (Model-View-Controller), avšak je zde několik odlišností. Hlavním odlišujícím parametrem, proč jsem se rozhodl tento návrhový vzor nevyužít, je jeho nepodpora ve vývojovém prostředí Visual Studio. Návrhový vzor MVC je defaultně podporován Visual Studiem, avšak k využití návrhového vzoru MVVM bych vyžadoval dodatečné frameworky. Avšak oddělování grafického rozhraní, modelu a business logiky aplikace jsem se i přesto snažil přizpůsobit. Dalším záporem nevyužití návrhového vzoru MVVM je však velmi obtížná integrace unit testů pro ovládání grafického rozhraní. Testování aplikace bylo z toho důvodu prováděno primárně prokováním chodu aplikace v debug režimu.

6 Závěr

Cílem této diplomové práce bylo vytvořit specifikovanou desktopovou aplikaci, která by uživatelovi Českého statistického úřadu dovolila na pozadí jeho pracovní stanice a běžné pracovní náplně zpracovávat tazatelské soubory jednotlivých výběrových domácnostních šetření. Aplikace splnila veškeré specifikované požadavky zadání, avšak je zde pár stránek aplikace, které by šli do budoucna vylepšit. Jmenovitě se jedná například o lepší ochranu přihlašovacích údajů, která je sice dostačující, ale mohla by být mnohem přísnější. Dalším potenciálním vylepšením by mohlo být uživatelem nastavitelný filtr pro vyhledávání tazatelských souborů.

Aplikace byla vytvořena převážně pro operační systém Windows, a nebyl proto testován na jiné operační systémy. Rozhraní je uzpůsobeno klasické Windows desktopové aplikaci a vzhledem k relativní náročnosti na spojení s databází se nepředpokládá provoz na tabletu nebo mobilním zařízení.

Při vývoji byly využity převážně online zdroje z důvodu rychle vyvíjejících se technologií a postupů, na které knižní zdroje nejsou dostatečně schopné rychle reagovat.

7 Seznam použitých zdrojů

[1] MACDONALD, Matthew. *Pro WPF in C# 2010*. Springer-Verlag Berlin and Heidelberg GmbH & Co., 2010. ISBN 1430272058.

[2] SKEET, Jon. *C# in Depth, 4E*. Manning Publications, 2019. ISBN 9781617294532.

[3] VIRIUS, Miroslav. *Cvičení k programování pro .NET*. CVUT Praha, 2019. ISBN 978-80-01-06546-4.

[4] VIRIUS, Miroslav. *Programování pro .NET*. CVUT Praha, 2011. ISBN 978-80-0104-864-1.

[5] SARCAR, Vaskaran. *Design Patterns in C#*. APress, 2018. ISBN 1484236394.

[6] KOCAN, Marek. *Co je a k čemu je Microsoft .NET* [online]. 6. října 2001 [cit. 2020-03-30]. Dostupné z: https://www.zive.cz/clanky/co-je-a-k-cemu-je-microsoft-net/sc-3-a-103190/default.aspx

[7] *What is .NET?* [online]. [cit. 2020-03-30]. Dostupné z: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

[8] *Overview of .NET Framework* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview

[9] *What is "managed code"?* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/managed-code

[10] *Přehled grafického subsystému WPF* [online]. 04. 11. 2016 [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/framework/wpf/introduction-to-wpf

[11] *PL/SQL Procedure* [online]. [cit. 2020-03-30]. Dostupné z: https://www.oracletutorial.com/plsql-tutorial/plsql-procedure/

[12] *Database 2 Day + .NET Developer's Guide* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.oracle.com/cd/B28359_01/appdev.111/b28844/toc.htm

[13] *Data Provider for .NET Developer's Guide* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.oracle.com/cd/B28359_01/win.111/b28375/OracleCommandClass.htm

[14] COOK, John Paul. *Build a .NET Application on the Oracle Database with Microsoft Visual Studio 2010* [online]. červen 2011 [cit. 2020-03-30]. Dostupné z: https://developer.oracle.com/dotnet/vs2010-oracle-dev.html

[15] *DotConnect for Oracle Documentation* [online]. [cit. 2020-03-30]. Dostupné z: https://www.devart.com/dotconnect/oracle/docs/Devart.Data.Oracle~Devart.Data.Oracle.OracleDataAdapter.html

[16] *DataSet Class* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.8

[17] *OracleDataAdapter Class* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/api/system.data.oracleclient.oracledataadapter?view=netframework-4.8

[18] *Isolated Storage* [online]. 30. 3. 2017 [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/io/isolated-storage

[19] ALBAHAR, Joseph. *Isolated Storage* [online]. [cit. 2020-03-30]. Dostupné z: http://www.albahari.com/nutshell/IsolatedStorage.pdf

[20] AMUNDSEN, Mike. *Take Advantage of Isolated Storage with .NET* [online]. [cit. 2020-03-30]. Dostupné z: https://www.codeguru.com/csharp/.net/net_data/datagrid/article.php/c8553/Take-Advantage-of-Isolated-Storage-with-NET.htm

[21] *C# Language Specification* [online]. prosinec 2017 [cit. 2020-03-30]. Dostupné z: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf

[22] ČÁPKA, David. *Úvod do WPF* [online]. [cit. 2020-03-30]. Dostupné z: https://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace/

[23] *Overview of XAML* [online]. 10. 1. 2020 [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/cs-cz/visualstudio/xaml-tools/xaml-overview?view=vs-2019

[24] *What is WPF?* [online]. [cit. 2020-03-30]. Dostupné z: https://www.wpf-tutorial.com/about-wpf/what-is-wpf/

[25] JECHA, Tomáš. *Úvod do windows presentation foundation* [online]. 19. 1. 2012 [cit. 2020-03-30]. Dostupné z: https://www.dotnetportal.cz/clanek/196/Uvod-do-Windows-Presentation-Foundation-WPF-

[26] *C# documentation* [online]. [cit. 2020-03-30]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/

[27] HAMILTON, Naomi. *A-Z of programming languages* [online]. Computerworld, 2008-2010 [cit. 2020-04-01]. Dostupné z: http://www.math.bas.bg/bantchev/misc/az.pdf

[28] *Asynchronous programming with async and await* [online]. 18. 3. 2019 [cit. 2020-04-01]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/

[29] JOSHI, Nimit. *Async and Await in Asynchronous Programming in C#* [online]. 30. června 2014 [cit. 2020-04-01]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/4b0136/async-and-await-in-asynchronous-programming-in-C-Sharp/

[30] CLEARY, Stephen. *Reporting Progress from Async Tasks* [online]. 16. února 2012 [cit. 2020-04-01]. Dostupné z: https://blog.stephencleary.com/2012/02/reporting-progress-from-async-tasks.html

[31] HOLAN, Tomáš. *Princip async/await z jazyka c# 5.0* [online]. 24. 9. 2012 [cit. 2020-04-01]. Dostupné z: https://www.dotnetportal.cz/blogy/15/Null-Reference-Exception/5304/Princip-async-await-z-jazyka-C-5-0

[32] *Základy relačních databází, jejich využití v programování webu* [online]. VŠE, 16. 8. 2018 [cit. 2020-04-01]. Dostupné z: https://kme.vse.cz/wp-content/uploads/page/534/4.-Z%C3%A1klady-rela%C4%8Dn%C3%ADch-datab%C3%A1z%C3%AD-jejich-vyu%C5%BEit%C3%AD-v-programov%C3%A1n%C3%AD-webu.pdf

[33] *SQL Tutorial* [online]. w3schools, 16. 8. 2018 [cit. 2020-04-01]. Dostupné z: https://www.w3schools.com/sql/default.asp

[34] *Introduction to the C# language and the .NET Framework* [online]. [cit. 2020-04-01]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework