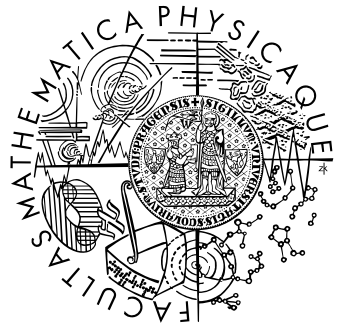
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Císař

Software pro správu sběratelských kolekcí

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Jiří Dokulil

Studijní program: Informatika - Programování

2008

Děkuji Mgr. Jiřímu Dokulilovi za cenné rady, které mi pomohly při zpracování této práce. Poděkování patří i RNDr. Davidu Hokszovi za zpřístupnění MS SQL serveru Sportka pro účely testovacího provozu. Zároveň bych chtěl poděkovat MgA. Jakubu Turečkovi za původní námět a praktické rady z oblasti problematiky.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 11.12.2008 Jan Císař

Název práce: Software pro správu sběratelských kolekcí

Autor: Jan Císař

Katedra (ústav): KSI

Vedoucí bakalářské práce: Mgr. Jiří Dokulil, KSI

e-mail vedoucího: jiri.dokulil@mff.cuni.cz

Abstrakt: Cílem práce je implementovat katalogovou utilitu určenou pro správu údajů o sběratelských kolekcích (zejména mincí) a usnadnění jejich doplňování o chybějící kusy. Program umožňuje porovnávat seznam předmětů ve sbírce se vzorovou sbírkou uloženou v databázi a vypisovat chybějící předměty. Obsahuje také funkce pro export a tisk různých typů přehledů o uložených sbírkách a předmětech. Aplikace komunikuje se vzdálenou databází, uchovává data o sbírkách více uživatelů přihlašujících se pod unikátním přístupovým jménem.

Klíčová slova: C#, .NET, sbírka, mince, katalog

Title: Collection maintenance software

Author: Jan Císař

Department: KSI

Supervisor: Mgr. Jiří Dokulil, KSI

Supervisor's e-mail address: jiri.dokulil@mff.cuni.cz

Abstract: Goal of the thesis is to implement a catalogue tool for collection (with special focus on coin collections) maintenance. The program allows to compare user's collections with model collections stored in the database and export lists of missing items. In addition to that it also contains functions designed to print and export various reports summarizing stored collections and items. The program communicates with remote database which contains data of registered users accessing it through unique login name.

Keywords: C#, .NET, collection, coins, catalogue

1. Úvod

K myšlence zpracovat v rámci studia softwarové řešení pro katalogizaci a případné sdílení sběratelských kolekcí (především sbírek mincí) mě přivedl můj dřívější spolupracovník MgA. Jakub Tureček, toho času příležitostný přispěvovatel do odborného časopisu „Mince a bankovky“. Po vlastním rozboru problematiky jsem dospěl k závěru, že je možné pro tento účel zpracovat rozumně standardizovanou aplikaci, kterou lze postupem času dále rozšiřovat (nemincovní sbírky, možnost zobrazení sbírek jiných uživatelů, funkce pro import a podobně).

Využitelnou výhodou je v tomto případě i poměrně snadná dostupnost precizně zpracovaných historických seznamů vydaných mincí, jejichž vložení do centrální databáze umožňuje snadné zjištění chybějících mincí do zkompletování sbírky a obecně tak přináší značné usnadnění oproti porovnávání s často nepřehlednými seznamy v „papírovém“ formátu.

Zvolené řešení navíc umožňuje tvorbu přehledů pro export a tisk různě strukturovaných a podrobných seznamů pro zlepšení prezentace vlastních sbírek a usnadnění dalšího nakládání s nimi.

Motivací pro mě byla i možnost dokončenou aplikaci v budoucnu nabídnout na CD jako přílohu časopisu Mince a bankovky a tedy šance vyzkoušet si v praxi veřejné publikování softwarového projektu.

1.1. Struktura bakalářské práce

Text v druhé kapitole se zabývá stručným rozborem problematiky a nastiňuje některé typické situace, kdy by měla být aplikace nápomocná. Kapitola 3 obsahuje základní informace o způsobu implementace a také zdůvodnění výběru jazyka, platformy, databáze, vývojového prostředí, způsobu propojení aplikace s databází, tisku přehledů a dalších metod a postupů. Popis GUI, uživatelská dokumentace a několik obecných návodů pro práci s aplikací se nachází v kapitole 4. Podrobnějšímu rozebrání z programátorského hlediska je věnována pátá kapitola. V kapitole 6 jsou uvedeny postupy při tvorbě DB a její

schéma. Kapitola 7 stručně rozebírá autorovi známé existující aplikace s podobným účelem. Závěrečné shrnutí a v neposlední řadě některé další myšlenky a varianty budoucího vývoj aplikace jsou nastíněny v kapitole 8. Součástí práce jsou i čtyři přílohy, příloha A shrnuje obsah přiloženého CD, příloha B obsahuje obsahuje popis některé vybraných databázových pohledů, příloha C popisuje postup instalace databáze pro aplikaci a konečně v příloze D je popsána struktura CSV souborů pro import a export.

V celém textu práce jsem se snažil důsledně pracovat s českými ekvivalenty cizozajazyčných technických výrazů.

2. Úvod do problematiky

Tato kapitola stručně rozebírá problematiku tvorby sběratelských kolekcí a nastiňuje některé možné postupy při uložení a katalogizaci údajů v relační databázi.

2.1. Postupy při sbírání mincí

Většina sběratelů obecně dodržuje určitý řád budování jednotlivých kolekcí, který si stanoví před počátečním vytvořením sbírky, nebo např. se získáním prvního kusu předmětu, vhodného pro založení takové sbírky. Při sbírání mincí je takové první nastavení typu sbírky obvykle neměnné a sběratel se tak už při jejím založení rozhoduje například pro:

- Sbírání všech typů mincí z jedné země a období (např. mince meziválečné ČSR)
- Sbírání všech ročníků mincí z jedné země a období
- Sbírání mincí ze všech mincoven, působících v jednom období v jedné zemi
- Kombinace výše uvedeného (např. všechny typy a ročníky)

Výše uvedené čtyři body představují případy, kdy je možné na základě pouhého popisu sbírky poměrně jednoznačně určit, které mince by do ní měly patřit, aby byla tato sbírka úplná. Je ale samozřejmě možné (a také časté), že sběratel tvoří sbírku podle jiného, komplikovanějšího nebo méně jednoznačného klíče. Taková sbírka je často nedokončitelná ve smyslu svého „naplnění“ – těžko např. vytvořit seznam všech dostupných mincí s vyraženým obrázkem lodi. Typickými příklady takového postupu při tvorbě sbírky jsou zejména následující:

- Sbírání variant jedné mince (řada mincí jednoho ražených po delší období vykazuje drobné odlišnosti nebo změny v ražbě, různé vady aj.)

- Sbírání mincí podle společného znaku v ražbě (např. všechny mince, na kterých je vyražena loď, zvíře, podobizna panovníka a podobně)
- Sbírání mincí podle společného znaku ve skladbě kovu (typicky drahé kovy, zejména zlato a stříbro, časté jsou i sbírky bimetalických nebo jinak neobvyklých mincí)
- Sbírání mincí na základě osobních vazeb a podobně (např. mince ze všech zemí, které dotyčný v životě navštívil)

V podobných případech je samozřejmě obtížné stanovit nějakou konečnou podobu sbírky, o to potřebnější je ale pečlivé zpracování katalogu a vedení záznamů.

2.2. Katalogizace sbírek

Běžnou a důležitou praxí při tvorbě kolekcí je alespoň základní forma jejich katalogizace – je obvyklé vést si minimálně pro osobní potřebu seznam svých sbírek a pro každou z těchto sbírek pak seznam předmětů, které k ní náleží (nemusí se ale nutně jednat o písemný seznam, častou formou „katalogizace" je např. ukládání do oddělených alb). Typický sběratel si tak vede záznamy o svých sbírkách způsobem, který je ve své podstatě poměrně blízký uložení údajů do relační databáze. Použití nějaké takové databáze se tedy jeví jako poměrně přímočará varianta, jak zajistit reprezentaci takového katalogu v elektronické poodbě.

Pokud dále analyzujeme standardní sběratelské návyky z pohledu katalogizace údajů v relační databázi, narazíme na další fakta, která hovoří pro zmíněný postup.

- U jednotlivých množin n-tic (*sbírky*, *předměty*…) je možné definovat atributy
- Mezi jednotlivými množinami n-tic (*sbírky*, *předměty*…) existují jednoznačné vztahy

Je například zřejmé, že příslušnost předmětu k nadřazené kolekci je jednoznačná a řídí se vztahem N : 1, tedy kolekce může obsahovat (a obvykle také obsahuje) více předmětů, ale předmět patří vždy nejvýše do jedné kolekce. Tato situace je dána tím, že kompletní sbírka je uceleným souborem, určeným ke společnému uložení a případné prezentaci (těžko si představit situaci, že by sběratel před prezentací své sbírky přesouval předměty ze sbírky jiné, již odprezentované).

Jako příklad můžeme uvést imaginárního sběratele, který vytváří sbírku mincí podle následujícího klíče [1]:

- Původ: ČSR
- Historické období: 1918 – 1939
- Typ sbírky: všechny typy a ročníky

Pro zjednodušení příkladu předpokládejme, že dotyčný sbírá pouze haléřové mince. Do sféry jeho zájmu tak spadají mince 2h, 5h, 10h, 20h, 25h a 50h s datem ražby 1921 – 1938 včetně (v jiných letech nebyly mince raženy). Typů mincí je v tomto případě 6, kompletní sbírka by měla podle [1] obsahovat celkem 53 ks mincí[1]. Pokud by se ale dotyčný rozhodl navíc ještě vytvořit sbírku stejného původu a období, složenou ovšem pouze z typů (pomiňme, že by to zřejmě nemělo praktický význam, neboť sbírka kombinující typy a ročníky zároveň je v praxi vždy cennější), musel by výše zmíněnou šestici typů získat ještě jednou a zařadit ji do oddělené sbírky.

Příkladem typického atributu uváděného u sbírky mincí je kvalita jednotlivých kusů. Ta je obvykle hodnocena pomocí standardizované škály [1], jejíž stupně jsou přibližně celosvětově odpovídající[2]. V České Republice se většinou dle [2] používá následující stupnice:

[1] U některých typů probíhala ražba po kratší dobu – např. 25h mince byla ražena jen v letech 1932 a 1933

[2] Někdy je anglická stupnice uváděna až do stupně kvality P (Poor)

- **Proof** – ražba z leštěných razidel (pouze u moderních mincí), odpovídá německému hodnocení PP (Polierte Platte)
- **0** – s ražebním leskem (RL), mince nebyla v oběhu, bez jakýchkoliv poškození, odpovídá německému St. (Stempelglanz nebo Stempelfrisch) nebo anglickému UNC (Uncirculated)
- **1** – krásná, s nepatrnými stopami oběhu (viditelnými pod lupou) a s malými chybami, odpovídá vz., vorz. nebo vzgl. (Vorzüglich), EF nebo XF (Extremely fine)
- **2** – horší, znatelné stopy oběhu (viditelné okem) s většími vadami, odpovídá ss nebo s.sch. (Sehr schön), VF (Very fine)
- **3** – ještě dobrá, čitelná, stopy oběhu větší, jasně zřetelné detaily, odpovídá s, sch. (Schön), F (Fine)
- **4** – již hodně otřelá, místy nečitelná, vyjímečně zařaditelná do sbírky pro svou velkou vzácnost jako dokladový materiál, odpovídá s.g.e. (Sehr gut erhalten), G (Good).

V tomto případě se opět nabízí reprezentace v tabulce relační databáze pomocí číselníkové tabulky a následné propojení cizím klíčem s tabulkou, obsahující jednotlivé předměty.

Po seznámení se s typickými situacemi lze další postupnou analýzou této problematiky dospět až k hrubému nástinu možného datového modelu a na jeho základě následně vytvořit databázové schéma, které je podrobněji rozebráno v kapitole 6.

3. Základní informace o programu

V této kapitole je obsažena technická specifikace aplikace a také zdůvodněn výběr technologií a postupů použitých při jejím vytvoření, včetně popisu možných nevýhod zvolených řešení a případných jiných variant.

3.1. Stručný technický a funkční popis

3.1.1. Platforma a operační systém

Program je zpracován ve vývojovém prostředí Microsoft Visual Studio 2005 v programovacím jazyce C# nad platformou Microsoft .NET Framework 2.0. Jako databázový server využívá Microsoft SQL Server 2005 (ve volně dostupné verzi Management Studio Express) a použitým dotazovacím jazykem je tedy Transact-SQL. Operačními systémy umožňujícími používání programu jsou všechny systémy podporující platformu Microsoft .NET Framework 2.0, v současnosti (dle [4]): Windows Vista, Windows XP, Windows 2000, dle [5] navíc také Windows 98 / Me.

3.1.2. Funkcionality

Základní funkcionalitou programu je zobrazení editovatelného seznamu jednotlivých sbírek (je možné sbírky přidávat, upravovat a mazat) příslušného uživatele. Ke každé jednotlivé sbírce lze zobrazit kartu s podrobnými údaji, a také tabulku v ní obsažených předmětů. Tyto předměty je možné procházet, zobrazovat jejich podrobné karty (s rozdílným vzhledem pro mincovní a nemincovní sbírky) a rovněž provádět jejich editaci. V případě existence *typické sbírky*[3] uložené v DB je možné porovnávat vytvářenou vlastní sbírku s touto typickou sbírkou a mít tak přehled o chybějících předmětech. Doplňujícími funkcionalitami jsou součásti pro export a tisk přehledů, administrátorský modul

[3] *Typickou sbírkou* je myšlena vzorová sbírka vytvářené sbírky, *typizovanou sbírkou* pak samotná vytvářená sbírka

pro editaci uživatelských účtů a modul pro vytváření a import vlastních typických sbírek. Podrobnější popis programu z uživatelského hlediska se nachází v kapitole 4, programátorské aspekty pak v kapitole 5.

3.2. Zdůvodnění volby .NET Framework 2.0

Pro platformu .NET 2.0 jsem se rozhodl především proto, že v rámci své *Base Class Library* poskytuje velmi širokou paletu tříd a připravených řešení zejména v oblasti GUI, komunikace s databází a přístupu k datům – její použití tedy znamenalo značné zpřehlednění práce zejména při prvotním návrhu aplikace. Roli hrála i má předchozí zkušenost s prací nad touto platformou. Za zmínku také stojí, že GUI komponenty .NET a způsob práce s nimi je v předpokládané „informaticky neodborné" uživatelské komunitě obecně známý a usnadňuje tak budoucí využití aplikace. Od verze 1.1 navíc došlo k podstatnému rozšíření tříd pro práci s daty (zejména pro variantu tzv. „odpojené aplikace", viz níže) a připojení k databázi.

3.3. Možné nevýhody .NET Framework 2.0

Zjevnou nevýhodou .NET Framework 2.0 je jeho téměř exkluzivní provázanost s operačními systémy firmy Microsoft, možnost portování např. na operační systém Linux je zatím velmi omezená a výsledek není příliš spolehlivý. Na jiných operačních systémech také neexistuje plnohodnotná náhrada za MS Visual Studio jako vývojového prostředí z hlediska programátorského komfortu.

Ve starších verzích Windows (XP Service Pack 1 a starší) není .NET Framework 2.0 standardně nainstalován a před spuštěním aplikace je tedy nutné ho doinstalovat, což může potenciálního uživatele odradit od používání aplikace.

Zejména na starší počítače vznáší použití .NET nemalé hardwarové nároky, především z důvodu JIT překladu kódu, použití Garbage Collectoru apod. To může vést k občasnému zpomalení aplikace a opět snížení komfortu jejího užívání [5].

S ohledem na to, že .NET 2.0 je již delší dobu k dispozici a od doby jeho prvotního uvedení se výkonnost hardwareového vybavení výrazně zvýšila (a naopak míra penetrace Windows XP SP1 nebo dokonce starších mezi uživateli poklesla), nezdály se mi být výše uvedené nevýhody velkou překážkou.

3.4. Zdůvodnění volby MS SQL Server

Při výběru databáze jsem zvažoval dva hlavní kandidáty – MS SQL a MySQL. Ve prospěch MS SQL hovořilo o něco snažší propojení s .NET (i když použití ODBC případné problémy do značné míry řeší), naopak pro MySQL poměrně snadná dostupnost různých zdarma poskytovaných řešení pro veřejné umístění databáze (služba *freesql.com* apod.). Případné výkonnostní rozdíly jsem příliš nebral v úvahu s ohledem na to, že s ohledem na poměrně úzkou cílovou skupinu uživatelů nepředpokládám velký rozsah a vytížení databáze při případném zveřejnění aplikace. Ostatně, ani v případě vyšší uživatelské zátěže nevychází takové srovnání jednoznačně [6]. Hlavní roli v rozhodnutí zvolit MS SQL tak nakonec hrála snažší správa konzistence databáze (cizí klíče a podobně) a také má předchozí zkušenost s jeho používaním a znalost výše zmíněného vývojového prostředí.

3.4.1. Varianta embedded databáze

Při analýze problematiky jsem narazil i na možnost lokálního používání aplikace (bez připojení k centrální databázi). V takovém případě se nabízí využití nějakého embedded databázového řešení – vzhledem k využití MS SQL Server pro centrální databázi pak celkem logicky přichází na mysl varianta MS SQL Server Compact Edition. Při podrobnějším prozkoumání možností tohoto řešení se bohužel ukázalo jako velmi obtížně použitelné z důvodů absence podpory uložených procedur a zejména databázových pohledů (bez kterých je prakticky nemožné rozumně realizovat porovnání sbírky s typickou protisbírkou). Kromě MS SQL Server Compact Edition výše zmíněný fakt vylučuje i další, jednodušší varianty, např. napojení na XML soubor. V úvahu by tak připadalo například

použití embedded řešení Firebird a SQLite. Po zralé úvaze a konzultaci s několika sběrateli jsem se nicméně rozhodl se této variantě vyhnout, s tím že hlavním přínosem aplikace je možnost porovnávání s typickými sbírkami v centrální databázi. Uživatelům je nicméně umožněno tvořit typické sbírky viditelné pouze pro ně.

3.5. Principy komunikace aplikace s datovým zdrojem

3.5.1. Možné varianty

Při tvorbě aplikace komunikující se vzdáleným datovým zdrojem je dle [5] možné využít dvou variant propojení datové a aplikační vrstvy. Prvním je varianta tzv. „připojené aplikace“, kdy tato má při manipulaci s daty ze zdroje vždy přímé připojení k tomuto zdroji (nejčastěji pomocí rozhraní `IDbConnection`, `IDbCommand`). Druhou, vývojově pozdější, je varianta tzv. „odpojené aplikace“, kdy je mezi datovou a aplikační vrstvu vložen mezičlánek, do kterého je jednak načten obraz dat z databáze a zároveň jsou do něj promítány uživatelské změny a rozdíly jsou pak v určitý moment promítnuty zpět do DB.

Dle [7] je výhodou „připojené aplikace” zejména fakt, že máme vždy k dispozici aktuální data, naopak nevýhodou je vyšší míra komunikace mezi DB a aplikací a tedy nezbytnost kvalitního a spolehlivého připojení k datovému zdroji. Zejména při vzdáleném připojení pomocí veřejných sítí může použití této varianty způsobovat problémy drobné výpadky a zpomalení práce.

U „odpojené aplikace” je výhodou nižší míra komunikace mezi DB a aplikační vrstvou (jedná se spíše o kratší úseky intenzivnějších přenosů). Případná zpomalení komunikace se také objevují na „očekávatelnějších” místech (např. načtení celé tabulky, ne procházení jednotlivých řádků). Použití „odpojené aplikace” není považováno za příliš výhodné nad daty, která jsou často souběžně měněna větším množstvím uživatelů, z důvodu rizika změnových konfliktů [4].

3.5.2. Použití „odpojené aplikace"

Vzhledem k charakteru práce jsem se rozhodl zvolit variantu „odpojené aplikace", kdy tato zůstává po většinu času od datového zdroje odpojena. Pro udržení obrazu dat v aplikaci jsou využity třídy `DataSet` a `DataAdapter`, obsažené ve jmenném prostoru `System.Data`. Konkrétně jsem zvolil méně obecnou variantu, tzv. strongly typed `DataSet` [4], při jehož použití jsou přímo vygenerovány rozšířené třídy a metody pro konkrétní sadu tabulek.

Výhodou takového postupu je jednak vyšší přehlednost výsledku, jednak fakt, že případné chyby v datových typech a předávání parametrů jsou odhaleny již při překladu. Svou roli hrálo také to, že strukturu databáze jsem vytvořil již během analýzy problematiky a tedy vznikala dříve, než samotná aplikace – přímé vytvoření strongly typed `DataSetu` pomocí nástroje Server Explorer tak znamenalo značnou časovou úsporu.

V případě této aplikace nehrozí bezprostřední riziko změnových konfliktů z důvodu absence přímo sdílených dat – např. ke sdílení jedné sbírky více uživateli při její tvorbě obecně nedochází.

3.6. Nástroje pro import, export a tisk

U katalogové aplikace nelze opomenout funkcionality pro tisk a export uložených dat, kdy jsou vhodně zformátovaná a utříděná data uložena do souboru nebo odeslána na tiskárnu za účelem další prezentace. Obecně se pro tuto část problematiky používá termín *reporting* a jednotlivé výstupní dokumenty jsou pak označovány jako *reporty*. Z důvodu výše zmíněné snahy o využívání tuzemské terminologie bude v následujícím textu používán termín *přehledy*.

3.6.1. Možné varianty tvorby přehledů

Přímočarým postupem k zajištění výše zmíněných funkcionalit je implementace vlastního řešení. Tato problematika je ovšem natolik rozsáhlá, že by tento postup byl pravděpodobně náročnější, než vytvoření samotné aplikace, pro kterou by byly zmíněné funkcionality určeny – pro dosažení použitelného

výsledku se jeví jako nezbytné důkladné proniknutí do struktury cílových formátů souborů, popř. využití velkého objemu cizího kódu v podobě tříd pro export. V podobném stylu by bylo potřeba zpracovat i výstup na tiskárnu. Tvorba přehledů je navíc velmi žádanou a často implementovanou funkcionalitou, čehož důsledkem je existence velkého množství volně i komerčně dostupných řešení. Některá taková řešení jsou přímo součástí .NET (třídy `ReportDocument` a `ReportClass`), další jsou součástí vývojového prostředí Microsoft Visual Studio 2005 (`CrystalReport`).

3.6.2. Třída CrystalReport

Hlavním důvodem pro volbu tohoto řešení je poměrně rozumné zpracování nástrojů pro vytváření nových přehledů, důležitou roli hraje i dobře vyřešený výstup na tiskárnu a export do souborů (ve srovnání se zmiňovanými třídami `ReportDocument` a `ReportClass`). Nevýhodou je, že se jedná o „odlehčenou" zdarma dostupnou variantu komerčního řešení – to v důsledku přináší některé nepříjemné překážky v používání (není například možné do jedné instance vložit více přehledových šablon).

3.6.3. Import

Funkce pro import je v případě aplikace významná zejména v případě vkládání již zpracovaného úplného seznamu mincí (typické sbírky), se kterou bude následně porovnávána vytvářená sbírka. V tomto případě jsem se rozhodl pro vlastní implementaci nekomplikovaného importu CSV souborů, jejichž struktura je popsána v příloze D.

3.7. Uživatelské účty a práva

Při tvorbě aplikace umožňující přístup více uživatelů pomocí oddělených účtů je dříve či později potřeba zabývat se správou uživatelských práv. Jednou z variant je zavedení systému konkrétních práv (např. k jednotlivým GUI komponentům, formulářům atp.) a následné vytváření uživatelských rolí jejich N : 1 svázáním s nadřazenou tabulkou. Jednodušší variantou je využití příznaků, kdy

jsou jednotlivá práva uvedena přímo v tabulce uživatelských účtů (např. práva typu `CAN_WRITE`, `IS_ADMIN` a podobně). Při tvorbě aplikace jsem se rozhodl pro druhou variantu, zejména z důvodu nižšího počtu potřebných práv a nepotřebnosti existence „více stupňů“ téhož práva – v takovém případě by byla zřejmě vhodnější tabulka oddělená tabulka rolí. Aplikace obsahuje tyto práva:

- Právo `USR_ISADMIN` – dává příslušnému uživateli administrátorská práva, uživatel může měnit práva a hesla jiných uživatelů pomocí příslušného formuláře
- Právo `USR_MAYADDTYPICAL` – dává příslušnému uživateli právo vytvořit ostatním uživatelům viditelnou typickou sbírku, bez tohoto práva smí uživatel tvořit pouze typické sbírky pro vlastní potřebu.

3.8. Validace uživatelských vstupů

Cílem validace uživatelských vstupů je jednak zajistit dodržení integritních omezení databáze při vkládání / editaci údajů, jednak snaha o maximální smysluplnost vkládaných dat (např. údaj ročník mince je v tomto případě z hlediska databáze celočíselnou hodnotou, z hlediska významu je v aplikaci navíc ošetřen počet číslic na čtyři platné).

Při validaci uživatelských vstupů je možné využít širokou paletu postupů, od porovnání zadaného řetězce s regulárním výrazem přes využití méně obecných ovladacích prvků (`MaskedTextBox`) až po kontrolu vstupu z klávesnice v rámci události `KeyPress` příslušné komponenty.

V rámci aplikace jsem se rozhodl využít kombinaci těchto metod, kdy je jednak programátorsky omezeno vkládání „nesmyslných“ znaků, jednak je využito validačních funkcí, ve kterých je vkládaný řetězec porovnán s vhodně sestaveným regulárním výrazem.

4. Uživatelská dokumentace

V této kapitole je stručně popsáno GUI aplikace a také vysvětlen princip jejího používání na několika příkladech. Jsou zde také zmíněny další komponenty nezbytné pro korektní nainstalování a spuštění programu.

4.1. Systémové požadavky

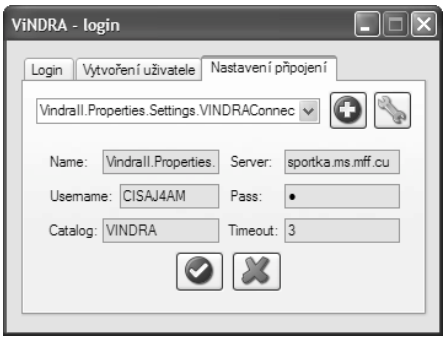
Pro korektní běh programu je nezbytná instalace prostředí .NET ve verzi nejméně 2.0 a také komponenty Microsoft Data Access 2.8 a Crystal Report for .NET Framework 2.0 (v případě absence některé z těchto komponent je tato nainstalována v rámci instalace programu). Doporučenými operačními systémy jsou Microsoft Windows ve verzích XP nebo Vista. Pro funkční připojení ke vzdálenému databázového serveru je nezbytné připojení k internetu a korektně nastavený firewall (nesmí být blokována komunikace na portu 1433).

4.2. Instalace a nastavení

Instalace aplikace se provede spuštěním souboru *setup.exe*, umístěného v adresáři *Instalace* na přiloženém CD. Pokud program nenalezne prostředí .NET ve verzi nejméně 2.0 nebo jiné výše zmíněné komponenty, provede nejprve jejich instalaci. Program je také možné přímo spustit z adresáře *Release* souborem *VindraII.exe* (bez testování instalace těchto komponent).

4.2.1. Konfigurace připojení

Úprava nastavení vzdáleného připojení je možná na kartě „Nastavení připojení“ v úvodním okně aplikace. Z vložených parametrů je vytvořen `ConnectionString` daného připojení, který je následně zobrazen v roletkách úvodního formuláře – údaje o vloženém připojení je tedy možné dále editovat, případně se ke zvolenému datovému zdroji zkusit přihlásit nebo na něm vytvořit nový uživatelský účet.



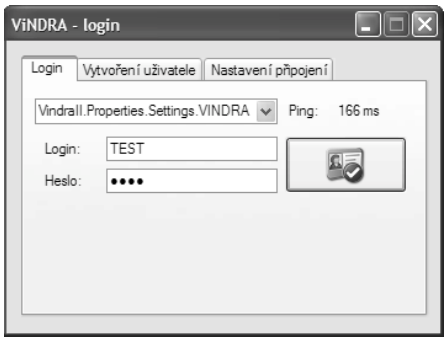
Obrázek 1: Nastavení připojení

Změna připojení je možná i ručně pomocí změny parametru `ConnectionString` souboru *.config v kořenovém adresáři aplikace. Pokud aplikace nenalezne konfigurační soubor, pokouší se o připojení k původnímu serveru (`sportka.ms.mff.cuni.cz`). Konfigurace připojení je podrobněji rozebrána v Příloze C.

4.3. Přihlášení / vytvoření uživatelského jména

Po spuštění aplikace je zobrazen úvodní formulář s kartami „Login“, „Vytvoření uživatele“ a „Nastavení připojení“.

Karta „Login“ obsahuje roletku pro zvolení datového připojení[4] a kolonky pro zadání přihlašovacího jména a hesla.



Obrázek 2: Přihlašovací okno aplikace

Pokud uživatel dosud s aplikací nepracoval, má možnost vytvořit nový účet zadáním příslušných údajů (jméno, příjmení, přihlašovací jméno, heslo) po

[4] Defaultně obsahuje hodnotu pro připojení k serveru sportka.ms.mff.cuni.cz

na kartě „Vytvořit nového uživatele“. Je-li toto vytvoření úspěšné, dojde k opětovnému zobrazení karty „Login“ a uživatel se může přihlásit pomocí nových údajů[5].

V případě úspěšného přihlášení je toto okno uzavřeno a dojde ke zobrazení hlavního okna programu, pokud došlo k zadání chybných údajů, je uživatel vyzván k opakování. Uzavření okna bez přihlášení ukončuje celou aplikaci. V databázi na serveru sportka.ms.mff.cuni.cz je pro účely testování a předvádění aplikace vytvořen uživatel s přístupovými údaji „test“ / „test“ a maximálními právy, kterému je přiřazeno několik vzorových sbírek různých typů.

4.4. Hlavní okno aplikace

Hlavní okno aplikace je rozděleno na tři části – „Seznam sbírek“, „Seznam předmětů ve sbírce“ a „Detailní karta“ (viz obrázek 3 na následující stránce).

4.4.1. Seznam sbírek

Seznam uložených sbírek je zobrazen v přehledovém okně v levé části hlavního okna aplikace. Typ ikony určuje, zda se jedná o mincovní, nebo jinou sbírku. Klik myší na jednotlivé sbírky vyvolá načtení jejich obsahu do detailních oken. Úpravy v seznamu sbírek je možné provádět pomocí tlačítek „Nová sbírka“, „Upravit sbírku“ a „Smazat sbírku“ ve vrchním menu. Některé základní atributy sbírky zadané při jejím vytvoření (např. typ) jsou nadále neměnné. Při pokusu o smazání je sbírka testována na neprázdnost – v případě, že obsahuje předměty, je nutné nejprve smazat je.

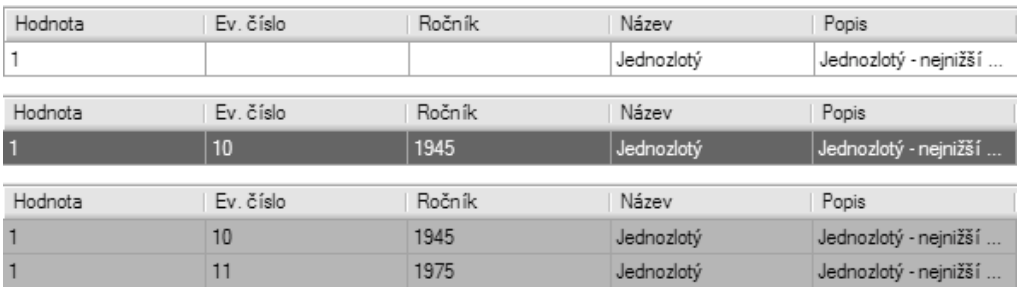
[5] Tuto metodu považuji za rozumnou pouze v testovací fázi, v případě zveřejnění aplikace je uživatelskou registraci nejspíše řešit jiným způsobem, např. pomocí webového formuláře, což je podrobněji zmíněno v kapitole 8.

Obrázek 3: Hlavní okno aplikace

4.4.2. Seznam předmětů ve sbírce

Seznam předmětů v aktuálně načtené sbírce je zobrazen formou tabulky v pravé dolní části hlavního okna. Zobrazené hodnoty a styl zobrazení se liší podle toho, jaký typ sbírky je zrovna procházen. Ve všech případech platí, že řádky reprezentující předměty ve sbírce jsou zobrazeny světle zelenou barvou. Pokud se ovšem jedná o sbírku s typickou protisbírkou, jsou navíc zobrazeny nezabarvené řádky reprezentující chybějící předměty. U chybějících předmětů zároveň nejsou vyplněna evidenční čísla a v případě sbírky bez ročníků ani ročníky.

Pro snažší pochopení je přiložen Obrázek 4, který zobrazuje seznam předmětů v typizované sbírce, jejíž protisbírka obsahuje jedinou minci (1 Zlotý). Na prvním řádku je zobrazena situace, kdy ve sbírce není žádná mince – pak je zobrazen řádek vzorové mince. Po vložení mince je tento řádek doplněn o vložené hodnoty (Ev. číslo a Ročník) a obarven. Pokud vložíme další minci (což je ze sběratelského hlediska v pořádku – sbíráme typ mince a můžeme mít více různých kusů), přibude další obarvený řádek s příslušnými hodnotami. Podrobněji je fungování datového zdroje pro tabulku vysvětleno v kapitole 6.3.1.



Hodnota	Ev. číslo	Ročník	Název	Popis
1			Jednozlotý	Jednozlotý - nejnižší ...

Hodnota	Ev. číslo	Ročník	Název	Popis
1	10	1945	Jednozlotý	Jednozlotý - nejnižší ...

Hodnota	Ev. číslo	Ročník	Název	Popis
1	10	1945	Jednozlotý	Jednozlotý - nejnižší ...
1	11	1975	Jednozlotý	Jednozlotý - nejnižší ...

Obrázek 4: Seznam předmětů

K seznamu předmětů navíc patří tlačítka „Export seznamu do CSV“ (viz příloha D) a „Vytvoření přehledu“ (viz 4.9. Přehledy).

4.4.3. Detailní karta

Detailní karta jednotlivých záznamů je zobrazena v pravé horní části hlavního okna. Typ karty a zobrazené údaje se mění podle toho, zda je aktuálně zvolena sbírka, nebo předmět ve sbírce. Karta předmětu je odlišena pro mincovní a nemincovní sbírky. V menu nad detailní kartou jsou umístěna tlačítka „Vložit předmět“, „Upravit předmět“, „Smazat předmět“ a „Vytvoření přehledu“. Pokud probíhá vkládání nebo úprava hodnot (ať sbírky, nebo předmětu), jsou v tomto menu zobrazena tlačítka „Uložit“ a „Zrušit“ a zároveň je zvýrazněna příslušná část detailní karty.

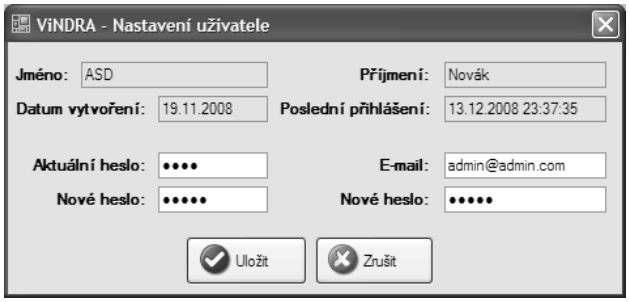
4.5. Formulář „Správa typických sbírek“

Formulář „Správa typických sbírek“ umožňuje uživateli vytvářet a upravovat vzorové sbírky, které je následně možné vyyužít jako typické protisbírky pro vlastní kolekce vytvářené v hlavním okně. Pokud není uživateli přiděleno příslušné právo, smí vytvářet jen soukromé sbírky (tj. viditelné pouze jemu).

Obrázek 5: Formulář „Správa sbírek“

4.6. Formulář „Nastavení uživatele“

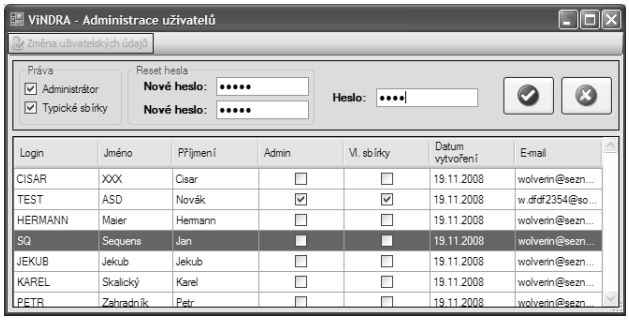
Formulář „Nastavení uživatele“ umožňuje zobrazit a měnit uživatelské údaje uložené v databázi k příslušnému loginu. Na tomto formuláři je také možné provést reset hesla.



Obrázek 6: Formulář „Nastavení uživatele“

4.7. Formulář „Administrace uživatelů“

Formulář „Administrace uživatelů“ je zobrazen pouze uživateli s právy administrátora. Umožňuje zobrazit údaje o všech uživatelích, měnit jejich práva a případně jim resetovat heslo.



Obrázek 7: Formulář „Administrace uživatelů“

4.8. Příklady práce s aplikací

Tato podkapitola uvádí několik typických postupů pro používání aplikace.

4.8.1. Založení nové sbírky

Vezměme za příklad vytvoření nové sbírky, zmíněné v kapitole 2.2, tedy sbírky typů mincí z období meziválečné ČSR. Kliknutím na tlačítko „Nová sbírka“ zahájíme editaci nového záznamu. Následně zadáme název sbírky a její číslo v naší sběratelské evidenci. Jedná se o mincovní sbírku, proto zaškrtneme přepínač „Sbírka mincí“. Pokud je sbírka obsažena v roletce „Předloha“, necháme přepínač „Podle předlohy“ v poloze „ano“ a budeme mít možnost sbírku porovnávat s její typickou protisbírkou. Zaškrtneme-li „ne“, zobrazí se parametr „Typ sbírky“, kde zvolíme, zda se jedná o sbírku typů, nebo typů a ročníků. Pokud jsme se zadanými údaji spokojeni, klikneme ve vrchním menu nad detailní kartou na nově zobrazené tlačítko „Uložit“. V opačném případě ukončíme vytváření sbírky tlačítkem „Zrušit“ a výsledek nebude uložen.

4.8.2. Vložení předmětu do sbírky

Pokud došlo k úspěšnému uložení sbírky do databáze, zobrazí se jako nový přírůstek v seznamu sbírek. Po kliknutí na ni se zobrazí detailní karta sbírky s právě zadanými údaji. Jelikož jsme při vytváření sbírky zvolili předlohu, jsou mince v této předloze zobrazeny v seznamu předmětů (žádný řádek není zvýrazněný, neboť sbírka ještě neobsahuje skutečnou minci). Pokud nyní klikneme na „Vložit předmět“, zobrazí se prázdná detailní karta nového předmětu s přednastaveným názvem sbírky, do které vkládáme. Opět zadáváme evidenční číslo předmětu v rámci naší sbírky, nominální hodnotu mince, rok ražby a kvalitu. Protože se jedná o předdefinovanou sbírku typů mincí, je výběr možných nominálních hodnot omezen pomocí roletky pouze na hodnoty, které jsou obsaženy v typické sbírce (v případě, že by se jednalo o sbírku typů a ročníků, bude podobně omezena i kolonka „Ročník“, naopak u sbírky bez předlohy

můžeme zadat libovolné numerické hodnoty). Máme-li k dispozici snímky mince, můžeme je do karty vložit pomocí tlačítek „Vložit líc“ resp. „Vložit rub“ v menu na detailní kartě. Podobně jako při tvorbě sbírky můžeme výsledek uložit tlačítkem „Uložit“, nebo přidání zrušit.

4.8.3. Úpravy sbírek / předmětů

Postup při úpravách je velmi podobný vkládání (s tím rozdílem, že je potřeba mít navolenou sbírku nebo předmět). Klikem na „Upravit sbírku“ / „Upravit údaje“ přepneme aplikaci do editačního módu (kdy není možné procházet jednotlivé seznamy) a je nám umožněno některé hodnoty v jednotlivých detailních kartách upravit. Postup při uložení / zrušení je pak zcela stejný jako v případě přidávání. Při úpravách nelze měnit všechny atributy (např. nominální hodnota mince je neměnná, naopak její evidenční číslo změnit lze).

4.8.4. Smazání sbírky

Mazání aktuálně zvolené sbírky je možné provést stiskem tlačítka „Smazat sbírku“ ve vrchním menu. Pokud je sbírka neprázdná, není možné ji přímo smazat a je nejprve nutné vymazat jednotlivé předměty v ní obsažené pomocí tlačítka „Smazat předmět“, umístěného v menu nad detailní kartou.

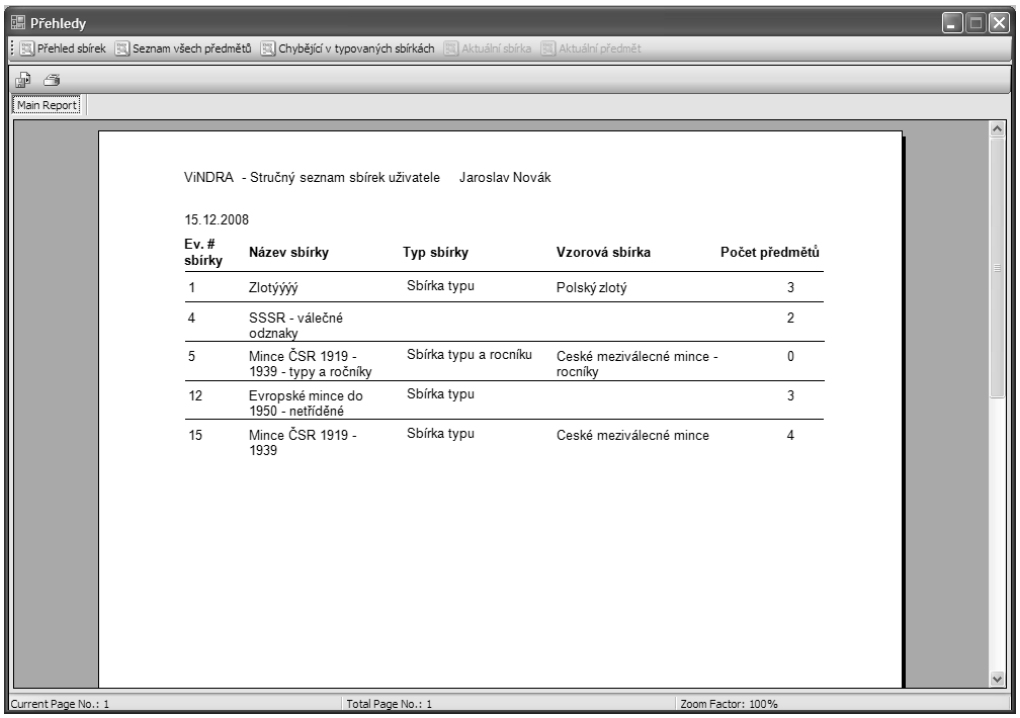
4.8.5. Vytvoření nové typické sbírky

Vytvoření nové typické sbírky se provádí na formuláři „Správa typických sbírek“. Po jeho spuštění jsou zobrazeny všechny aktuálnímu uživateli příslušné neaktivní sbírky. Novou sbírku vložíme do seznamu tlačítkem „Nová sbírka“, po zadání identifikačních údajů, typu a přístupnosti sbírky (lze umožnit zobrazení sbírky jiným uživatelům, je ovšem třeba mít příslušné právo) a jejím uložení můžeme následně jednotlivě vkládat jí přislušné mince tlačítkem „Vložit minci“, popřípadě využít možnosti hromadného importu seznamu ze souboru typu *.CSV (viz Příloha D). Pokud jsme s vytvořenou typickou sbírkou spokojeni, můžeme ji učinit viditelnou pro hlavní formulář stiskem tlačítka „Aktivace

sbírky“. Po aktivaci již není možné typickou sbírku upravovat (mohlo by dojít k nekonzistenci databáze, pokud by již byla vzorem pro skutečnou sbírku).

4.9. Přehledy – export a tisk údajů

Pro export a tisk záznamů zadaných v DB je vytvořen samostatný formulář „Přehledy“, který je možné vyvolat stejnojmenným tlačítkem v menu nad seznamem sbírek. Stisk tohoto tlačítka vyvolá nové okno s menu pro přepínání jednotlivých připravených přehledů. Samotné okno prohlížeče přehledů má své vlastní menu s tlačítky „Export“ a „Print“ , pomocí kterých je možné zobrazený přehled exportovat do souboru (typu *.pdf, *.xls a podobně) nebo s pomocí tiskového dialogu odeslat k tisku na tiskárnu. Po ukončení práce s přehledy je možné vrátit se k hlavnímu oknu aplikace uzavřením přehledového okna.



Obrázek 8: Formulář „Přehledy“

4.9.1. Seznam přehledů

Pomocí zmiňovaného menu ve formuláři „Přehledy“ je možné zobrazit jeden z pěti obsažených přehledů:

- *Přehled sbírek* – vytiskne seznam všech sbírek spolu s počtem předmětů v každé z nich
- *Seznam všech předmětů* – vytiskne seznam všech předmětů ve všech sbírkách
- *Chybějící předměty v typických sbírkách* – pro všechny sbírky s předlohou vytiskne seznam chybějících předmětů
- *Aktuální sbírka* – vytiskne seznam předmětů v aktuálně vybrané sbírce v hlavním okně aplikace (není-li vybrána žádná sbírka, nelze tento přehled zobrazit). Vzhled přehledu je pro mincovní a nemincovní sbírky odlišný.
- *Aktuální předmět* – vytiskne detailní kartu předmětu aktuálně vybraného v tabulce hlavního formuláře (není-li vybrán žádný předmět, nelze tento přehled zobrazit). I zde je graficky odlišen vzhled karty pro mince a jiné předměty.

Přehledy „Aktuální sbírka“ a „Aktuální předmět“ je možné vyvolat přímo z menu nad tabulkou v hlavním okně, resp. v menu nad detailní kartou předmětu.

Několik souborů vytvořených pro ukázku pomocí formuláře „Přehledy“ je uloženo ve stejnojmenném adresáři na přiloženém CD. Popisem vybraných datových zdrojů se zabývá Příloha B.

5. Programátorská dokumentace

Kapitola číslo 5 shrnuje hlavní postupy, které byly použity při návrhu / tvorbě aplikace. Některá méně standardní nebo z mého pohledu zajímavá řešení uvádím včetně krátké ukázky kódu. Pro lepší pochopení činností které aplikace vykonává je vhodné nejprve se seznámit s předchozími kapitolami, zejména 3 a 4.

5.1. Vrstva komunikace s DB

Jak bylo uvedeno výše v kapitole 3, komunikace s DB probíhá na principu „odpojené aplikace“ a pro tento účel je v projektu obsažen strongly typed `DataSet`. Jeho struktura obsahuje tabulky se strukturou analogickou struktuře tabulek databáze, doplněné navíc o tabulky pro ukládání výstupů jednotlivých databázových pohledů. Každá tabulka má vlastní třídu TableAdapter, která obsahuje metody typu `SELECT`, `UPDATE`, `INSERT` a `DELETE` pro synchronizaci této tabulky s datovým zdrojem. Pro aktualizaci obrazu dat v DataSetu jsou využívány metody `Fill()` (neparametrizovaná) a `FillBy()` (parametrizovaná, tabulka jich může mít více) vázané buď na příkazy typu SELECT v těle DataSetu nebo na procedury uložené v databázi.

5.1.1. Třída Connection

Ve třídě `Connection` jsou instanciovány TableAdaptery pro jednotlivé tabulky DataSetu – veškerá práce s daty pak probíhá voláním metod těchto TableAdapterů. V rámci konstruktoru třídy jsou nejprve synchronizovány číselníky a tabulka `COLLECTIONS`, jejíž prvky je třeba zobrazit již při spuštění hlavního okna. Konstruktoru je předáván celočíselný parametr `UserID`, který zajišťuje načítání pouze přihlášenému uživateli příslušných sbírek.

5.1.2. Třída Login

Ve třídě `Login` je definován další (netypovaný) `DataSet` a `DataAdapter` pouze pro načtení hodnot `USR_ID`, `USR_LOGIN` a

USR_PASS a příznaků pro práva z tabulky USERS. Jako parametr metody FillBy() je adaptéru předán uživatelem zadaný login (pokud je následně DataSet prázdný, pak login v databázi neexistue). Mimo to je v této třídě definován samostatný příkaz typu SqlCommand, jehož metodou ExecuteNonQuery() je provedeno volání DB procedury NewUser a vložení nového uživatele (viz kapitola 6). V konstruktoru třídy login je spuštěno nové vlákno s cílem je detekovat překročení času určeného pro připojení k datovému zdroji.

```
Stopwatch sw = new Stopwatch();
bool connectSuccess = false;

sqlConnection = new SqlConnection(ConnectionString);

Thread t = new Thread(delegate()
{
      try
      {
         sw.Start();
         sqlConnection.Open();
         connectSuccess = true;
      }
      catch { MessageBox.Show ... }
});

t.IsBackground = true;
t.Start();

while (timeout > sw.ElapsedMilliseconds)
      if (t.Join(1)) break;

      if (!connectSuccess) MessageBox.Show ...
```

5.1.3. Ošetření výjímek

Vzhledem k faktu, že úspěšné volání většiny metod ve výše zmíněných třídách je závislé na stabilitě vzdáleného připojení, případně korektnosti používaných parametrů, jsou tato „riziková“ volání umístěna do „bezpečných“ bloků try {}. Vznik případných výjímek je ošetřen v blocích catch {} zobrazením chybového hlášení.

5.2. Aplikační vrstva a GUI

Tato kapitola popisuje jednotlivé formuláře a také statické třídy `CSVImport` a `Validation`.

5.2.1. Formulář MainForm

Formulář `MainForm` obsahuje všechny hlavní uživatelské prvky, konkrétně se jedná o instance tříd `ListView` (seznam sbírek), `DataGridView` (seznam předmětů), `TabControl` (detailní karty) a `ToolStripMenu`. Uživatelské vstupy jsou vkládány pomocí instancí standardních GUI komponent `TextBox`, `ComboBox` a podobně. Datovým zdrojem je instance třídy `Connection`.

5.2.2. Formulář LoginForm

Instance formuláře `LoginForm` je inicializována v rámci metody `MainForm_Load()` formuláře MainForm. Pomocí instance třídy `Login` načte data odpovídající zadanému uživatelskému jménu, provede zahashování zadaného hesla algoritmem SHA1 a porovná výsledek.

```
byte[] rawstring =
System.Text.Encoding.Unicode.GetBytes(textBox2.Text);

byte[] hashstring = sha.ComputeHash(rawstring);

byte[] query =
(Byte[])(log.dsLogin.Tables[0].Rows[0]["USR_PASS"]);

string hashed1 = Convert.ToBase64String(hashstring);
string hashed2 = Convert.ToBase64String(query);

if (hashed1 == hashed2) …
```

Třída `LoginForm` obsahuje referenci na `MainForm` a pomocí veřejné metody `setUserID()` předává v případě úspěšného přihlášení uživatele jeho ID hlavnímu formuláři.

5.2.3. Formulář ReportForm

Instance formuláře `ReportForm` je inicializována v rámci metody `toolStripButtonReport_Click()` volané tlačítkem z formuláře MainForm. Formulář je následně zobrazen voláním metody `ShowDialog()`. Hlavní součástí formuláře je instance třídy `CrystalReportViewer`, sloužící ke zobrazení přehledů (viz 5.3.).

5.2.4. Formulář AdminForm

Instance formuláře `AdminForm` je inicializována při stisku tlačítka „Administrace uživatelů“ v hlavním okně aplikace. Formulář obsahuje vlastní `DataGridView` napojený na seznam uživatelů a další GUI komponenty. Zároveň obsahuje metody pro aktualizaci údajů a porovnání zahashovaného hesla.

5.2.5. Formulář ChangeForm

Instance formuláře `ChangeForm` je inicializována při stisku tlačítka „Nastavení uživatele“ v hlavním okně aplikace. Obsahuje metody pro aktualizaci údajů a porovnání zahashovaného hesla, analogické metodám formuláře `LoginForm`.

5.2.6. Formulář NewTypical

Instance formuláře `NewTypical` je inicializována při stisku tlačítka „Správa typických sbírek“ v hlavním okně aplikace. Formulář obsahuje vlastní `DataGridView` se seznamem mincí a další GUI komponenty. Zároveň obsahuje metody pro import / export CSV s pomocí `FileDialogu` a statické třídy `CSVImport` (viz 5.2.13.).

5.2.7. Práce s obrázky

Před uložením do `DataSetu` je třeba převést obrázek z `PictureView.Image` na pole typu `Byte[]` (datový typ `Image` v MS

SQL). K převodu je použita třída `MemoryStream`, pole je následně uloženo do příslušné tabulky ds1 (`ds1.COINS`). Podrobněji je tento postup popsán v [4].

```
MemoryStream ms1 = new MemoryStream();

pictureBox1.Image.Save(ms1, ImageFormat.Jpeg);

Byte[] bytBLOBdata1 = new Byte[ms1.Length];
ms1.Position = 0;
ms1.Read(bytBLOBdata1, 0, Convert.ToInt32(ms1.Length));

con.ds1.COINS.Rows[dataGridView1.CurrentRow.Index]["CNS_IMAGE_F
RONT"] = bytBLOBdata1;
```

V rámci šetření prostoru v DB (a v neposlední řadě zachování rozumné rychlosti procházení sbírek) je omezena velikost vkládaného obrázku, který je převzorkován na velikost příslušného `PictureBoxu`.

```
private Bitmap ResizeBitmap(Bitmap b, int nWidth, int nHeight)

{
            Bitmap result = new Bitmap(nWidth, nHeight);
            using (Graphics g =
Graphics.FromImage((Image)result))
                g.DrawImage(b, 0, 0, nWidth, nHeight);
            return result;
}
```

5.2.8. Zobrazení dat pomocí DataGridView

K procházení jednotlivých předmětů zařazených v kolekci je využita instance třídy `DataGridView`. Její událost `SelectionChanged()` slouží k aktualizaci prvků na detailní kartě. Vzhledem k tomu, že je potřeba, aby `DataGridView` zobrazoval data z různých tabulek `ds1` (konkrétně `ds1.COINS` pro netypizované a `ds1.JOINEDCOINS` pro typizované sbírky), jsou hlavičky jednotlivých sloupců pojmenovávány a zobrazovány „ručně“ pomocí funkce `setGridMode()`.

5.2.9. Validace uživatelských vstupů a třída Validation

Před zápisem řádku do příslušné tabulky v `DataSetu` je provedena kontrola uživatelských vstupů na aplikační úrovni. Nepovolené zadání nečíselných hodnot do uživatelských ovladacích prvků typu `TextBox` je omezeno pomocí kontroly vstupu regulárním výrazem.

```
private void txtNumber_KeyDown(object sender, KeyPressEventArgs
e)

{

if
(!System.Text.RegularExpressions.Regex.IsMatch(e.KeyChar.ToStri
ng(), "\\d+"))

e.Handled = true;

}
```

Metody pro validaci uživatelských vstupů před pokusem o jejich uložení do `DataSetu` obsahuje statická třída `Validation`. Najdeme zde například metody `IsValidValue()` (je-li text platnou hodnotou mince) resp. `IsValidYear()` (je-li text platným označením ročníku). Většina těchto funkcí pracuje s porovnáním regulárních výrazů. Mimo zmíněných funkcí obsahuje tato třída také metody pro zobrazení typických chybových hlášení.

Tato kontrola je obecně „přísnější“, než ošetření pomocí constraint na DB úrovni, kde jsou řešeny pouze situace, které mohou způsobit nekonzistenci databáze.

5.2.10. Vkládání do DataSetu

Při uživatelském vkládání sbírky nebo mince (a tedy řádky v příslušném `DataSetu`) je vždy zvlášť vytvořena nová instance třídy `DataRow` reprezentující jednu řádku dat, která odpovídá konfigurací sloupců dotčené tabulce `DataSetu`. Vložení uživatelem zadaných údajů je nejprve provedeno pro tuto řádku, která je následně vložena do příslušné tabulky v rámci `DataSetu` jako celek voláním metody `AddRow()`.

5.2.11. Update DataSetu

Při update `DataSetu` je vytvořena `DataRow` reference na příslušný řádek (vyhledaný podle primárního klíče). Po vložení / úpravě dat je proveden aktualizace databáze voláním metody update příslušného `DataAdapteru`.

5.2.12. Třída CSVImport

Statická třída `CSVImport` obsahuje metody pro export a import dat do / z CSV souborů. Obě metody pracují s tabulkou `DataTable`, pro import je využito prosté načtení souboru do pole `String[]`, pro export instance třídy `StreamWriter`.

5.3. Přehledy

Pro tisk a export seznamů z DB je využita třída `CrystalReport`, která je rozšířením třídy `ReportClass` a jako taková je součástí Microsoft Visual Studia. Jednotlivé instance přistupují prostřednictvím referencované třídy k `DataSetu ds1` jako ke svému datovému zdroji. Pro každou instanci třídy `CrystalReport` je rovněž zvlášť zpracován pohled na databázové úrovni (podrobněji rozebráno v kapitole 6 a také v příloze B). Zobrazení provádí instance třídy `CrystalReportViewer` umístěná na formuláři `ReportingForms`.

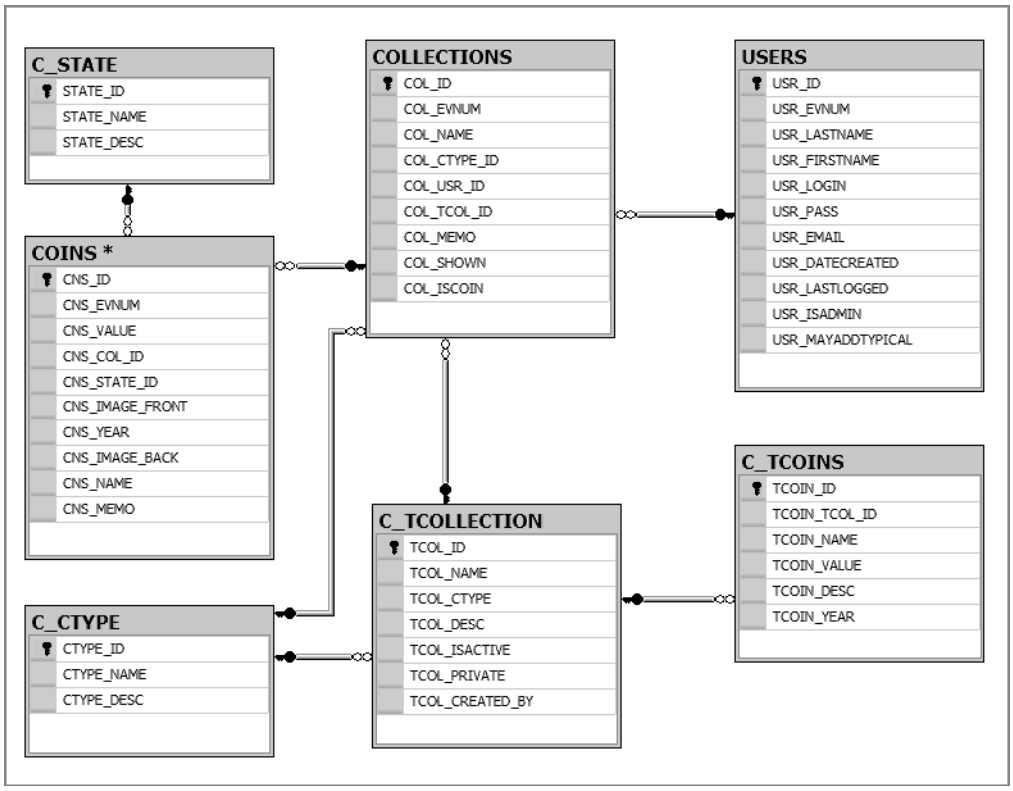
Využití třídy CrystalReport přináší problémy při použití na 64bitových systémech, čemuž lze předejít vhodným nastavením kompilátoru (x86).

6. Dokumentace databáze

Shrnutí řešení na úrovni DB jsem se rozhodl vzhledem k většímu rozsahu věnovat samostatnou kapitolu. Část poznatků, podle kterých bylo při tvorbě DB návrhu postupováno, je zmíněna již v rámci analýzy problematiky v kapitole 2. Část příkazů typu `SELECT` (zejména u přehledů) pro načtení dat do `DataSetu ds1` a následně do uživatelských prvků je prováděna pomocí v DB uložených procedur – toto řešení považuji za přehlednější (je omezeno psaní SQL skriptů přímo v třídě `DataSetu`), výhodou je i to, že je možné provádět drobné úpravy na DB úrovni bez nutnosti zasahovat do definice `DataSetu`. Při tvorbě návrhu bylo využito poznatků uvedených v [8].

6.1. Základní návrh a schéma

Datový model aplikace je tvořen třemi hlavními a čtyřmi číselníkovými tabulkami, nad kterými jsou dále vytvořeny pohledy, procedury, indexy atd. Návrh počítá s možnými rozšířeními v podobě dalších číselníků, v případě přidání většího množství atributů k jednotlivým předmětům i s rozštěpením některé z hlavních tabulek. Tabulky jsou vzájemně propojeny pomocí cizích klíčů, což podrobně vyobrazuje schéma na následující stránce.



Obrázek 9: Schéma databázového modelu

6.1.1. Tabulka COLLECTIONS

Smyslem tabulky `COLLECTIONS` je uchování dat o jednotlivých sbírkách, jejich typech, vlastnících a podobně. Je ve vztahu N : 1 k tabulce `USERS`, na kterou odkazuje prostřednictvím sloupce `COL_USR_ID`. Další propojení jsou s číselníky `C_CTYPE`, `C_TCOLLECTION` a `C_TCOINS` (viz níže).

6.1.2. Tabulka COINS

Tabulka `COINS` uchovává informace o jednotlivých předmětech obsažených ve sbírkách (nemusí se nutně jednat pouze o mince). Její vztah k tabulce `COLLECTIONS` je N : 1, odkazujícím sloupcem je `CNS_COL_ID`. Je propojena také s číselníkem `C_STATE`. V případě přidání dalších atributů považuji za rozumné tuto tabulku rozštěpit (např. na tabulku pro mince a tabulku jiných předmětů).

6.1.3. Tabulka USERS

V tabulce USERS jsou uchovávána data o jednotlivých uživatelích aplikace/vlastnících příslušných sbírek. Její součástí jsou i přístupové údaje USR_LOGIN a USR_PASS. Pro vložení uživatele včetně zahashování hesla je v DB vytvořena procedura NewUser (viz níže). Zároveň obsahuje příznaky USR_ISADMIN a USR_MAYADDTYPICAL označující uživatelská práva.

6.2. Číselníky

V případě, že je množina možných hodnot daná (např. již zmíněná stupnice kvality mincí), je využito provázání pomocí cizího klíče na číselník. Číselníkové tabulky mají na začátku svého názvu C_ (např. C_STATE pro již zmiňovanou stupnici kvality).

Zvláštním případem číselníků jsou tabulky C_TCOLLECTION a C_TCOINS (jejichž vztah je ekvivalentní vztahu hlavních tabulek COLLECTIONS a COINS), které obsahují údaje o typických sbírkách a v nich obsažených mincích. S těmito sbírkami je pak možné porovnávat vytvářené uživatelské sbírky pomocí příslušných DB pohledů.

6.2.1. Tabulka C_TCOLLECTION

Smyslem tabulky C_TCOLLECTION je uchování dat o jednotlivých typických sbírkách. Je ve vztahu N : 1 k tabulce USERS, kam prostřednictvím sloupce TCOL_USR_ID odkazuje na ID svého tvůrce/vlastníka. Mimo dalších sloupců ekvivalentních sloupcům v tabulce COLLECTIONS obsahuje i dva příznaky TCOL_ISACTIVE a TCOL_PRIVATE. První označuje, zda byla tato typická sbírka aktivována a je tedy možné podle ní vytvářet skutečné sbírky (zároveň už není možná její editace – jiný postup by mohl vést k nekonzistentnosti databáze), druhá označuje soukromou sbírku (tedy takovou, která je po aktivace zobrazena pouze svému vlastníkovi).

6.2.2. Tabulka C_TCOINS

Tabulka obsahuje vzorové mince, zařazené do jednotlivých typických sbírek. Obsahuje sloupce s vlastními hodnotami `TCOIN_VALUE`, `TCOIN_NAME`, `TCOIN_YEAR` a `TCOIN_DESC` a odkaz na nadřazenou typickou sbírku `TCOIN_TCOL_ID`.

6.3. Pohledy

Složitější dotazy kombinující data z více tabulek jsou realizovány pomocí sady pohledů. Jedná se typicky o dotazy, kde je porovnávána sbírka a typizcká protisbírka, nebo o dotazy, jejichž výstupy jsou využity pro tisk přehledů (tyto pohledy jsou uvedeny v Příloze B). Jako příklad prvního případu může posloužit view `MISSING` pro vyhledání chybějících typů mincí porovnáním uživatelské sbírky a typické protisbírky:

```
CREATE VIEW MISSING AS

SELECT COL_ID, COL_NAME, TCOIN_VALUE AS VALUE
FROM COLLECTIONS
LEFT OUTER JOIN
C_TCOLLECTION ON COL_TCOL_ID = TCOL_ID
LEFT OUTER JOIN C_TCOINS ON TCOIN_TCOL_ID = TCOL_ID

EXCEPT

SELECT COL_ID, COL_NAME,CNS_VALUE AS VALUE
FROM COLLECTIONS
LEFT OUTER JOIN COINS ON CNS_COL_ID = COL_ID;
```

6.3.1. Pohled JOINEDCOINS

Pohled `JOINEDCOINS` je využíván pro současné vyhledání mincí příslušných dané sbírce a zároveň mincí příslušných její typické protisbírce. Jeho výstup je využíván v hlavním gridu aplikace pro barevné odlišení mincí v typické protisbírce a mincí vlastněných.

```
CREATE VIEW

JOINEDCOINS AS

SELECT TCOIN_ID, TCOIN_VALUE, TCOIN_TCOL_ID,
ISNULL(TCOIN_YEAR,CNS_YEAR) AS TCOIN_YEAR, TCOIN_DESC,
TCOIN_NAME, COINS.*, COL_USR_ID, ISNULL(COL_ID,0) AS COL_ID
```

```
FROM

C_TCOINS INNER JOIN
COLLECTIONS ON COL_TCOL_ID = TCOIN_TCOL_ID LEFT OUTER JOIN
COINS ON

CNS_VALUE = TCOIN_VALUE AND CNS_COL_ID = COL_ID AND ((CNS_YEAR
= TCOIN_YEAR) OR (TCOIN_YEAR IS NULL))
```

Pro různé druhy typizovaných sbírek dává tento pohled mírně odlišný výstup:

- Jedná-li se o sbírku typů mincí (a tedy u typické protisbírky není vyplněn ročník), pak je pro vlastněné mince navíc vyhledáno `CNS_ID` (tedy jejich ID v tabulce mincí), `CNS_EVNUM` a `CNS_YEAR`.
- Jedná-li se o sbítku typů a ročníků, pak musí být u vlastněné mince hodnota `CNS_YEAR` rovna hodnotě `TCOIN_YEAR` a řádky se tedy liší jen ve dvou sloupcích – `CNS_ID` a `CNS_EVNUM`.

6.4. Procedury

V databázi je vytvořena kolekce procedur, kromě už zmiňovaných příkazů typu `SELECT` pro jednotlivé `DataAdaptery` je jejich účelem zejména zjednoduššení předpokládaných častých operací vkládání dat. Jako příklad může posloužit procedura `NewUser` pro vytvoření nového uživatele v tabulce `USERS`:

```
@lastname NVARCHAR(30),
@firstname NVARCHAR(30),
@read BIT = 1,
@login NVARCHAR(20) = NULL,
@password NVARCHAR(20) = NULL,
@write BIT = 1,
@evnum INT,
@email NVARCHAR(50)

AS
BEGIN

BEGIN TRANSACTION;

IF @read = 1 -- uživatel se mùže pøihlásit do systému
BEGIN
```

```
  IF @login IS NULL OR @password IS NULL -- ale chybí mu login
nebo heslo
    BEGIN
      RAISERROR('Uzivatel s pristupem do systemu musi mit login
a heslo.', 16, 1);
      ROLLBACK TRANSACTION;
      RETURN -1;
    END
END
ELSE -- uživatel se nemùže pøihlásit do systému
BEGIN
  IF @write = 1
    SET @write = 0; -- takže do nìj nemùže ani zapisovat
  IF @login IS NOT NULL OR @password IS NOT NULL
    BEGIN
      RAISERROR('Vyplnen login nebo heslo, ale nepovolen
pristup do systemu.', 16, 1);
      ROLLBACK TRANSACTION;
      RETURN -1;
    END
END

-- zahashovat heslo sha1
DECLARE @encpass VARBINARY(160);
DECLARE @datecreated datetime;
SET @encpass = HASHBYTES('SHA1', @password);
SET @datecreated = Getutcdate();

-- vložení do DB
INSERT INTO USERS(USR_EVNUM, USR_LASTNAME, USR_FIRSTNAME,
USR_LOGIN, USR_PASS, USR_EMAIL, USR_DATECREATED)
VALUES(@evnum, @lastname, @firstname, @login, @encpass, @email,
@datecreated);

COMMIT TRANSACTION;

RETURN;

END
```

Vidíme, že do sloupce `USR_PASS` je vloženo pouze pomocí algoritmu SHA1 zahashované heslo.

6.5. Indexy

V rámci urychlení operací nad databází byly vytvořeny indexy nad sloupci, podle jejichž hodnot je v pohledech a uložených procedurách nejčastěji vyhledáváno. Např. index pro vyhledávání mincí příslušejících k jedné sbírce:

```
CREATE INDEX I_CNSCOL_ID ON COINS (CNS_COL_ID)
```

Podobně pro sbírky jednoho uživatele:

```
CREATE INDEX I_COLUSRID ON COLLECTIONS (COL_USR_ID)
```

7. Přehled již existujících řešení podobné problematiky

První tři programy zmíněné v této kapitole jsou vesměs staršího data a v limitovaných verzích jsou dostupné zdarma. Použití plné verze je u nich zpoplatněno částkou od 10 do 25 USD. Na závěr této kapitoly je zmíněn stále vyvíjený program CoinManage, specializovaný na mince USA a Kanady. Jeho kvalita stojí o poznání výše, je to ale vykoupeno také podstatně vyšší pořizovací cenou. Na poli evropských mincí je situace obecně výrazně horší – existuje pouze několik již historických řešení ze sousední SRN (vesměs v němčině a zaměřených pouze na pevně dané seznamy německých mincí).

7.1. Coin Collection Wizard

Program Coin Collection Wizard pochází z dílny australské společnosti Information Packacing, která na svém webu nabízí poměrně širokou paletu různých drobných utilit (některé jako freeware, jiné za mírný poplatek – to je i případ Coin Collection Wizard). Aplikace nabízí možnost ukládat k jednotlivým mincím velké množství údajů (např. nákupní cenu a podobně), bohužel ale neumožňuje další třídění do sbírek a nenabízí ani žádné předpřipravené číselníky – je nutné si tyto uživatelsky samostatně nadefinovat. Ukládání probíhá do strukturovaného textového souboru, je možný export tabulkového seznamu do jednoduchého *.html dokumentu bez vizuálních stylů. U programu je uveden rok vzniku 2005, ale grafické řešení napovídá ještě o něco starší datum (což potvrzují i data souborů po instalaci, kdy některé pochází z roku 2003). Krom už zmíněného považuji za nevýhodu i značně nepřehledné grafické zpracování.

7.2. WorldCoins

Aplikace World Coins byla vyvinuta britskou firmou ArteCode Software, která se dle vlastních slov specializuje na vývoj software pro katalogizaci sbírek (kromě WorldCoins nabízí i aplikaci WorldStamps, zpracovanou podobným způsobem). Plná verze programu stojí 22.5 USD, volně dostupná trial varianta obsahuje několik omezení (nemožnost importovat předvytvořené textové seznamy, správa max. 40 mincí). Aplikace je zpracována v prostředí Visual Basic a jako lokální databázi využívá soubor *.mdb typu MS Access. Výhodou je, že aplikace obsahuje několik poměrně dobře zpracovaných číselníků (stav mince, kov a podobně) a dává uživateli možnost je snadno upravovat a rozšiřovat. Podobně jako v předchozím případě není možné vytvořit vlastní sbírku ve smyslu přidávání předmětů – program vkládá všechny sobě známé mince do jednoho seznamu, přičemž je možné u každé mince vyznačit situaci (mám, sháním...). Kromě zjevné nevýhody nemožnosti vzájemného oddělení sbírek je tato metoda značně nepřehledná, zejména pokud je obsaženo větší množství údajů.

7.3. Coin Tracker

Program Coin Tracker vytvořila společnost CyberNiche Software (opět u ní najdeme celou řadu jednoduchých utilit pro různé účely). Jedná se již o poněkud historickou záležitost cílenou na operační systém Windows 95, ale přes tento hendikep obsahuje několik zajímavých funkcí – mimo jiné historii každé mince z pohledu vývoje její tržní ceny, nákupů a prodejů. Při exportu údajů je možné v omezené míře ovlivnit seznam exportovaných parametrů. Uživatelský komfort je výrazně nižší, než je v současnosti běžné, což je potřeba přičíst době vzniku.

7.4. CoinManage 2008

Tvůrcem aplikace CoinManage je kanadská společnost Liberty Street, specializovaná na vývoj sběratelských a inventářních katalogů. Pořízení plné verze vyjde na 60 USD, je ale možné program vyzkoušet pomocí testovací verze, kterou

je možné používat po dobu jednoho měsíce. Jednoznačnou výhodou programu je velmi rozsáhlá a podrobná databáze (bohužel pouze mincí s původem v USA, Kanadě a Velké Británii), která obsahuje mimo jiné i počty vyražených mincí a vývoj jejich historické ceny na sběratelském trhu. Za zmínku stojí i možnost pokročilého vyhledávání v databázi pomocí palety různých filtrů a podobně. Možnosti tvorby přehledů jsou velmi široké, je možné podrobně definovat vlastní přehledy a volit jejich vzhled v závislosti na cílovém formátu (samostatné přehledy pro typ MS Excel atd.). Poměrně zajímavá je i možnost přímo z programu vyhledávat k jednotlivým mincím probíhající aukce na serveru eBay. Uživatelské recenze velmi často zmiňují problémovou instalaci aplikace, na tento problém jsem ale nenarazil. Další častá výtka na nízkou přehlednost je věcí vkusu a je daní za velké množství funkcí.

8. Závěr

Cílem této práce bylo vytvořit katalogovou utilitu zaměřenou na správu sběratelských kolekcí. Problematika tvorby, rozšiřování a doplňování sbírek byla před započetím práce jednak samostatně analyzována, jednak konzultována s několika potencionálními uživateli a výsledné řešení tak respektuje ustálenou metodiku a postupy při těchto činnostech obvyklé.

Zvolené téma se samo o sobě nezdá být velkou překážkou z pohledu implementeční náročnosti nebo nutnosti „vymýšlet a zapisovat algoritmy“, na druhou stranu představuje práci na několika vzájemně sice souvisejících, ale fakticky oddělených částech, s nutností dbát na ošetření uživatelských vstupů.

Domnívám se, že se podařilo implementovat funkční základ řešení s rozumným uživatelským komfortem, které má pro sběratele svou hodnotu zejména díky funkcím pro porovnávání typických a vytvářených sbírek a funkcím pro tisk a export přehledů. Samotná míra použitelnosti a užitečnosti aplikace je dle mého názoru do značné míry úměrná počtu a rozsahu importovaných typických sbírek (zejména českých a středoevropských mincí).

Použití „odpojené aplikace“, funkcionality pro tvorbu přehledů a tvorba databázových pohledů určených pro tyto přehledy je v současnosti žádanou vývojářskou dovedností a považuji za přínos to, že jsem si tyto činnosti mohl v rámci tohoto projektu vyzkoušet.

Několik možností dalšího rozšíření aplikace uvádím v následující podkapitole.

8.1. Další plánovaná rozšíření

Vzhledem ke způsobu zpracování a využití vzdáleného uložení dat se přímo nabízí zahrnout do aplikace v rámci dalšího rozvoje vyšší úroveň uživatelské kooperace (založenou na možnosti nějakým způsobem prohlížet cizí sbírky nebo vypisovat možné transakce mezi uživateli), doplněnou např. o prezentaci jednotlivých sbírek na webu s použitím technologie .ASP. Na

databázové úrovni je s takovýmto rozšířením částečně počítáno (v tabulkách jsou zobrazeny příznaky pro veřejné zobrazení apod., jsou přípraveny pohledy pro vypsání duplicit a možných výměn), pro zajištění uživatelského komfortu je ale nezbytné ošetřit možné konflikty a také zapracovat rozdělení uživatelských práv a rolí.

Program byl také dán k vyzkoušení několika spolupracovníkům magazínu „Mince a bankovky“ s příslibem zapracování jejich odborných připomínek (pravděpodobným důsledkem je zvýšení počtu atributů u jednotlivých předmětů, přidání několika dalších číselníků a podobně).

Před případným zveřejněním práce považuji za nezbytné zajistit vhodný MS SQL databázový server.

9. Seznam použité literatury

1. Petr Vorel: *Od pražského groše ke koruně české*, Havran, 2005, ISBN: 80-86515-40-0
2. *Aurea Numismatika*, http://www.aurea.cz
3. *Numismatic Guaranty Corporation*, http://www.ngccoin.com
4. *Microsoft Developer Network*, http://msdn.microsoft.com
5. S. Robinson a kol.: *C# Programujeme profesionálně*, Computer Press, 2003, ISBN: 80-251-0085-5
6. Troels Arvin: *Comparison of different SQL implementations*, http://troels.arvin.dk
7. Petr Puš: *Poznáváme C# a Microsoft .NET*, http://www.zive.cz
8. Forrest Houlette: *SQL Příručka programátora*, Softpress, 2001, ISBN: 80-86497-14-3

Příloha A – Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře:

- CSV – Obsahuje několik vzorových souborů pro import CSV a také několik vzorových exportů z aplikace.
- *DBSkripty* – Obsahuje vytvořující skript DB struktury.
- *Instalace* – Instalace vytvořená pomocí nástroje „Publish“ v MS Visual Studio 2005. Testuje přítomnost instaluje požadovaných prerekvizit a případně provede jejich instalaci.
- *Prehledy* – Obsahuje vzorová PDF vytvořená pomocí formuláře Přehledy.
- *Prerekvizity* – Samostatné instalační soubory komponent nutných pro korektní běh aplikace.
- *Project* – Adresář se zdrojovými soubory projektu.
- *Release* – Spustitelný build projektu (možné v případě, že jsou na příslušném počítači zmiňované prerekvizity).
- *Text* – Text této práce ve formátu PDF.

Příloha B – Databázové pohledy

Tato příloha obsahuje vytvořující skripty pro trojici pohledů, jejichž výstupy jsou pak využívány při tisku / exportu přehledů (viz kapitoly 4 a 5). Zbývající pohledy pracují pouze s aktuálními daty již zobrazenými pomocí pohledů COINS nebo JOINEDCOINS, který je podrobněji popsán v kapitole 6.

B1 - Přehled „Seznam sbírek“

Cílem tohoto pohledu je zobrazit seznam všech sbírek (v samotné aplikaci omezený jen na sbírky přihlášeného uživatele). Použití FULL OUTER JOIN je nezbytné proto, aby byly zobrazeny i sbírky bez vzoru (neodkazují na C_TCOLLECTION) a nemincovní sbírky (neodkazují na C_CTYPE).

```
CREATE VIEW [dbo].[COLLECTIONSSUMMARY] AS

SELECT COL_EVNUM, COL_NAME, CTYPE_DESC, TCOL_NAME,
COUNT(CNS_ID) AS COINCOUNT, (USR_FIRSTNAME + ' ' +
USR_LASTNAME) AS FULLNAME, USR_ID

FROM

COLLECTIONS LEFT OUTER JOIN C_CTYPE
      ON COL_CTYPE_ID = CTYPE_ID

FULL OUTER JOIN C_TCOLLECTION
      ON COL_TCOL_ID = TCOL_ID

FULL OUTER JOIN COINS
      ON CNS_COL_ID = COL_ID

INNER JOIN USERS
      ON COL_USR_ID = USR_ID

GROUP BY USR_ID, USR_LASTNAME, USR_FIRSTNAME, COL_EVNUM,
COL_NAME, CTYPE_DESC, TCOL_NAME
```

B2 - Přehled „Seznam všech předmětů“

Účelem tohoto pohledu je vytisknout seznam všech předmětů (v samotné aplikaci opět omezený jen na předměty přihlášeného uživatele), včetně údajů o nadřazené sbírce.

```
CREATE VIEW [DBO].[COINSSUMMARY] AS

SELECT COL_EVNUM, COL_NAME, CNS_EVNUM, CNS_VALUE, CNS_YEAR,
STATE_NAME, (USR_FIRSTNAME + ' ' + USR_LASTNAME) AS FULLNAME,
USR_ID

FROM

COINS LEFT OUTER JOIN COLLECTIONS
      ON CNS_COL_ID = COL_ID

LEFT OUTER JOIN C_STATE
      ON CNS_STATE_ID = STATE_ID

INNER JOIN USERS
      ON COL_USR_ID = USR_ID
```

B3 - Přehled „Chybějící v typizovaných sbírkách“

Tento pohled vypisuje pro všechny sbírky které obsahují odkaz na typickou sbírku seznam v nich chybějících předmětů. Podobně jako u pohledů volaných přímo v aplikaci při změnách datavého zdroje pro uživatelský ovladací prvek DataGridView je zde využito odečtení výsledků.

```
CREATE VIEW [DBO].[MISSINGINTYPEDCOLLECTIONS] AS

SELECT COL_USR_ID, COL_EVNUM, COL_NAME, TCOIN_VALUE AS VALUE,
TCOIN_YEAR AS YEAR, (USR_FIRSTNAME + ' ' + USR_LASTNAME) AS
FULLNAME

FROM

COLLECTIONS INNER JOIN C_TCOINS
      ON COL_TCOL_ID = TCOIN_TCOL_ID

INNER JOIN USERS
      ON COL_USR_ID = USR_ID

EXCEPT

SELECT COL_USR_ID, COL_EVNUM, COL_NAME, CNS_VALUE AS VALUE,
CNS_YEAR AS YEAR, (USR_FIRSTNAME + ' ' + USR_LASTNAME) AS
FULLNAME

FROM

COLLECTIONS INNER JOIN COINS
      ON CNS_COL_ID = COL_ID

INNER JOIN USERS
      ON COL_USR_ID = USR_ID
```

Příloha C – instalace databáze

V této příloze je popsán postup pro nainstalování databáze a nakonfigurování aplikace. Postup byl testován na Microsoft SQL Server 2005 Management Studio Express.

C1 – Instalace databáze

Před samotným vytvořením struktury je nezbytné vytvořit na cílovém serveru databázi (název katalogu „VINDRA“, schéma typu „dbo“), založit uživatelský účet s příslušnými právy k této databázi a v neposlední řadě nastavit mód „Server Authentication“ na hodnotu „SQL Server and Windows Authentication mode“, jinak na serveru není umožněno vzdálené připojení s SQL přihlašovacími údaji.

Po provedení těchto kroků je možné vytvořit kompletní strukturu databáze spuštěním SQL skriptu *DBStructureCreate.sql*, který je umístěn v adresáři DBSkripty na přiloženém CD. Pro správnou funkčnost aplikace je následně nezbytné spustit ještě tamtéž umístěný skript *DBInsertData.sql*, který jednak provede naplnění číselníkových tabulek C_STATE a C_CTYPE, jednak vytvoří uživatelský účet s přístupovými údaji admin / admin a maximálními právy.

C2 – Konfigurace aplikace

Pro připojení k nově vytvořené databázi je třeba přidat příslušný ConnectionString do konfiguračního souboru app.config. Toto je možné provést jednak ručně, jednak pomocí okna „Nastavení připojení“ v úvodním přihlašovacím formuláři aplikace. Nedoporučuje se zasahovat do defaultního ConnectionStringu, a proto doporučuji vytvořit nový s následujícími parametry:

- Data Source: adresa pro připojení k SQL serveru
- Initial Catalog: VINDRA

- User: Název účtu SQL uživatele
- Pass: Přístupové heslo SQL uživatele
- Timeout: 1 – 3 (hodnota v sekundách)

Po potvrzení vložení tohoto ConnectionStringu je možné zvolit nové připojení v roletce na kartě „Nastavení připojení“ a připojit se s přístupovými údaji admin / admin, případně vytvořit nový uživatelský účet.

Příloha D – Struktura CSV souborů

V této příloze je na několika příkladech popsána struktura CSV souborů pro export / import. Vzorové soubory jsou umístěny v adresáři CSV na přiloženém CD.

D1 – Export CSV

Při exportu z hlavního formuláře se výsledná struktura souboru liší podle toho, zda se jedná o mincovní, nebo nemincovní sbírku. V obou případech jsou pro každý předmět exportovány tři hodnoty:

- Pro mincovní sbírku se jedná o `HODNOTU`, `ROČNÍK` a `EV. ČÍSLO`
- Pro nemincovní sbírku se jedná o `EV. ČÍSLO`, `NÁZEV` a `POPIS`

Při exportu z formuláře „Správa typických sbírek“ se výsledná struktura liší podle toho, zda se jedná o sbírku typů, nebo typů a ročníků.

- Pro sbírku typů a ročníků se jedná o `HODNOTU`, `ROČNÍK`, `NÁZEV` a `POPIS`
- Pro sbírku typů se jedná o `HODNOTU`, `NÁZEV` a `POPIS`

D2 – Import CSV

Při importu do sbírky ve formuláři „Správa typických sbírek“ musí mít importovaný soubor následující strukturu:

- `HODNOTA`, `ROČNÍK`, `NÁZEV` a `POPIS`

Při vkládání neceločíselné hodnoty je nutné použít desetinnou tečku, nikoli čárku (došlo by k chybě v parsování). Importované hodnoty procházejí stejnými validacemi, jako hodnoty vkládané ručně.