Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Aplikace pro spouštění uživatelských funkcí v ekonomickém systému Stormware POHODA

Bakalářská práce

Autor: Jakub Schejbal

Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. RNDr. Petra Poulová, Ph.D.

Hradec Králové duben 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 3.4.2023 Jakub Schejbal

Poděkování:

Rád bych tímto poděkoval vedoucí práce, doc. RNDr. Petře Poulové, Ph.D., za metodické vedení, ochotu a věnovaný čas při realizaci této práce.

Anotace

Cílem práce je navrhnout a implementovat aplikaci pro spouštění uživatelských funkcí v ekonomickém systému Stormware POHODA. Účelem aplikace je umožnit spouštění uživatelsky definovaných funkcí, které by využívaly přímý přístup do databáze systému. Aplikace je primárně zaměřena na řadu systému POHODA, která využívá databázi typu Microsoft Access. Implementace aplikace je provedena v jazyce C# s využitím open source frameworku .NET 6 a je určena pouze pro operační systém Windows. Aplikace poskytuje možnost výběru databáze k připojení a umožňuje volit funkce ze seznamu funkcí, které je schopna dynamicky načítat z adresáře. Zároveň zajišťuje řízení připojení k databázi, včetně využití transakcí, a zaznamenává průběh funkce. Zdrojový kód implementované aplikace je přílohou této práce.

Annotation

Title: User Functions in the Economic System Stormware POHODA

The aim of this thesis is to design and implement an application for launching user functions in the Stormware POHODA economic system. The purpose of the application is to enable the execution of user-defined functions that would use direct access to the system database. The application primarily targets at the POHODA system series, which uses a Microsoft Access type database. The implementation of the application is done in C# using the open-source .NET 6 framework and is intended for the Windows operating system only. The application provides the possibility to select the database to connect to and allows to select functions from a list of functions that it can dynamically load from the directory. It also provides database connection management, including transaction usage, and logs the progress of the function. The source code of the implemented application is an appendix to this paper.

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

1 Úvod

Standardní funkcionality, které jsou implementovány v rámci aplikací, obvykle představují prostředek pro provádění uživatelsky univerzálních operací. Jedná se o funkcionality, které předpokládají velmi obdobné potřeby standardních uživatelů, a mohou tak tedy poskytovat jednotné řešení za použití univerzálně definovaného scénáře. V případech, kdy by se požadavek uživatele mohl lišit od standardního scénáře, mohou poskytovat možnost lehkého uzpůsobení svého průběhu prostřednictvím dodatečných parametrů. Hlavním cílem standardních funkcionalit je poskytovat službu, která je přínosná pro široký okruh uživatelů dané aplikace.

V některých případech se ovšem může vyskytnout potřeba provádět operace, které nelze označit za standardní. Jejich scénář je zaměřen pouze na specifický problém konkrétního uživatele či malé skupiny, což má za následek skutečnost, že takovéto funkcionality nepředstavují pro ostatní uživatele žádný přínos. Začlenění všech uživatelských funkcí do aplikace by mělo za následek zahlcení aplikace velkým množstvím funkcionalit, ačkoliv každý uživatel by využíval pouhý zlomek z nich. V důsledku toho by došlo ke zvýšení velikosti a komplexnosti aplikace, což by mělo za následek sníženou přehlednost a celkové snížení uživatelské přívětivosti.

Jako řešení potřeby provádět uživatelsky specifické operace poskytují některé aplikace možnost definování uživatelských funkcí, které jsou následně tyto aplikace schopny načítat do svého prostředí, poskytovat jim některé své prostředky a umožňovat jejich spouštění. Ne všechny aplikace však disponují touto schopností, a proto je nutné hledat vhodné řešení v rámci zbývajících možností. Jednou z možných cest by mohlo být využití externí aplikace, která by umožňovala spouštět uživatelské funkce v rámci svého prostředí, namísto aby byly spouštěny přímo v aplikaci.

Právě shledání se s několika případy, kdy by možnost definování uživatelských funkcí v ekonomickém systému Stormware POHODA mohla nahradit zdlouhavou rutinní manuální činnost a žádná z dostupných možností neposkytovala vhodné řešení, bylo podnětem k myšlence ověření možnosti návrhu spouštění uživatelských funkcí v systému POHODA a ke vzniku této práce.

2 Systém Stormware POHODA

Systém Stormware POHODA, dále uváděný také pouze jako „systém POHODA“, představuje softwarový produkt, zaměřený na ekonomiku a účetnictví. Tvůrcem tohoto systému je česká společnost STORMWARE s.ro., dále uváděná také pouze jako „společnost STORMWARE“. [1](i)

2.1 Společnost STORMWARE

Společnost Stormware uvádí dobu svého působení v oblasti vývoje aplikací v České republice od roku 1993 [1](i). Dle údajů dostupných ke květnu roku 2022 [2](ii) má společnost přes 150 zaměstnanců, kteří působí v celkem šesti pobočkách v České republice a čtyřech pobočkách na Slovensku. V České republice se jedná konkrétně o města Praha, Brno, Hradec Králové, Ostrava, Olomouc a Plzeň [2](ii).

Jak společnost STORMWARE uvádí na svých webových stránkách [2](ii), je certifikovaným partnerem společnosti Microsoft s titulem Microsoft Gold Certified Partner[1]. Tento titul získala v roce 2014 [2](ii). Společnost STORMWARE je zároveň držitelem několika dalších titulů a certifikací, které během svého působení na českém trhu získala. Jako podstatnou lze vyzdvihnout například certifikaci dle normy ČSN EN ISO 9001:2016[2] [4, s. 20], kterou úspěšné získala v oboru *„[…] vývoje, implementace a podpory softwarových produktů a včetně organizace a provádění seminářů a kurzů“* [4, s. 20].

Kromě již zmíněného systému POHODA, je tato společnost tvůrcem několika dalších softwarových produktů, mezi kterými lze nalézt například systém zaměřený na personalistiku a mzdy či systém specializovaný na daňová přiznání. [2](ii)

---

[1] *„Titul Microsoft Gold Certified Partner je známkou profesionality služeb nejvyšší úrovně, přístupu k nástrojům a podpoře ze strany společnosti Microsoft, odpovídající výjimečnému postavení společnosti na trhu.“* [2](iii)

[2] *„ISO 9001 poskytuje rámec řízení jakosti, který mohou společnosti využít k zajištění konzistentní kvality svých produktů a služeb. Společnosti si vybírají certifikaci ISO 9001, aby prokázaly, že dbají na dodržování vysokých standardů. Snižují tak pravděpodobnost závad produktů a jejich stahování z trhu nebo servisních nedostatků a zajišťují, že u nich mohou jejich zákazníci nakupovat s důvěrou.“* [3]

2.2 Systém POHODA

Systém POHODA je jedním z produktů vytvářených společností STORMWARE. Jedná se o počítačový software, který je určen pro osoby i organizace, kterým slouží, jak je uvedeno v uživatelské příručce systému POHODA [4], *„[...] nejen pro zpracování účetnictví, skladového hospodářství, majetkové evidence, personalistiky a mezd, ale i pro správu obchodních kontaktů a každodenní získávání aktuálních ekonomických a obchodních informací o svých firmách.“* [4, s. 11]. Jedná se tedy o poměrně komplexní systém, který se snaží obsáhnout velké množství potřeb, které se mohou u podniku vyskytnout.

Takto široké pokrytí různých potřeb podniku je umožněno díky celé řadě specializovaných agend v systému POHODA. Tyto agendy poskytují struktury a potřebnou funkcionalitu pro danou oblast a obvykle jsou propojeny s ostatními agendami systému. Z přehledu podporovaných funkcí lze, komě dalších, uvést například zpracování účetnictví a daňové evidence, obchodní evidence, skladové hospodářství, mzdovou evidenci či evidenci majetku [4, s. 17-18]. Systém zároveň disponuje možností rozšíření, například o licenci, která zpřístupní cizojazyčné uživatelské rozhraní v anglickém jazyce [4, s. 18]. Množství dostupných funkcionalit, které může uživatel používat, je ovšem závislé na konkrétní variantě a řadě systému, se kterou uživatel pracuje [4, s. 15]. Více o variantách a řadách systému POHODA je uvedeno v kapitole 2.4.

Vývoj systému POHODA probíhá v souladu s platnou účetní legislativou. Dokladem toho jsou účetní a daňové audity systému, které úspěšně proběhly v roce 2007 a v roce 2012 [2](iv). Jak uvádí společnost STORMWARE k auditu z roku 2012, *„Účelem auditu bylo ověřit, zda produkt splňuje vysoké nároky kladené na ekonomický informační systém určený pro vedení účetnictví, resp. daňové evidence v souladu s českou účetní legislativou.“* [2](iv). Souhrn důležitých norem, které se vztahují k oblasti působení systému, a které systém POHODA splňuje, je detailně popsán v rámci uživatelské příručky [4, s. 19-20] Snaha o soulad s legislativou je také patrná z častých aktualizací systému POHODA, které obvykle reflektují aktuální legislativní změny.

2.3 Popularita systému POHODA

Využití specializovaného účetního softwaru pro vedení účetnictví a pro podporu dalších aktivit podniku lze v dnešní době označit za běžnou praxi. Na českém trhu je možno vybírat z celé řady takovýchto systémů [5, 6, 7]. Jsou dostupná řešení, která mohou využívat jedinci či malé společnosti, které využívají pouze základní funkce systému a nevyžadují příliš pokročilé funkcionality. Zároveň lze ale také nalézt systémy, které jsou určeny pro velké společnosti, u nichž je třeba zajistit možnost současné práce několika desítek či dokonce stovek uživatelů, a kde jsou vyžadovány pokročilé funkcionality a automatizace. Výběr systému se tak vždy odvíjí od konkrétních požadavků zákazníka, který systém poptává.

Přehled a porovnání nejpoužívanějších účetních systémů v České republice uvádí Svobodová a Černá (2016) [5]. Systém POHODA se v tomto přehledu [5, s. 3] umístil na první příčce s více jak 150 000 instalacemi. V případě porovnání s ostatními účetními systémy [5, s. 3], které měli více než 10 000 instalací, se tak jednalo o 62,5 % z celkového počtu licencí. Dle celkového porovnání, které Svobodová a Černá uvádí [5, s. 3], byl v roce 2016 systém POHODA nejpoužívanějším účetním systémem v České republice.

Porovnání počtu instalací účetních systémů v České republice uvádí také Svobodová a Hedvičáková (2018) [6]. V tomto případě byly porovnávány účetní systémy s více jak 5000 instalacemi, celkem tedy 7 různých systémů [6, s. 430]. Systém POHODA se, v porovnání s ostatními účetními systémy [6, s. 430], umístil opět na prvním místě. S celkovým počtem instalací více jak 200 000 [6, s. 430] byl tedy v roce 2018 opět nejpoužívanějším účetním systémem v České republice. V porovnání s rokem 2016 došlo k nárůstu počtu aktivních instalací systému POHODA přibližně o 50 000.

V současné době společnost STORMWARE na svých webových stránkách uvádí, že poskytuje 212 000 aktivních licencí systému POHODA [2](ii) (údaj k 29.4.2022). Oproti roku 2018 jde opět o další navýšení. Jedná se tak o nemalý počet uživatelů, kteří tomuto systému důvěřují a aktivně ho využívají. Jak uvádí Svobodová a Černá [5], systém POHODA užívají uživatelé, kteří *„[…] chtějí platit malou částku […] a nevadí jim užívání balíčkového softwaru.“* [5, s. 5, volný překlad].

Ohodnocení a porovnání účetních systémů v České republice provedla například i Koběrská (2018) [7], která porovnala systémy POHODA, HELIOS Red, Ježek Software, KELOC, KOSYS, PREMIER systém a ABRA. Ve shrnutí přehledu jednotlivých systémů Koběrská (2018) vyzdvihuje u systému POHODA jeho uživatelskou přívětivost, přehlednost a intuitivní ovládání [7]. Zároveň konstatuje, že *„[…] systém POHODA je optimální volbou z hlediska ceny a výkonu.“* [7, s. 54].

Systém POHODA tak tedy dlouhodobě zastává pozici nejpoužívanějšího účetního systému v České republice. Tato dlouhodobá pozice může signalizovat, že stávající uživatelé jsou se systémem POHODA spokojeni. Navíc, dále stoupající počet instalací zároveň naznačuje, že je pro uživatele stále atraktivní volbou při výběru nového systému pro vedení účetnictví.

2.4 Varianty a řady systému

Systém POHODA je společností STORMWARE poskytován ve více řadách a variantách [2](v; vi; vii). Konkrétně lze vybírat ze tří řad systému, a to z řady POHODA [2](v), POHODA SQL [2](vi) a POHODA E1 [2](vii). Tyto řady se liší v úrovni zabezpečení dat, poskytované funckionalitě či v možnostech nastavení uživatelských práv. Podstatným rozdílem je také odlišná architektura systému a použitá databázová technologie. Nicméně, každá z řad obsahuje variantu s kompletní funckionalitou pro vedení účetnictví a daňové evidence.

Architektury, které jsou použity v jednotlivých řadách systému, jsou uvedeny v následující tabulce, viz Tabulka 1. Je patrné, že řady POHODA SQL a POHODA E1 využívají shodný druh architektury, který je odlišný od architektury použité v nižší řadě POHODA. [2](v; vi; vii)

Tabulka 1 – Architektury řad systému POHODA

	POHODA	**POHODA SQL**	**POHODA E1**
Architektura	File-Server	Klient-server	Klient-server

Zdroj: [2](v; vi; vii), vlastní zpracování

Obdobné rozdělení řad systému platí i v případě použitých databázových technologií. Stručný přehled použitých databázových technologií v jednotlivých řadách je uveden v následující tabulce, viz Tabulka 2.

Tabulka 2 – Databázové technologie řad systému POHODA

	POHODA	POHODA SQL	POHODA E1
Databázová technologie	MS Access	Microsoft SQL Server	Microsoft SQL Server

Zdroj: [2](v; vi; vii), vlastní zpracování

Každá z řad systému POHODA je dostupná v několika variantách, které uživatelům poskytují různé kombinace jednotlivých funkcionalit a agend systému. U řady POHODA se jedná konkrétně o varianty Mini, Lite, Jazz, Standard, Profi, Premium a Komplet [2](v). Řady POHODA SQL a POHODA E1 jsou dostupné v řadách Jazz, Standard, Profi, Premium a Komplet [2](vi; vii). Uživatel má tedy k dispozici velmi širokou škálu možností k tomu, aby nalezl nejvhodnější variantu, která maximálně odpovídá jeho požadavkům. V ideálním případě tedy lze nalézt variantu, která obsahuje veškerou potřebnou funkcionalitu a není zatížena žádnými nadbytečnými prvky, které by nechtěně navyšovaly pořizovací cenu systému, a to i v případě, že by je uživatel nevyužíval.

2.4.1 Řada POHODA

Dle informací uvedených na webových stránkách společnosti STORMWARE [2](v), je řada POHODA nejpoužívanější řadou systému a lze vybírat z celkem z osmi různých variant, které obsahují různé kombinace funkcionalit. Tato řada je určena primárně pro jedince či malé společnosti, které nevyžadují přístup velkého množství uživatelů. Zároveň je možná volba ze dvou různých režimů použití, kdy lze volit z nesíťové či síťové verze [2](v). V obou případech je ovšem pro ukládání dat využíván databázový systém Microsoft Access, který ukládá data do lokálního datového souboru v některém z připojených zařízení [2](v).

V případě nesíťové verze je tedy datový soubor, společně se samotnou aplikací, umístěn na stejném zařízení. Uživatel tak pracuje s daty, ke kterým nemusí vzdáleně

přistupovat, například prostřednictvím sítě, což umožňuje velmi rychlý přístup k těmto datům [2](v). Tento režim ovšem nelze využít v případě, kdy je třeba, aby více uživatelů pracovalo se stejnými daty.

Ke sdílení dat v menší skupině uživatel lze využít síťovou variantu řady POHODA, která využívá architekturu File-Server, díky které je k datům umožněn přístup více klientům ve společné síti [2](v).

2.4.1.1 Architektura File-Server

V případě architektury File-Server je sdílený datový soubor uložen pouze na jednom zařízení, které zastává pozici poskytovatele dat, jinak nazývaného také jako File Server [8, s. 172]. Toto zařízení následně poskytuje k uloženému datovému souboru přístup prostřednictvím sítě. Díky tomuto přístupu může s daty, která jsou uložena v takto sdíleném datovém souboru, pracovat více klientů na různých zařízeních [8, s. 172].

Jak uvádí Otte (2013) [8, s. 172], nevýhodou architektury File-Server je skutečnost, že systém řízení báze dat je provozován jednotlivě na zařízení každého jednoho klienta. Všechna potřebná data jsou tedy prostřednictvím sítě přenesena ze zařízení, na kterém je datový soubor uložen, na stranu klienta, kde teprve dochází ke zpracování dotazu, jako například vyhledávání či filtrování [8, s. 172]. Z tohoto důvodu dochází k velkému přenosu dat po síti, což má za následek vyšší zatížení sítě [8, s. 172]. Jelikož každý klient obsahuje systém řízení báze dat, je také někdy označován jako tzv. „Tlustý klient“ [8, s. 172]. Architektura File-Server tudíž není příliš efektivní v případech, kdy je třeba pracovat s velkým množstvím dat, nebo v případech, kdy je vyžadováno velké množství klientů, kteří by měly k těmto datům přistupovat a pracovat s nimi.

2.4.2 Řada POHODA SQL

Tato řada systému POHODA je určena pro použití v případech, kdy je třeba umožnit současnou práci většího množství uživatelů s přístupem do sdíleného datového souboru [2](vi). Může se jednat například o střední a velké společnosti, které zpracovávají velké množství dat. Řada POHODA SQL takovýto způsob přístupu

umožňuje díky architektuře Klient-Server [2](vi). Zároveň lze využít většího výkonu a zabezpečení systému, jelikož přístup k datům je umožněn pouze skrze jednoho poskytovatele a není tedy třeba provádět zabezpečení na několika různých místech [2](vi). Správa dat je v této řadě svěřena databázi Microsoft SQL Server [2](vi).

2.4.2.1 Architektura Klient-Server

V případě architektury Klient-Server, jak uvádí Otte (2013) [8, s. 172-173], jsou data umístěna na jednom zařízení, obvykle serveru, které zastává roli databázového serveru. Tento server je také jediným zařízením, na kterém běží systém řízení báze dat [8, s. 172]. Přes síť jsou přenášeny pouze požadavky jednotlivých klientů směrem na server a odpovědi s výsledky směrem na klienty [8, s. 173]. Tímto dochází k minimálnímu zatížení sítě a zmenšení velikosti klienta, který již nemusí obsahovat kompletní systém řízení báze dat, proto je také někdy označován jako tzv. „Tenký klient“ [8, s. 173]. Tato architektura je tedy efektivní i při velkém objemu spravovaných dat a velkém množství klientů, kteří k těmto datům přistupují.

2.4.3 Řada POHODA E1

Řada POHODA E1 představuje nejvíce pokročilou řadu ekonomického systému POHODA. Obdobně jako řada POHODA SQL je i POHODA E1 založena na architektuře Klient-Server. Systém v této řadě zároveň obsahuje řadu pokročilých funckionalit, jako například možnost detailní správy přístupových práv, rozšiřování existujících agend o nová datová pole či vytváření nových agend dle vlastních specifikací. POHODA E1 umožňuje také připojení dalších dodatečných rozšíření, díky kterým, jak uvádí společnost STORMWARE [2](vii), lze systém využívat jako dostupný systém ERP. Jedná se tedy o řadu, která je určena pro nejnáročnější uživatele, kteří chtějí využívat tento systém ke komplexnímu řízení aktivit svého podniku. [2](vii)

2.5 Požadavky na operační systém

Systémové požadavky pro instalaci a provozování systému POHODA jsou kompletně specifikovány v uživatelské příručce [4, s. 21-23]. Jedná se o specifikaci hardwarových požadavků, které definují například požadované vlastnosti

procesoru, operační paměti či pevného disku. Dále jsou také specifikovány požadavky softwarové, jako například operační systém a další softwarové doplňky. Tyto specifikace se liší v závislosti na řadě a variantě systému POHODA.

Následující tabulka, viz Tabulka 3, zobrazuje požadavky na desktopové a serverové operační systémy, které jsou systémem podporovány. V tabulce jsou uvedeny poslední a aktuální verze operačních systémů, starší verze byly vynechány.

Tabulka 3 – Podporované operační systémy

Desktopové operační systémy	Windows 11 Windows 10 Windows 8.1
Serverové operační systémy	Windows Server 2022 Windows Server 2019 Windows Server 2016 Windows Server 2012 R2 Windows Server 2012

Zdroj: [4, s. 22] [2](viii), vlastní zpracování

Jak je patrné z výše uvedené tabulky, viz Tabulka 3, systém POHODA je určen pouze pro desktopová zařízení s operačním systémem Windows nebo pro serverová zařízení se serverovým operačním systémem Windows Server. Žádné další operační systémy nejsou podporovány. Systém tedy nelze instalovat na zařízení s jiným operačním systémem.

Pro práci se systémem ze zařízení s nepodporovaným operačním systémem lze využít některé z řešení, které společnost STORMWARE uvádí jako možné alternativy [2](viii). První možností je provedení virtuální instalace operačního systému Windows na nepodporovaném operačním systému a následná instalace systému POHODA v takto vytvořeném virtuálním prostředí [2](viii). Druhou možností je připojení k jinému zařízení s operačním systémem Windows a instalací systému POHODA pomocí vzdáleného přístupu [2](viii). Třetí možností je využití hostingu a instalace systému POHODA na serverovém prostoru a následný vzdálený přístup na tento server [2](viii).

Toto omezení, pouze na operační systém Windows, může být pro některé uživatele nemilé a způsobovat jim řadu komplikací. Jak je však uvedeno v kapitole 2.1, společnost STORMWARE je certifikovaným partnerem společnosti Microsoft [2](ii). Je tedy pochopitelné primární zaměření na produkty společnosti Microsoft.

2.6 Požadované komponenty

Pro správné fungování systému POHODA je vyžadováno, aby na zařízeních byly přítomny některé komponenty, na kterých je systém závislý. Tyto komponenty jsou obvykle automaticky nainstalovány při instalaci samotného systému a není tak vyžadována žádná přímá akce uživatele. V této kapitole je zmíněna pouze jedna komponenta, která je velmi podstatná v dalších částech této práce.

Velmi ojediněle se může stát, že při některých aktualizacích operačního systému či aktualizacích produktů Microsoft Office dojde k chybnému navázání některých komponent, a je třeba provést několik opravných akcí manuálně [2](ix; x). Tento problém může nastat také v případě, že sám uživatel omylem poškodí či odinstaluje tyto komponenty, například společně s jiným softwarem. O výskytu tohoto problému je obvykle uživatel informován chybovou hláškou, která je zobrazena při spuštění systému POHODA [2](ix; x). Detailní postup pro opravu tohoto problému, včetně jednotlivých komponent ke stažení, je k dispozici na webových stránkách společnosti STORMWARE v sekci otázek a odpovědí k systému POHODA [2](ix; x).

2.6.1 Microsoft Access Database Engine

Tato komponenta je stěžejní pro fungování celého systému POHODA. V případě její absence či poškození nelze provést spuštění systému [2](ix; x). Prostřednictvím této komponenty, jak uvádí společnost Microsoft na webové stránce, kde je tato komponenta přístupná [9], *„[...] se nainstaluje sada komponent, které lze využít k usnadnění přenosu dat mezi soubory systému Microsoft Office [...] a aplikacemi jiných výrobců než Microsoft Office.“* [9, volný překlad]. V detailním popisu je pak uvedeno, že soubory, se kterými lze komunikovat, jsou například soubory typu

Microsoft Office Access nebo Microsoft Office Excel [9]. Podstatnou součástí této komponenty jsou také rozhraní ODBC[1] a OLE DB[2] [9].

2.7 Uživatelské prostředí

Systém POHODA pracuje primárně v režimu jednoho okna, které je zároveň také oknem hlavním. V případě potřeby je u některých funkcionalit systému toto okno doplněno dialogovými okny, která umožňují detailněji specifikovat případné parametry dané funkcionality. Při spuštění programu je v hlavním okně zobrazena výchozí domovská plocha, v uživatelské příručce nazývaná jako informační plocha [4, s. 55]. Tato plocha poskytuje přehledové informace o otevřené účetní jednotce, datum, jméno přihlášeného uživatele nebo informace o licenci programu [4, s. 56]. Zároveň jsou na této ploše dostupné různé přehledy ekonomických, mzdových a statistických údajů dané společnosti [4, s. 56]. Informační plocha je detailně popsána v uživatelské příručce [4].

Tabulka 4 – Hlavní oblasti okna agendy

1	**Hlavní nabídka**	V rámci hlavní nabídky jsou zpřístupněny hlavní příkazy systému.
2	**Standardní lišta**	Poskytuje základní paletu akcí, které jsou dostupné pro práci se záznamy.
3	**Formulář agendy**	Obsahuje pole pro jednotlivé údaje záznamů a umožňuje jejich přímou editaci.
4	**Panel agend**	Obsahuje ikony, které reprezentují agendy, se kterými uživatel aktuálně pracuje.
5	**Tabulka agendy**	Obsahuje řádkový přehled dostupných záznamů, které se nachází v otevřené agendě.

Zdroj: [4, s. 57], vlastní zpracování

Při práci se systémem, kdy uživatel pracuje s jednotlivými agendami systému, se ovšem nejčastěji setkává s tzv. oknem agendy. V tomto okně jsou zobrazovány,

[1] *„Microsoft Open Database Connectivity je rozhraní […], které umožňuje aplikacím přístup k datům z různých systémů řízení báze dat (DBMSs).“* [10, volný překlad]

[2] *„Rozhraní OLE DB poskytují aplikacím jednotný přístup k datům uloženým v různých informačních zdrojích nebo úložištích dat.“* [11, volný překlad]

vytvářeny a editovány záznamy jednotlivých agend. Celkově je okno rozvrženo tak, aby umožňovalo zobrazení všech záznamů dané agendy, a zároveň umožnilo nahlížet a editovat detailní informace o vybraném záznamu pomocí formuláře. Funkce hlavních částí okna uvádí následující tabulka, viz Tabulka 4. Kompletní popis okna agendy a všech jeho jednotlivých částí obsahuje uživatelská příručka k programu POHODA. [4, s. 57]

3 Datová komunikace

Systém POHODA disponuje celou řadou možností datové komunikace, které poskytují různé způsoby přenosu dat směrem ze systému ven, či na opak směrem do systému POHODA. Prostřednictvím této komunikace dokáže systém poměrně snadno a bezpečně komunikovat s jiným softwarem, ale i s dalšími externími zařízeními [4, s. 463]. Jedná se například o komunikaci se čtečkami čárových kódů, platebními terminály, elektronickými vahami či pokladnami a pokladními displeji [4, s. 466-472]. V případě spolupráce s jiným softwarem se jedná například o propojení s internetovými obchody či zpracování a vytváření bankovních souborů [4, s. 472-477].

3.1 ISDOC komunikace

Jak uvádí společnost STORMWARE [2](xi), systém POHODA plně podporuje komunikaci prostřednictvím formátu elektronické fakturace ISDOC, který je založen na značkovacím jazyce XML. Formát ISDOC umožňuje výměnu dokladů v elektronické podobě, kdy doklad v takovémto formátu může být snadno vytvořen jedním ekonomickým či účetním systémem a jiným systémem opět naimportován a zpracován [2](xi).

Systém POHODA je navíc ještě doplněn externím nástrojem, který usnadňuje práci právě s elektronickými doklady ve formátu ISDOC [2](xi). Jak je uvedeno v příručce k systému POHODA [4], *"STORMWARE ISDOC Reader je program, který slouží k zobrazení a kontrole elektronických dokladů ve formátu ISDOC."* [4, s. 465]. Tento program obsahuje také možnost přímého importu otevřených dokladů do systému POHODA [2](xi). Stejně tak lze provést import a export souborů ve formátu ISDOC přímo v systému POHODA [2](xi).

3.2 Import a export dat

Systém POHODA umožňuje provádět import dat některých agend do databáze systému. Tento import je možno provádět prostřednictvím průvodce pro import, kterého lze otevřít v nabídce horního menu programu [4, s. 477]. Data lze takto importovat z několika různých typů datových souborů. Dalším způsobem je využití

dalších dostupných možností přímo v okně agendy nad vybranými záznamy. V takovém případě lze možnosti exportu vyvolat prostřednictví nabídky po stisknutí pravého tlačítka myši [4, s. 477].

Průběh a způsob importu lze detailněji specifikovat nastavením v okně průvodce importem [4, s. 477]. K dispozici je několik režimů importu, kdy jsou importované záznamy zpracovávány různým způsobem. Volit tak lze z běžného, aktualizačního nebo nahrazujícího importu. Charakter každého importu je detailně popsán v uživatelské příručce [4, s. 477]

Export dat ze systému POHODA je možno provést více různými způsoby. V horním menu okna lze otevřít okno průvodce exportem dat, který umožňuje specifikovat parametry exportu. [4, s. 478]. Výstupem může být opět několik různých typů datových souborů, které budou obsahovat data zvolených záznamů. Exportovat lze rovněž záznamy vybrané v okně agendy prostřednictvím nabídky pod pravým tlačítkem myši [4, s. 479].

3.3 XML komunikace

Systém POHODA poskytuje možnost datové komunikace prostřednictvím souborů XML, které jsou systému předány [4, s. 480]. Pro každou agendu, která podporuje XML komunikaci, je definována struktura vstupního souboru XML pro jednotlivé akce prostřednictvím souboru XSD [2](xii). Soubory pro komunikaci jsou umístěny do specializované složky, ze které jsou systémem při spuštění komunikace načteny [4, s. 480]. Díky schématům XSD je zajištěna vysoká bezpečnost akcí vykonávaných při XML komunikaci a lze snadno provádět kontrolu vstupních dat. Případné chyby a nepřesnosti v datových souborech jsou tak velmi rychle odhaleny, komunikace je ukončena a chyby zapsány do výstupních souborů ve specializované složce [4, s. 480].

Možnost XML komunikace ovšem podporují pouze vybrané agendy, což může v některých případech omezovat použití této komunikace. Seznam agend, které podporují XML komunikaci je uveden na webových stránkách [2](xii), kde lze také nalézt vzorové soubory pro jednotlivé agendy. Seznam těchto agend je zobrazen i

následující tabulce, viz Tabulka 5. Agendy jsou v této tabulce rozděleny do dvou skupin, dokladové a seznamové, dle typu agendy.

Tabulka 5 – Agendy podporující XML import

Dokladové agendy	Seznamové agendy
- Faktury - Objednávky - Nabídky - Poptávky - Zakázky - Příjemky - Výdejky - Převodky - Výroba - Interní doklady - Pokladní doklady - Prodejky - Banka	- Adresy - Zásoby - Členění skladů - Sklady - Inventurní seznamy - Předkontace - Uživatelské seznamy - Dodavatelské zásoby - Číselné řady - Bankovní účty - Volitelné parametry - Parametry internetového obchodu - Kategorie internetového obchodu - Skupiny zásob

Zdroj: [2](xii), vlastní zpracování

XML komunikaci lze spouštět z grafického uživatelského rozhraní nebo prostřednictvím příkazu v příkazové řádce. Samotný průběh XML komunikace lze specifikovat pomocí několika parametrů. Nastavením lze specifikovat, zda zpracovávat pouze jeden soubor, nebo zda všechny soubory ve zvolené složce. Výsledky XML komunikace jsou zapsány do výstupní složky, ze které je lze následně zobrazit a prověřit informace o průběhu komunikace [4, s. 481]. Zároveň jsou zaznamenávány informace o provedených XML importech. Pro tento účel je dostupná agenda XML Log, ve které lze zobrazit záznamy importů a jednotlivé doklady [4, s. 482].

4 Přímý přístup do databáze systému POHODA

Jak je uvedeno v kapitole 2.4, systém POHODA, v závislosti na konkrétní řadě systému, využívá různou databázovou technologii. Jedná se o databázový systém Microsoft Access v případě řady POHODA [2](v) a databázový systém Microsoft SQL Server v případě řad POHODA SQL a POHODA E1 [2](vi; vii). Následující části této práce jsou zaměřeny pouze na řadu POHODA a databázový systém Microsoft Access.

4.1 Motivace

Někteří uživatelé systému POHODA se dostávají do situací, kdy žádná ze zabudovaných funkcí systému neposkytuje jimi požadovanou funkcionalitu. Tyto situace velmi často nastávají například v případech, kdy uživatel zpracovává velké množství dat, která prošla skrze několik systémů třetích stran. V takových případech byla data několikrát exportována a importována, přičemž mezi tím s nimi byly prováděny různé operace.

Pro demonstraci uvažujme jednoduchou fiktivní situaci, kdy uživatel vede účetní jednotku, která generuje účetní doklady v externí aplikaci a zároveň využívá služeb dopravce pro přepravu zásilek. V tuto chvíli se jedna informace vydává dvěma různými směry. V prvním případě jsou doklady z externí aplikace exportovány a standardní cestou naimportovány do systému POHODA. Předpokládejme, že importní soubory s daty jsou ve formátu, který je systémem POHODA podporován. Již v tuto chvíli však mohou nastat první komplikace, jako například, že informace v externí aplikaci byly zadány do jiného pole či že obsahují chybu způsobenou lidskou nepozorností. Příklad importovaného dokladu zobrazuje následující tabulka.

Tabulka 6 - Příklad informace z externí aplikace

Variabilní symbol	Jméno	Datum	Částka k uhrazení	Poznámka
1234	Jan Novák	10.10.2022	400	

Zdroj: Autor, vlastní zpracování

Předpokládejme však, že importovaná data neobsahují žádné chyby a import proběhl v pořádku. V druhém směru jsou data z externí aplikace předána přepravci, který je pomocí importu zavedl do svého systému, kde jsou s nimi prováděny další

operace. Systém dopravce ovšem tyto operace řídí na základě vlastního variabilního symbolu a původní variabilní symbol je ztracen či, v tom lepším případě, přesunut do pomocného textového pole, například poznámky. Ze systému dopravce jsou následně data exportována a importována do systému POHODA. Na jejich základě je třeba v systému POHODA provádět určité operace.

Tabulka 7 - Příklad informace z aplikace dopravce

Variabilní symbol	Jméno	Datum	Uhrazená částka	Poznámka
5678	Jan Novák	10.10.2022	400	1234

Zdroj: Autor, vlastní zpracování

Výsledkem tohoto postupu je, že v systému POHODA se nachází doklad a informace o jeho uhrazení, které k sobě na první pohled náleží (viz Tabulka 8). Systém POHODA ovšem není schopen tyto informace automaticky propojit, jelikož jednou z hlavních podmínek pro provedení této akce je shodný variabilní symbol.

Tabulka 8 - Spojení informace dopravce a dokladu

Úhrada				
Variabilní symbol	**Jméno**	**Datum**	**Uhrazená částka**	**Poznámka**
5678	Jan Novák	10.10.2022	400	1234
Neshodný variabilní symbol. Třeba hledat v poznámce.	Shodné.	Shodné.	Shodné.	
Doklad				
Variabilní symbol	**Jméno**	**Datum**	**Částka k uhrazení**	**Poznámka**
1234	Jan Novák	10.10.2022	400	

Zdroj: Autor, vlastní zpracování

Pro osobu, která tyto doklady zpracovává, tedy není jiné cesty, než tyto informace ručně vyhledávat a párovat. V závislosti na velikosti účtované společnosti se může množství dokladů, u kterých je třeba provést takovéto ruční zpracování, pohybovat ve velmi vysokých číslech.

Komunikace prostřednictvím XML, kterou systém POHODA poskytuje, by mohla představovat možnou cestu k nalezení řešení tohoto problému. Existuje ovšem několik omezení, kvůli kterým není tato možnost vhodná. Hlavní překážkou je

omezené množství agend systému POHODA, které podporují komunikaci prostřednictvím XML. Další překážkou je omezená škála operací, které lze prostřednictvím komunikace XML provádět.

Pokud tedy nelze využít žádnou z výchozích dostupných možností, je třeba hledat alternativu, která by poskytovala širší škálu funckionalit. Lze předpokládat, že pokud taková alternativa existuje, bude její použití komplikovanější, něž u dostupných výchozích možností. Zároveň lze předpokládat, že úroveň bezpečnosti této komunikace, především z hlediska správnosti a integrity dat, bude oproti dostupným výchozím možnostem velmi nízká. Je tedy třeba nalézt alternativu schopnou vykonat potřebné operace a provést maximální opatření k vyvážení poměru funkčnost / bezpečnost.

4.2 Uložení dat

V této části je popsáno umístění a struktura adresářů, které systém POHODA využívá pro ukládání dat. Dále jsou uvedeny informace o datových souborech, které jsou v těchto adresářích umístěny. Všechny tyto informace jsou uvedeny a volně dostupné v rámci uživatelské příručky k systém POHODA [4].

4.2.1 Složky systému

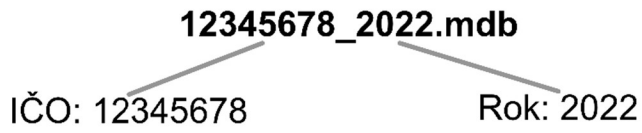
Jak uvádí webová příručka systému [4, s. 25], při instalaci systému je vytvářena celá řada složek a podsložek. Kompletní popis složek a jejich účelů je detailně popsán v uživatelské příručce k systému POHODA [4, s. 25].

V uživatelské příručce lze také nalézt složku, která slouží k uložení datových souborů systému. Datové soubory, ve kterých jsou uložena veškerá účetní data, jsou uloženy ve složce *Data* [4, s. 25]. Tato složka se nachází v rámci instalační složky. Ve složce *Data* se také nachází podsložka *Zálohy*, která slouží k ukládání záloh datových souborů [4, s. 25].

4.2.2 Datové soubory

Jak uvádí webová příručka [4, s. 26], systém POHODA ukládá data účetních jednotek do datových souborů formátu databáze Microsoft Access 2000. Jedná se o datové

soubory s příponou **.mdb*, které se nachází ve složce *Data* [4, s. 94]. Výchozí pojmenování těchto souborů, jak uvádí uživatelská příručka [4], je kombinací identifikačního čísla společnosti a roku, který je v souboru obsažen, jak je znázorněno na následujícím obrázku, viz Obrázek 1 [4, s. 94]. Systém POHODA ovšem umožňuje uživateli zadat libovolný název při zakládání účetní jednotky nebo soubor kdykoliv přejmenovat [4, s. 94].



Obrázek 1 – Struktura názvu datového souboru
Zdroj: [4, s. 94], vlastní zpracování

4.3 Přístup k datům

Data účetních jednotek jsou uložena v datových souborech formátu Microsoft Access 2000 [4, s. 26]. Pro přístup k datům v těchto souborech by bylo třeba využít některých dostupných ovladačů, které podporují připojení k tomuto typu datového souboru. Jak je uvedeno v kapitole 2.6.1, v rámci instalace systému POHODA je instalována také komponenta Microsoft Access Database Engine, která takovýto ovladač obsahuje [9].

Jedná se o ovladač ODBC [9]. Vzhledem k tomu, že je tento ovladač instalován v rámci instalace systému POHODA a komponenta, ve které je obsažen, je pro správné fungování systému vyžadována, lze se spolehnout na skutečnost, že bude v systému v případě přítomnosti systému POHODA nainstalován. V takovém případě nebude nutno vyžadovat jeho instalaci a bude možno ho využít k přístupu do datového souboru.

V konkrétní implementaci daného programovacího jazyka je pak možno využít knihovnu, která poskytuje přístup k funkcionalitě tohoto ovladače. K vytvoření připojení k datovému souboru je třeba nejprve definovat připojovací řetězec, který obsahuje potřebné údaje, jako například typ souboru, cestu k souboru či přístupové

údaje. Pro připojení k souboru Microsoft Access 2000 má tento řetězec následující formát.

Driver={Microsoft Access Driver (.mdb)};Dbq=**[soubor]**;*

Zdroj: [10]

V případě uživatelského jména a hesla jsou do připojovacího řetězce přidány další údaje.

Driver={Microsoft Access Driver (.mdb)};Dbq=**[soubor]**;Uid=**[jméno]**;Pwd=**[heslo]**;*

Zdroj: [12]

Na základě takovéhoto připojovacího řetězce je možno vytvořit spojení s datovým souborem a skrze třídy a metody knihovny pracovat s daty v souboru s využitím standardního jazyka SQL.

4.4 Výhody

V případě přímého připojení k datovému souboru systému POHODA lze využít komunikace s využitím standardního jazyka SQL (Structured Query Language). Pomocí příkazu SELECT [13, s. 69] se tak lze přímo dotazovat na data obsažená v datovém souboru. Zároveň lze, s použitím podmínek v příkazu WHERE, data libovolně filtrovat [13, s. 124] a pomocí příkazu ORDER BY také libovolně řadit. Pomocí dalších příkazů, jako je INSERT, UPDATE a DELETE lze s těmito daty dále manipulovat. Tímto způsobem lze bezmezně přistupovat k veškerým datům v datovém souboru a téměř neomezeně s nimi manipulovat. Díky tomu zanikají veškerá omezení, která vytvářela překážku v případě využití XML komunikace.

4.5 Nevýhody

V důsledku přímého a neomezeného přístupu do datového souboru je snížena celková bezpečnost prováděných operací. Jak je zmíněno v kapitole 4.1, kvůli zvýšení flexibility došlo ke snížení bezpečnosti. Při tomto způsobu přístupu do datového souboru je prováděna pouze minimální kontrola operací, které se s daty provádí. Jedná se tedy pouze o výchozí kontrolu prvků databázové integrity.

Tímto způsobem tak lze snadno a nenávratně poškodit či zničit data nebo celý datový soubor. Dalším možným rizikem je nechtěná změna dat takovým způsobem, který z počátku nemusí působit problémy a může být nepovšimnut, nicméně bude ovlivňovat jiné procesy v systému a způsobovat, že výstupy ze systému budou chybné. Společnost STORMWARE v rámci uživatelské příručky [4] varuje před přímým přístupem do databáze systému:

„*Pozor, neodborným zásahem do datových souborů MDB, otevřených z jiného programu než z programu POHODA, se vystavujete nebezpečí nenávratného porušení databáze a vztahů mezi jednotlivými tabulkami. Výrobce v žádném případě neručí za takto způsobené problémy!*“ [4, s. 26]

4.6 Závěr

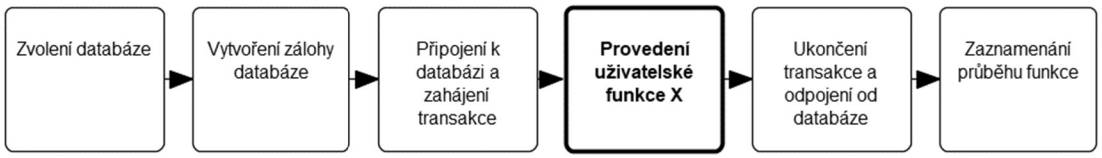
Přímý přístup do databáze systému POHODA poskytuje maximální flexibilitu a neomezené možnosti prováděných operací. Zároveň ovšem s sebou nese velké riziko poškození či zničení databáze. V každém případě je tedy výhodnější využít některou z výchozích možností komunikace, které systém POHODA poskytuje.

Nicméně v některých případech nelze využít žádnou z těchto možností. Jedná se například o situace, kdy je třeba provádět rutinní operaci v takovém množství opakování, že není manuálně proveditelná. V takových případech je přímý přístup do databáze jedinou možností a je třeba udělat maximum pro to, aby tento způsob byl co možná nejbezpečnější.

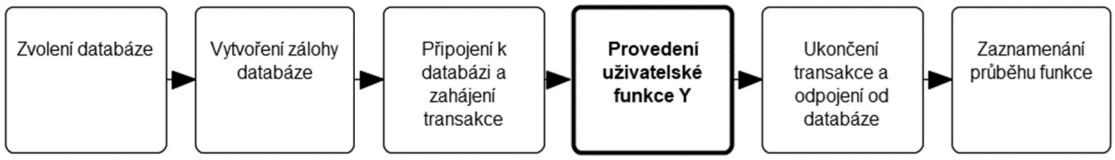
Pro zvýšení bezpečnosti lze provádět například automatické zálohování datového souboru před vykonáním operace, detailní zaznamenávání průběhu operace, využívat transakce pro ošetření případů výskytu chyby, omezit množství ovlivňovaných dat na nezbytné minimum či danou operaci dostatečně otestovat a ověřit její důsledky. Vždy je ovšem třeba mít na paměti, že operace prováděné tímto způsobem nejsou výrobcem systému POHODA doporučeny [4, s. 26] a jsou proto prováděny na vlastní riziko.

5 Návrh aplikace

Ve chvíli, kdy bylo implementováno několik na sobě nezávislých, uživatelských funkcí v podobě samostatných aplikací bylo zjištěno, že jednotlivé funkce obsahují velké množství shodného kódu. Tato skutečnost je znázorněna v následujících obrázcích, viz Obrázek 2 a Obrázek 3, které zobrazují dvě nezávislé aplikace X a Y, kdy každá z těchto aplikací obsahuje a vykonává právě jednu uživatelskou funkci X a Y. V obrázcích jsou zjednodušeně, blokově znázorněny jednotlivé etapy průběhu uživatelské funkce v jednotlivých aplikacích.



Obrázek 2 - Průběh aplikace s uživatelskou funkcí X
Zdroj: Autor, vlastní zpracování



Obrázek 3 - Průběh aplikace s uživatelskou funkcí Y
Zdroj: Autor, vlastní zpracování

Jednotlivé etapy průběhu uživatelských funkcí se sestávají ze zvolení databáze a vytvoření zálohy zvolené databáze. Zálohování je prováděno z preventivních důvodů, aby byla zachována původní verze datového souboru pro případ jeho poškození. Následuje připojení k databázi a zahájení transakce. Transakce jsou využívány, aby byla zajištěna konzistence dat v případě, kdy by v průběhu funkce nastala chyba. Poté je vykonána samotná uživatelská funkce s využitím zvolené databáze a vytvořené transakce. Po provedení uživatelské funkce, v případě úspěšného i neúspěšného ukončení, je transakce ukončena a aplikace je odpojena od databáze. Na závěr je celý průběh funkce zaznamenán do souboru, aby bylo možno zpětně dohledat případné problémy a chyby při provádění funkcí.

Z obrázků zobrazujících průběh funkcí X a Y je patrné, že aplikace X a Y se liší pouze v kódu samotných uživatelských funkcí. Veškerý ostatní kód aplikací, který je prováděn před a po spuštění uživatelských funkcí je identický. Opakování kódu není efektivní a vyžaduje nadbytečný čas vynaložený při vývoji nových uživatelských funkcí. Bylo byl tedy vhodné vytvořit prostředí, ve kterém by mohly být uživatelské funkce spouštěny, a které by implementovalo sdílenou funkcionalitu. Návrh takovéto aplikace je zobrazen na následujícím obrázku, viz Obrázek 4.

Obrázek 4 - Průběh aplikace pro spouštění uživatelských funkcí
Zdroj: Autor, vlastní zpracování

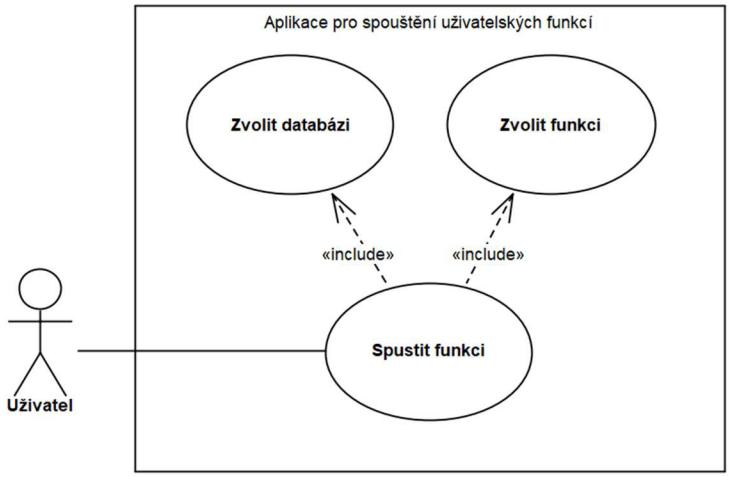
Výslednou aplikací bude desktopová aplikace umožňující spouštění uživatelských funkcí dle výběru uživatele aplikace. Uživatel bude moci uživatelské funkce do aplikace vkládat v podobě dynamicky připojovaných knihoven umístěných do vybraného adresáře systému. Pro dynamické načítání uživatelských funkcí bude využívána schopnost reflexe[1] jazyka C#. Konkrétně tedy prostředek *Assembly.LoadFrom()*. Praktický vzor použití tohoto prostředku uvádí [14, s. 160].

„Tento vzor byste mohli použít k vývoji aplikace, která bude podporovat doplňky jiných tvůrců. Každý doplněk bude muset implementovat stejné rozhraní a bude uložen ve své vlastní sestavě. Vaše aplikace pak může použít Assembly.LoadFrom() a Activator.CreateInstance() k aktivování doplňkového modulu.“ [14, s. 160]

Aplikace bude umožňovat připojení primárně k databázi typu Microsoft Access 2000 [4, s. 26]. Dále budou v aplikaci implementovány potřebné mechanismy za účelem předcházet poškození či zničení databáze při provádění operací z uživatelských funkcí. Zároveň bude automaticky zajišťováno vytváření záloh databáze pro zachování možností obnovení původní verze pro případ poškození

[1] *„Prozkoumávání existujících typů prostřednictvím jejich metadat se označuje za reflexi a uskutečňuje se pomocí bohaté sady typů v oboru názvů System.Reflection. Je rovněž možné dynamicky vytvářet nové typy za běhu pomocí tříd v oboru názvů Systém.Reflection.Emit.“* [14, s. 156]

datového souboru. Jednoduchý diagram případů užití z pohledu uživatele zobrazuje následující obrázek.



Obrázek 5 - Diagram případů užití
Zdroj: Autor, vlastní zpracování

5.1 Funkční požadavky

V této sekci jsou detailněji popsány jednotlivé funkční požadavky, které jsou kladeny na implementovanou aplikaci. Jak uvádí [15], *„Funkčním požadavkem je formulace toho, co byl měl systém dělat – popisuje požadovanou funkci systému.“* [15, s. 80]. Níže uvedené požadavky vychází z již zmíněného návrhu aplikace, viz Obrázek 4, ve kterém jsou uvedeny všechny podstatné etapy průběhu spuštění uživatelské funkce v aplikaci.

5.1.1 Zvolení databáze

Pro vytvoření připojení k databázi je třeba definovat připojovací řetězec se správně vyplněnými připojovacími údaji. Struktura připojovacího řetězce, včetně definice datového souboru, uživatelského jména a hesla pro připojení k databázi Microsoft Access, byla již zmíněna v kapitole 4.3. Z tohoto důvodu je třeba, aby aplikace umožňovala zadání potřebných údajů pro připojení k databázi. Tyto údaje jsou cesta k datovému souboru a přihlašovací údaje pro připojení.

5.1.2 Načtení funkcí

Uživatelské funkce budou obvykle vytvářeny tzv. „na míru“, dle aktuálních požadavků konkrétního uživatele. Jak již bylo zmíněno, z tohoto důvodu nebudou pevně zakomponovány do aplikace. Namísto toho budou existovat jako samostatný celek v podobě dynamicky načítaných knihoven, které budou do aplikace dynamicky načítány. Aplikace bude schopna jednotlivé funkce dynamicky načítat z definované složky a umožňovat uživateli jejich výběr.

5.1.3 Zobrazení uživatelského rozhraní funkce

Jednotlivé funkce budou volitelně obsahovat jednoduché uživatelské rozhraní pro zadání případných parametrů. Může se jednat o konkrétní hodnoty parametrů či například o výběr souborů, které mají být uživatelskou funkcí zpracovány. Aplikace bude schopna toto uživatelské rozhraní funkce zobrazit ve vyčleněném prostoru v rámci svého uživatelského rozhraní.

5.1.4 Zálohování databáze

Zálohování databáze je nejjednodušší a zároveň nejefektivnější způsob, jak předcházet případnému poškození či zničení databáze. Zálohování bude ve výchozím nastavení z preventivních důvodů prováděno před každým spuštěním uživatelské funkce. Aplikace bude schopna vytvořit zálohu zvolené databáze do speciálně vyhrazené složky. Funkcionalita zálohování bude řešena na úrovni aplikace tak, aby uživatelské funkci nemusely být přímo zpřístupněny žádné informace o databázi a jejím uložení.

5.1.5 Připojení k databázi

Přihlašovací údaje do databáze a cesta k databázovému souboru jsou citlivé informace. Bylo by tedy vhodné, aby tyto údaje zůstaly skryty v rámci aplikace a jednotlivé uživatelské funkce k nim neměly přístup. Aplikace bude spravovat připojení k databázi. Komunikace s databází bude uživatelským funkcím zprostředkována prostřednictvím veřejného rozhraní, které bude obsahovat metody pro komunikaci. Nebude však zpřístupňovat žádné informace o databázi.

Aplikace bude schopna vytvořit a řídit připojení k databázi, aniž by poskytla uživatelským funkcím citlivé údaje o připojené databázi.

5.1.6 Transakční zpracování

Pro zajištění konzistence dat v databázi v případě výskytu chyby v uživatelských funkcích lze v databázi využít transakčního zpracování. Aby se předešlo poškození databáze je nanejvýš vhodné této možnosti využít. Komunikace v rámci jedné funkce by tak by měla probíhat jako jednolitý celek. V případě chyby by mělo být vše navráceno do konzistentního stavu. Aplikace bude schopna zajišťovat řízení transakcí v rámci správy připojení k databázi, aniž by poskytla uživatelským funkcím citlivé údaje o připojené databázi.

5.1.7 Zaznamenání průběhu funkce

Uživatelské funkce budou prostřednictvím zpráv informovat uživatele o svém průběhu. Tyto zprávy budou uživateli zobrazeny v rámci okna průběhu funkce. Pro pozdější kontrolu a zpětné dohledání případných vzniklých chyb by bylo vhodné tento výstup ukládat a uchovávat v čitelné podobě. Aplikace bude schopna ukládat výstupy z průběhu funkcí do specifikované složky systému ve formě textového souboru.

5.2 Nefunkční požadavky

V této části jsou popsány nefunkční požadavky, které jsou kladeny na implementovanou aplikaci. Jak uvádí [15], *„Nefunkční požadavek je omezující podmínka uvalená na daný systém“* [15, s. 80]. Implementovaná aplikace bude sloužit jako doplňková k aplikaci jiné. Z tohoto důvodu je vhodné, aby technologie a závislosti použité v implementované aplikaci byly shodné nebo minimálně odvozené s technologiemi a závislostmi aplikace hlavní. Implementovaná aplikace tak bude své požadavky zakládat na požadavcích systému POHODA.

5.2.1 Operační systém

Jak je uvedeno v kapitole 2.5 a [2](viii), systém POHODA lze instalovat pouze na zařízeních s operačním systémem Windows. Lze tedy předpokládat, že

implementovaná aplikace bude taktéž využívána pouze na operačním systému Windows. Z tohoto důvodu nemusí být nuceně zajišťována podpora dalších operačních systémů. Podpora dalších operačních systémů není na závadu, nicméně je v rámci využití aplikace bezpředmětná a nebude jí věnována žádná pozornost.

5.2.2 Přístup do databáze

Pro přístup do databáze bude implementovaná aplikace využívat ovladače, instalované v rámci komponenty Microsoft Access Database Engine [9], konkrétně tedy ovladače ODBC, viz kapitola 2.6.1. Předpokládá se, že tato komponenta je na cílovém zařízení již instalována v rámci instalace systému POHODA. Aplikace bude pro připojení k databázi využívat některou z dostupných knihoven, která podporuje připojení a následnou komunikaci s databází Microsoft Access 2000 [4, s. 26].

5.2.3 Použité technologie

Implementovaná aplikace bude napsána v programovacím jazyce C#. Jedná se o moderní, pokročilý, typově zabezpečený jazyk s podporou objektově orientovaného přístupu [14, s. 5-6]. Jak uvádí [14], *„Jako každý moderní objektově orientovaný jazyk podporuje i C# koncepty, jakými jsou dědičnost, zapouzdření, mnohotvárnost a programování s využitím rozhraní.“* [14, s. 5]. Tento jazyk bude využíván společně se softwarovým frameworkem .NET 6. V tomto případě je velmi důležitá podpora reflexe [14, s. 156], na které je založena myšlenka dynamického načítání uživatelských funkcí, viz kapitola 5. Pro vývoj formulářů aplikace bude dále použita knihovna Windows Forms.

5.3 Grafické uživatelské rozhraní

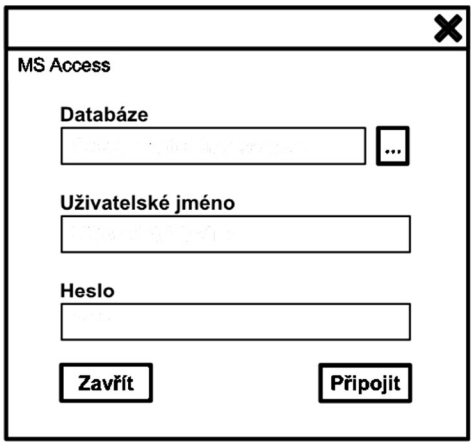
V následujících několika podkapitolách je představen návrh jednotlivých formulářů uživatelského rozhraní implementované aplikace. Vzhled těchto formulářů se bude odvíjet od vzhledu dle použité knihovny Windows Forms, viz kapitola 5.2.3. Jelikož hlavním cílem aplikace není poskytovat dokonalé a bohaté uživatelské rozhraní, ale provádět potřebnou funkcionalitu, budou prvky aplikace omezeny pouze na funkční minimum. Pro jednoduchost jsou navrhované formuláře představeny pouze

v podobě drátových modelů. Formuláře jsou představeny v pořadí, v jakém se s nimi může uživatel setkat při práci s aplikací.

5.3.1 Okno připojení

Jedním z funkčních požadavků na aplikaci je možnost zvolení databáze k připojení a specifikaci přihlašovacích údajů, viz kapitola 5.1.1. Tento požadavek je řešen prostřednictvím formuláře, ve kterém bude možno všechny tyty informace specifikovat. Formulář bude následně zobrazován v rámci formulářového okna pro připojení, ve které se může nacházet více formulářů pro více různých způsobů připojení. Toto okno bude zobrazováno formou dialogového formuláře[1]. Základní a jediný formulář, vytvořený v rámci této implementace, je určen k připojení k databázi Microsoft Access 2000 [4, s. 26].

Tento formulář bude obsahovat následující pole ve vertikálním pořadí, jak jsou zmíněna: textové pole pro zadání cesty k datovému souboru, textové pole pro zadání uživatelského jména a textové pole pro zadání hesla. Rozšířením formuláře bude tlačítko umožňující výběr datového souboru prostřednictvím systémového průzkumníka souborů.



Obrázek 6 - Návrh okna připojení
Zdroj: Autor, vlastní zpracování

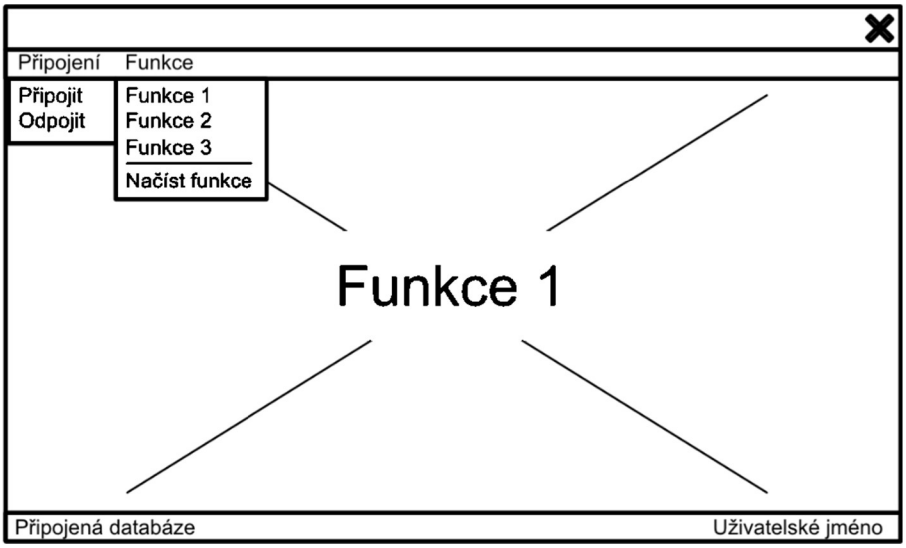
[1] *„Dialogový formulář je podobný oknu se zprávou třídy MessageBox. Oba tyto formuláře například zobrazují jiný formulář a jsou modální; to znamená, že se uživatel nemůže vrátit do hlavního formuláře, dokud neukončí dialogový formulář klepnutím na některé z jeho tlačítek.“* [16, s. 187]

Ve spodní části okna pod zobrazeným formulářem se budou nacházet dvě tlačítka pro potvrzení a zrušení akce. Tlačítko „Zrušit“, na levé straně, bude sloužit ke zrušení výběru databáze a uzavření okna. Tlačítko „Připojit“, na pravé straně okna, bude potvrzovat zvolení databáze.

5.3.2 Hlavní okno

Hlavní formulářové okno aplikace bude obsahovat několik komponent. S tímto oknem bude uživatel aplikace pracovat nejčastěji, jelikož představuje výchozí bod pro veškeré funkcionality aplikace. V úrovni spodního okraje okna se bude nacházet lišta, ve které budou zobrazeny některé informace o připojené databázi. Podél horního okraje bude umístěno hlavní menu aplikace. Součástí tohoto menu budou dvě záložky, v nichž bude možno spouštět určité akce.

Bude se jednat o záložku „Připojení“, ve které budou dostupná tlačítka pro připojení a odpojení databáze. Díky těmto tlačítkům bude možno měnit připojenou databázi za běhu aplikace bez nutnosti jejího restartu. Ve druhé záložce nazvané „Funkce“ budou vypsány všechny načtené uživatelské funkce. Odděleně zde bude také dostupné tlačítko pro načtení funkcí, což je jeden z funkčních požadavků aplikace, viz kapitola 5.1.2. Díky tomu bude možné přidávat a znovu načítat funkce i za běhu aplikace.

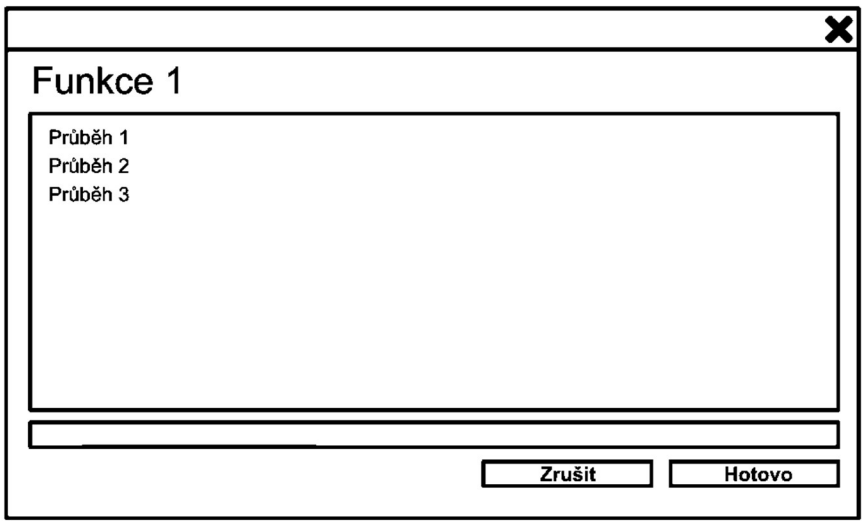


Obrázek 7 - Návrh hlavního okna
Zdroj: Autor, vlastní zpracování

Hlavní část okna bude vyhrazena jako prostor pro zobrazení formuláře zvolené uživatelské funkce. Tím je vyhověno dalšímu z funkčních požadavků na implementovanou aplikaci, viz kapitola 5.1.3. Formulář funkce bude zobrazen po zvolení funkce v hlavním menu okna.

5.3.3 Okno průběhu funkce

Dalším z funkčních požadavků na aplikaci je zaznamenání průběhu spuštěné uživatelské funkce a jeho zobrazení uživateli, viz kapitola 5.1.7. Tento průběh bude uživateli zobrazen v rámci okna průběhu funkce. Pro plnou informovanost uživatele bude toho okno obsahovat název spuštěné funkce a needitovatelné pole pro zobrazení informací o průběhu. Pod polem, ve kterém bude zobrazen průběh, bude umístěna animovaná průběhová lišta, která bude uživateli signalizovat průběh funkce. V případě probíhající funkce bude tato lišta zobrazovat animaci průběhu. Po dokončení funkce bude její animace zastavena.



Obrázek 8 - Návrh okna průběhu funkce
Zdroj: Autor, vlastní zpracování

Okno bude dále obsahovat dvě tlačítka při pravém spodním okraji. Tlačítko s popisem „Zrušit“ bude sloužit ke zrušení průběhu funkce a bude dostupné od zahájení funkce po její ukončení. Po skončení funkce bude zneaktivněno. Druhé tlačítko s popisem „Hotovo“ bude sloužit k uzavření formulářového okna. Toto tlačítko bude neaktivní od zahájení funkce. Po skončení bude aktivováno. Uzavření

formulářového okna bude v případě probíhající funkce blokováno. Nemělo by tak dojít k uzavření okna průběhu, aniž by byla funkce ukončena, ať již úspěšně či s neúspěšně.

Jednotlivé zprávy z průběhu funkce budou vypisovány do needitovatelného pole, a to vždy na nový řádek. Zprávy budou řazeny od nestarší zprávy nahoře po nejnovější zprávu dole. Ve chvíli, kdy bude přidána nová zpráva na nový řádek, bude obsah pole zarovnán tak, aby poslední zpráva byla vždy viditelná.

6 Implementace aplikace

V následující části práce je velmi zkráceně popsána implementace aplikace pro spouštění uživatelských funkcí. Implementace aplikace vychází z požadavků, viz kapitola 5.1 a kapitola 5.2 a návrhů, které byly provedeny v kapitole 5.3. Jednotlivé následující kapitoly popisují projekty a jejich části. Zároveň uvádí ukázky kódu některých komponent.

Vzhledem k tomu, že uživatelské funkce budou vytvářeny odděleně od samotné aplikace a jejich načítání do aplikace bude, jak již bylo zmíněno, probíhat prostřednictvím reflexe [14, s. 160] je třeba, aby bylo definováno sdílené rozhraní, se kterým bude moci aplikace pracovat, a ze kterého budou jednotlivé uživatelské funkce vycházet. Řešení aplikace je tedy rozděleno do dvou projektů, a to na projekt sdíleného kódu a na projekt kódu aplikace. Projekt sdíleného kódu je jednou ze závislostí projektu aplikace.

6.1 Projekt sdíleného kódu

V rámci tohoto projektu jsou definovány veškeré funkcionality, které jsou sdíleny mezi implementovanou aplikací pro spouštění uživatelských funkcí a jednotlivými uživatelskými funkcemi. Výstupem tohoto projektu je knihovna, na kterou je následně v projektech funkcí a aplikace odkazováno. Veškerá funkcionalita v tomto projektu je implementována v podobě rozhraní či abstraktních tříd. Jak uvádí [14], *„Rozhraní poskytuje specifikaci a nikoli implementaci svých členů.“* [14, s. 60]. Konkrétní implementace pak náleží do odvozených projektů.

6.1.1 Rozhraní IDatabase

K tomu, aby uživatelské funkce mohly vykonávat svoji funkcionalitu je třeba, aby byly schopny komunikovat s databází, která je připojena k aplikaci. Jak je uvedeno ve funkčních požadavcích, viz kapitola 5.1.5, je vhodné, aby funkce neměly od aplikace přímý přístup k údajům o připojené databázi. Aby bylo vyhověno těmto požadavkům je ve sdíleném projektu definováno veřejné rozhraní *IDatabase*, které definuje veřejné členy pro komunikaci s databází. Kód tohoto rozhraní je uveden na následující ukázce kódu, viz Ukázka kódu 1.

Rozhraní *IDatabase* předepisuje dvě události, konkrétně událost *SysMessageReported* a *UsrMessageReported*, které slouží k ohlašování systémových a uživatelských oznámení objektu. Prostřednictvím těchto událostí jsou zasílány textové zprávy o průběhu funkce. Tato funkcionalita je dále využívána při zaznamenávání průběhu funkce, viz funkční požadavek v kapitole 5.1.7, kdy objekt databáze informuje o provedených příkazech. Rozlišení uživatelských a systémových oznámení je z důvodu odlišení systémových a uživatelských zpráv ve výsledném záznamu průběhu funkce.

```
public interface IDatabase : IDisposable
{
    /// Occurs when system message is reported.
    event ReportEventHandler? SysMessageReported;
    /// Occurs when user message is reported.
    event ReportEventHandler? UsrMessageReported;

    /// Runs command in database.
    int Execute(string cmd, params object[] parameters);
    /// Selects data from database.
    DataTable Select(string cmd, params object[] parameters);
    /// Selects one item from database.
    T SelectItem<T>(string cmd, T defVal, params object[] parameters);
    /// Selects list from database.
    List<T> SelectList<T>(string cmd, T defVal, params object[] parameters);
}
```

Ukázka kódu 1 - Rozhraní IDatabase
Zdroj: Autor, vlastní zpracování

Dále rozhraní definuje dvě veřejné metody, prostřednictvím kterých je možno obsloužit veškeré příkazy v jazyce SQL. Pro příkazy typu SELECT, tedy pro příkazy pro výběr dat z databáze, je určena metoda *Select*. Tato metoda na základě poskytnutého příkazu a případných parametrů vykoná příkaz a navrátí požadované hodnoty, pokud byly nalezeny. Pro příkazy ostatních typů je určena druhá metody *Execute*. Pomocí této metody je vykonán daný příkaz s použitím případných parametrů, který navrátí pouze informaci o počtu ovlivněných řádků.

Rozhraní dále obsahuje další dvě veřejné metody, které jsou modifikací metody *Select*. V případě metody *SelectItem* je navrácena pouze jedna jediná hodnota požadovaného datového typu. V případě problémů je navrácena výchozí hodnota. Metoda *SelectList* navrací list hodnot požadovaného typu.

Možnost volitelného předání parametrů jednotlivým metodám umožňuje bezpečné vložení parametrů do některých klauzulí příkazů jazyka SQL. Například příkaz SELECT obsahuje několik klauzulí, mezi kterými je i klauzule WHERE [13, s. 71], která slouží k filtrování výsledků. Hodnoty tohoto filtru tak mohou být v příkazu zastoupeny speciálním znakem a do dotazu vloženy dodatečně pomocí parametrů.

6.1.2 Rozhraní IUDFunction

Veřejný vzhled uživatelských funkcí je definován rozhraním *IUDFunction*, viz Ukázka kódu 2. Každá z uživatelských funkcí, která bude načítána do aplikace, musí závazně toto rozhraní implementovat. Opět jsou definovány dvě události pro ohlašování systémových a uživatelských oznámení objektu, a to z důvodu plnění funkčního požadavku, viz kapitola 5.1.7. Dalšími údaji jsou jméno uživatelské funkce a nastavení, zda je pro funkci vyžadováno transakční zpracování a vytvoření zálohy. Tyto vlastnosti jsou přítomny na základě funkčních požadavků z kapitol 5.1.4 a 5.1.6. Hlavním prvkem je ovšem vlastnost *Database*, která je nastavována aplikací při spuštění funkce, a která umožňuje funkci komunikovat s databází dle funkčního požadavku v kapitole 5.1.5. Jedinou metodou tohoto rozhraní je metoda *Run*, prostřednictvím které je funkce spouštěna.

```
public interface IUDFunction
{
    /// Occurs when system message is reported.
    event ReportEventHandler? SysMessageReported;
    /// Occurs when user message is reported.
    event ReportEventHandler? UsrMessageReported;

    /// Name of user defined function.
    string FunctionName { get; }
    /// Indicates if transaction should be used.
    bool UseTransaction { get; }
    /// Indicates if backup should be created.
    bool CreateBackup { get; }
    /// Indicates if cancel is pending.
    bool IsCancelPending { get; set; }
    /// Connected database.
    IDatabase Database { get; set; }

    /// Runs function.
    void Run();
}
```

Ukázka kódu 2 - Rozhraní IUDFunction
Zdroj: Autor, vlastní zpracování

Pro snadnější implementaci uživatelských funkcí je ve sdíleném projektu dále vytvořena abstraktní třída *UDFunction*. Tato třída implementuje rozhraní *IUDFunction*. Dále zavádí abstraktní startovací metodu *Start*, metodu pro ohlašování uživatelských oznámení *Report* a metodu pro přerušení funkce *CanEnd*. V případě implementace uživatelské funkce, která je odvozena od abstraktní třídy *UDFunction*, lze tyto prvky využít.

Implementované uživatelské funkce tak lze, pro ušetření opakovaného kódu, odvozovat od této abstraktní třídy. V takovém případě postačuje implementovat pouze tělo metody *Start*, která je předepsána abstraktní třídou.

6.1.3 Rozhraní IRunFunctionService

Pro spouštění funkcí je ve sdíleném projektu definováno veřejně dostupné rozhraní *IRunFunctionService*, viz Ukázka kódu 3. Toto rozhraní definuje pouze jednu veřejnou metodu, kterou je metoda *Run*. Metoda přijímá jako parametr instanci třídy, která implementuje již zmíněné rozhraní *IUDFunction*. Cílem je na této instanci zavolat metodu *Run* pro spuštění funkce.

```
public interface IRunFunctionService
{
    /// Runs user defined function.
    void Run(IUDFunction function);
}
```

Ukázka kódu 3 - Rozhraní IRunFunctionService
Zdroj: Autor, vlastní zpracování

6.1.4 Rozhraní IFunctionControl

Jelikož uživatelské funkce obsahují formuláře a jsou v aplikaci reprezentovány vizuálně, je ve sdíleném projektu definováno veřejné rozhraní *IFunctionControl*, viz Ukázka kódu 4, které udává povinné prvky těchto komponent. Jedná se o vlastnost *ControlName*, která je v aplikaci použita v seznamu funkcí, a o metodu *Initialize*, pomocí které je v aplikaci každá instance uživatelské funkce inicializována. Tato metoda je používána při načítání funkcí pomocí reflexe [14, s. 160]. Při načítání funkcí z definovaného adresáře jsou pomocí reflexe vyhledávány prvky, které implementují právě toto rozhraní. Následně jsou

vytvořeny jejich instance, které jsou zobrazeny v aplikaci. Parametrem této metody je služba typu *IRunFunctionService*, prostřednictvím které je funkci podstoupena schopnost spouštění funkcí.

Neabstraktní implementaci tohoto rozhraní poskytuje třída *FunctionControl*, taktéž ze sdíleného projektu. Tuto třídu lze využívat jako rodičovskou při implementaci formuláře uživatelské funkce. Povinným parametrem konstruktoru této třídy je název uživatelské funkce. Tato třída není abstraktní, ačkoliv by se jednalo o vhodnější řešení. V případě nastavení definování funkce jako abstraktní docházelo k chybám v grafickém tvůrci formulářů při tvorbě formulářů, které byly od tohoto odvozeny. Editor nebyl schopen vytvořit instance abstraktní rodičovské třídy a jeho použití tak nebylo možné. Z tohoto důvodu byla zvolena možnost neabstraktní třídy.

```
public interface IFunctionControl
{
    /// Name of control.
    string ControlName { get; }

    /// Initializes function control with run function service.
    void Initialize(IRunFunctionService runFunctionService);
}
```

Ukázka kódu 4 - Rozhraní IFunctionControl
Zdroj: Autor, vlastní zpracování

6.2 Projekt aplikace

Tento projekt obsahuje kompletní kód aplikace, která umožňuje výběr databáze, vytvoření zálohy, načtení uživatelských funkcí a jejich spouštění. V rámci tohoto projektu jsou vytvořeny konkrétní implementace některých rozhraní a abstraktních tříd z projektu sdílené funkcionality. Z tohoto důvodu je projekt sdíleného kódu jednou ze závislostí tohoto projektu. Samotný projekt je z důvodu přehlednosti dále členěn do několika dalších adresářů. Výstupem projektu je spustitelná, desktopová formulářová aplikace, určená pro operační systém Windows.

6.2.1 Stores

V adresáři *Stores* jsou definována rozhraní, jejichž účelem je ukládat a spravovat informace potřebné pro správné fungování aplikace. Všechna tato rozhraní jsou

založena na podobném principu. Obsahují vlastnosti, ve kterých jsou požadované informace uloženy a funkcionalitu, jak těmto vlastnostem nastavovat nové hodnoty. Pokud je to vhodné obsahují také událost, prostřednictvím které informují okolí o změně informací ve svých vlastnostech. Tyto „sklady“ jsou vytvořeny v rámci startovací třídy a odkaz na ně je poskytnut všem prvkům, které ho vyžadují. Díky tomu mohou různé části aplikace přistupovat k těmto datům. Pro každé rozhraní existuje v adresáři alespoň jedna konkrétní implementace. Přehled prvků v adresáři zobrazuje následující tabulka, viz Tabulka 9.

Tabulka 9 - Přehled prvků adresáře Stores

IConnectionStore	Definice veřejného rozhraní úložiště připojovacích údajů.
ConnectionStore	Základní implementace úložiště připojovacích údajů.
IFunctionsStore	Definice veřejného rozhraní úložiště uživatelských funkcí.
FunctionStore	Základní implementace úložiště připojovacích údajů.

Zdroj: Autor, vlastní zpracování

6.2.2 Database

V adresáři *Database* jsou implementovány prvky pro komunikaci s databází, které rozšiřují veřejné rozhraní *IDatabase* z projektu sdíleného kódu, viz kapitola 6.1.1. Z důvodu různých databázových systémů je odděleně definováno rozhraní *ITransactional*, které představuje databázi podporující transakční zpracování, a rozhraní *IBackupable*, představující databázi podporující zálohování.

Jako výchozí bod pro implementaci konkrétních databázových tříd je v adresáři *Database* definována abstraktní třída *TransactionalDatabase*. Tato třída implementuje rozhraní *IDatabase* a *ITransactional* a zároveň definuje jejich základní funkcionalitu.

Z abstraktní třídy *TransactionalDatabase* vychází konkrétní implementace v podobě třídy *MsAccessMdbDatabase*. Tato třída představuje implementaci databázové třídy specificky určené pro databázi Microsoft Access 2000 [4, s. 26]. Třída taktéž začleňuje rozhraní *IBackupable* pro podporu zálohování databáze.

6.2.3 Services

Činnosti, které aplikace umí vykonávat, jsou implementovány pomocí funkcionalit v adresáři *Services*. Tento adresář je dále rozčleněn dle zaměření služby, kterou obsažená funkcionalita obsahuje. Převážná část těchto funkcionalit využívá úložišť typu *Store,* zmíněných v předchozí kapitolách. Obdobně jako úložiště, jsou i tyto služby vytvořeny v rámci startovací třídy a následně jsou poskytnuty ostatním prvkům aplikace, které je vyžadují. Jednotlivé služby včetně rozhraní uvádí následující tabulka.

Tabulka 10 - Přehled prvků adresáře Services

ISetConnectionService	Definice veřejného rozhraní služby pro nastavení připojení.
FormSetConnectionService	Služba pro nastavení připojení prostřednictvím připojovacího okna.
IClearBackupFilesService	Definice veřejného rozhraní služby pro vyčištění zálohových souborů.
ClearBackupFilesService	Základní implementace služby pro vyčištění zálohových souborů z adresáře.
ILoadFunctionsService	Definice veřejného rozhraní služby pro načtení uživatelských funkcí.
DirLoadFunctionService	Služba pro načtení uživatelských funkcí z adresáře.
RunFunctionService	Základní implementace služby pro spouštění uživatelských funkcí, viz kapitola 6.1.3.
ILogService	Definice veřejného rozhraní služby pro zaznamenávání průběhu funkce.
FileLogService	Služba pro zaznamenávání průběhu funkce do souboru v adresáři.

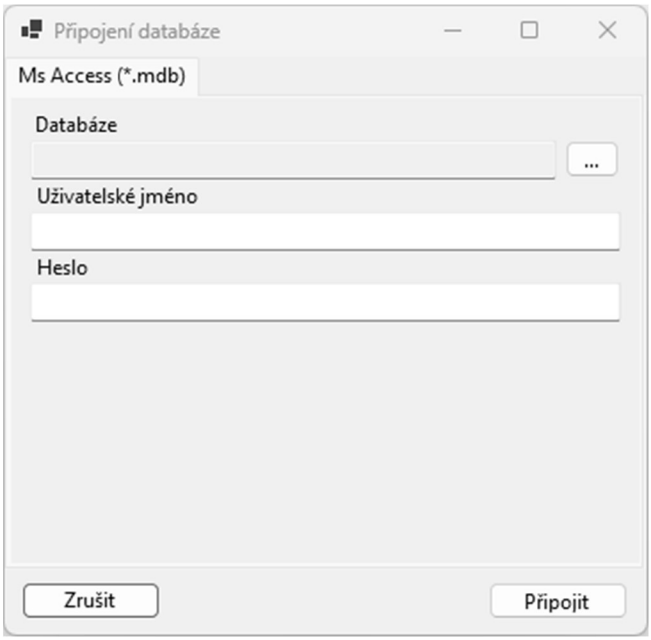
Zdroj: Autor, vlastní zpracování

6.2.4 Forms a Controls

Obsahem adresáře *Forms* jsou jednotlivá formulářová okna aplikace. Tato okna jsou vytvořena na základě návrhů z kapitoly 5.3 a obsahují veškerou potřebnou funkcionalitu pro správné fungování aplikace. Na základě návrhu také odpovídají funkčním požadavkům, které byly stanoveny v kapitole 5.1. Některá z formulářových oken také využívají uživatelských komponent, které se nacházejí v adresáři *Controls*. Tyto komponenty sdružují několik dalších prvků do souvislého celku.

Okno připojení obsahuje všechny prvky potřebné pro nastavení připojení k databázi Microsoft Access 2000 [4, s. 26], uvedené v rámci návrhu okna v kapitole 5.3.1. Uživateli je umožněn výběr datového souboru prostřednictvím průzkumníka souborů, zadání přihlašovacího jména a hesla. Editace pole, které zobrazuje cestu k datovému souboru není povolena. Pro výběr datového souboru je vždy třeba využít přilehlého tlačítka, které otevře dialogové okno pro výběr souboru. V tomto okně je aktivován filtr, který omezuje zobrazené soubory pouze na ty s příponou *.mdb. Pole pro zadání hesla zobrazuje namísto zadaných znaků hesla speciální zástupný znak.

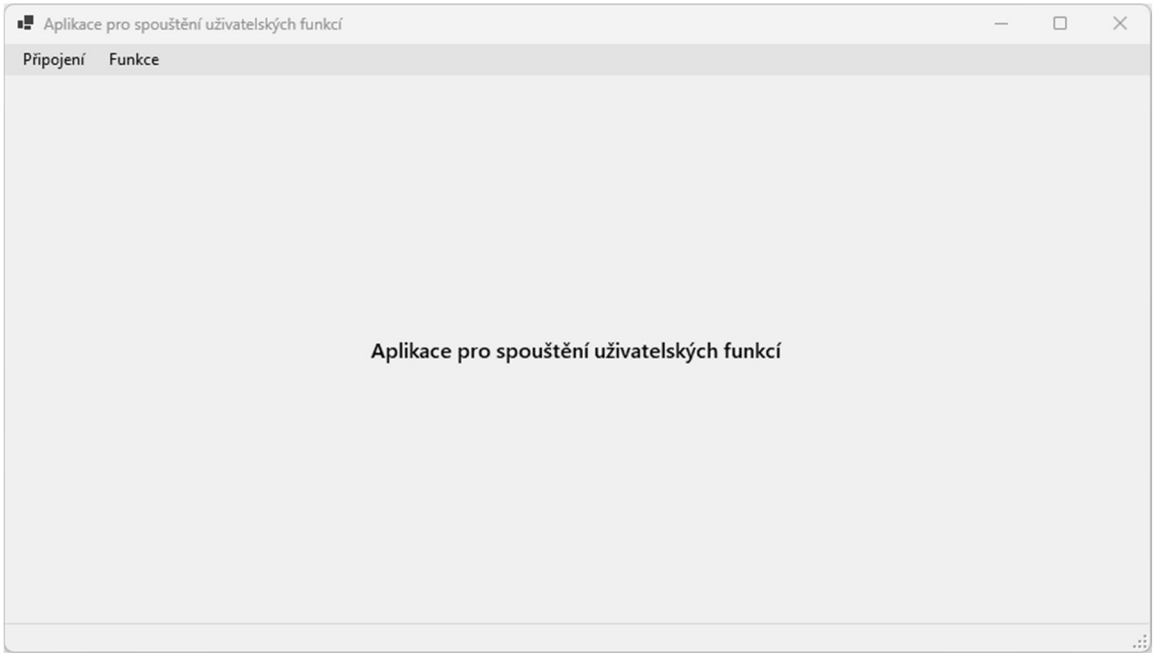
Údaje vložené do okna jsou aplikací dále zpracovány a v případně využity. Z tohoto důvodu je okno zobrazováno formou dialogového formuláře a vyčkává na akci uživatele. Po akci uživatele lze okno odbavit dvěma způsoby. Prvním z nich je zadání validních údajů a jejich potvrzení tlačítkem „Připojit“. Druhým způsobem je zrušení akce pomocí tlačítka „Zrušit“. Tyto různé způsoby odbavení okna jsou rozlišeny prostřednictvím vlastnosti *DialogResult*, která může nabývat jedné z hodnot stejnojmenného výčtového typu [16, s. 188]. Vzhled výsledného okna je zobrazen na následujícím obrázku, viz Obrázek 9.



Obrázek 9 - Okno připojení
Zdroj: Autor, vlastní zpracování

Po úspěšném zadání validních údajů a potvrzení jsou zadané údaje zapsány do některého z uvedených úložišť, připojovací okno je zavřeno a uživateli je zpřístupněno hlavní okno aplikace.

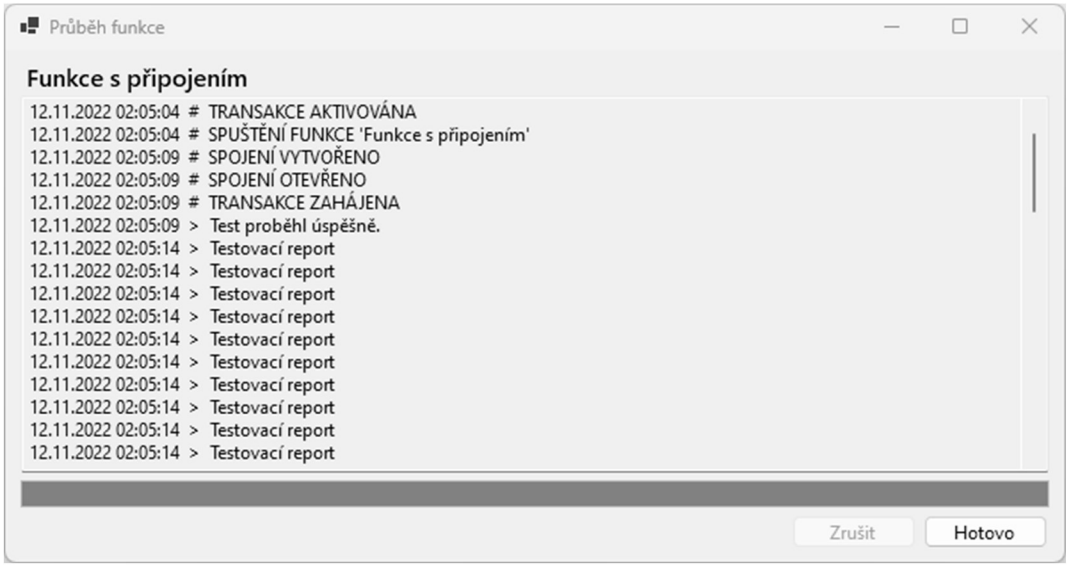
Hlavní okno aplikace poskytuje přístup k funkcionalitám aplikace prostřednictvím hlavního menu, které se táhne podél celého horního okraje okna. Dle návrhu z kapitoly 5.3.2, toto okno obsahuje záložku pro správu akcí pro připojení databáze a záložku pro správu uživatelských funkcí. Podél spodního okraje okna se táhne informační lišta. V případě připojení databáze jsou v ní zobrazeny informace o připojeném datovém zdroji, přístupové uživatelské jméno a ovladač zajišťující připojení. Hlavní středovou část okna zabírá prostor pro zobrazení formulářů uživatelských funkcí. Obrázek 10 zobrazuje vzhled hlavního okna.



Obrázek 10 - Hlavní okno
Zdroj: Autor, vlastní zpracování

Při zvolení databáze je v hlavním okně zobrazen formulář funkce, ve kterém může uživatel specifikovat případné parametry potřebné pro průběh funkce. Některý z prvků zobrazeného formuláře bude také sloužit k samotnému spuštění funkce, které proběhne prostřednictvím služby pro spouštění funkcí. Tato služba je funkcím podstoupena při jejich načtení. V takovém případě je funkce spuštěna s využitím okna průběhu funkce.

Okno průběhu funkce, dle návrhu v kapitole 5.3.3, obsahuje název spuštěné funkce a prostor pro zobrazení zpráv o průběhu. Ve spodní části formulářového okna se nachází animovaná průběhová lišta. Dále se zde nachází tlačítka pro zrušení funkce a zavření okna, jejichž aktivita je řízena dle stavu provádění funkce. Okno je vytvořeno zcela dle návrhu a plní tak funkční požadavky z kapitoly 5.1.7. Vzhled okna průběhu je zobrazen na následujícím obrázku, viz Obrázek 8.



Obrázek 11 - Okno průběhu funkce
Zdroj: Autor, vlastní zpracování

6.3 Spuštění aplikace

Celá aplikace je spouštěna prostřednictvím třídy *Bootstrapper*. V této třídě dochází k načtení konfigurací a vytvoření všech potřebných prvků aplikace. Jednotlivé objekty jsou vytvořeny v takovém pořadí, aby objektům se závislostmi na jiné mohly tyto objekty být poskytnuty. Nejprve dochází k načtení konfigurací a s jejich využitím k vytvoření objektů úložišť, které jsou zmíněny v kapitole 6.2.1. Dále jsou inicializovány objekty služeb, viz kapitola 6.2.3, kterým jsou, dle potřeby, podstoupena vybraná úložiště. Jako poslední jsou inicializována některá okna aplikace, kterým jsou poskytnuty potřebné služby.

Při spuštění aplikace je zobrazeno okno pro výběr databáze k připojení. Následně se kontrolují soubory záloh. Jelikož jsou soubory záloh ve výchozím nastavení vytvářeny při každém spuštění funkce, tak jejich množství ve složce pro zálohy může

narůst do vysokých počtů. Z tohoto důvodu jsou soubory záloh, které jsou starší než nastavený limit, při spuštění aplikace zobrazeny uživateli a je nabídnuto jejich automatické odstranění. Po zpracování souborů záloh, jsou načteny uživatelské funkce ze specifikovaného adresáře a je zobrazeno hlavní okno aplikace. V tomto okně již uživatel může vybírat z nabízených funkcí.

6.4 Implementace uživatelské funkce

Příklad implementace uživatelské funkce je uveden v projektu v podobě testovací funkce. Výstupem projektu je knihovna, která podporuje dynamické načítání. Pro načtení knihovny do aplikace jako uživatelské funkce je třeba, aby byla vstupní knihovna umístěna do správného adresáře aplikace. Knihovny uživatelských funkcí musí taktéž dodržovat jmennou konvenci, že název knihovny musí začínat textem *„NazevAplikace.Functions.“*. Podmínka na název souborů je aplikována při hledání funkcí pro načtení.

Díky předpřipraveným abstraktním třídám, které jsou dostupné v projektu sdíleného kódu, je velmi jednoduché vytvořit uživatelskou funkci. Pro vytvoření funkce postačuje, aby knihovna obsahovala třídu, která bude implementovat rozhraní *IFunctionControl*, ve které je formulář funkce. Kód vzorové třídy je uveden v následující ukázce kódu, viz Ukázka kódu 5. V tomto případě formulář obsahuje pouze jednoduché tlačítko, po jehož stisknutí je vytvořena instance testovací funkce, která je spuštěna prostřednictvím služby pro spouštění funkcí, viz kapitola 6.1.3.

```
public partial class TestFunctionControl : FunctionControl
{
    public TestFunctionControl() : base("Testovací funkce")
    {
        InitializeComponent();
    }

    private void RunFunctionButton_Click(object sender, EventArgs e)
    {
        TestFunction testFunction = new()
        {
            FunctionName = "Testovací funkce"
        };
        RunFunctionService.Run(testFunction);
    }
}
```

Ukázka kódu 5 – Příklad formuláře testovací funkce
Zdroj: Autor, vlastní zpracování

Dále by knihovna měla obsahovat třídu, která vychází z rozhraní *IUDFunction*, a která obsahuje kód samotné funkce. Díky využití abstraktní třídy *UDFunction* postačuje pro vytvoření funkce definovat pouze tělo metody *Start*. Ve funkci lze využívat atributy a metody, které jsou dostupné v rámci objektu funkce. Metoda *CanEnd* slouží k označení místa, ve kterém je možno funkci přerušit. Tato metoda vnitřně provádí kontrolu, zda uživatel nevyžadoval ukončení funkce. Pokud ano, je vyvolána výjimka a funkce je přerušena. Pokud uživatel ukončení funkce nevyžadoval, funkce pokračuje bez přerušení. Vlastnost *Database* poskytuje metody pro komunikaci s databází. Funkce tak nemá přímý přístup k údajům o připojené databázi. Pomocí metody *Report* může tvůrce funkce vypisovat vlastní zprávy o průběhu funkce. Příklad třídy funkce, včetně použití zmíněných metod, zobrazuje následující Ukázka kódu 6.

```
internal class ConnectFunction : UDFunction
{
    public override void Start()
    {
        CanEnd();
        int result = Database.SelectItem("SELECT 1", 0);
        if (result == 1) Report("Test proběhl úspěšně.");
        else Report("Test proběhl neúspěšně.");
    }
}
```

Ukázka kódu 6 - Příklad testovací funkce
Zdroj: Autor, vlastní zpracování

7 Shrnutí výsledků

V rámci praktické části této práce byla implementována desktopová aplikace umožňující připojení databáze Microsoft Access 2000, kterou využívá systém POHODA [4, s. 26], a následné spouštění uživatelských funkcí, které ve svém průběhu využívají data z připojené databáze. Jednotlivé uživatelské funkce nejsou pevnou součástí implementované aplikace. Jejich načítání do aplikace probíhá dynamicky za běhu aplikace z cesty předem specifikovaného adresáře. K tomu je využita schopnost reflexe jazyka C# [14, s. 160]. Aplikace pro načtené funkce zajišťuje připojení k databázi, její případné zálohování, zaznamenávání průběhu vykonaných akcí a zobrazení jejich formulářů. Zároveň řídí vytváření a ukončování transakcí, které jsou ve spojení s databází využity, a obsluhuje vykonávání příkazů. Načtené funkce jsou tak odstíněny od všech informací o databázi, včetně přístupových údajů, což zvyšuje bezpečnost při implementaci funkcí.

V případě implementace nové uživatelské funkce je naprosto dostačující vytvoření formuláře této funkce a třídy s kódem samotné funkce, viz kapitola 6.4. Není třeba opakovaně implementovat funkcionality pro komunikaci s databází, pro zálohování, ani pro zaznamenávání průběhu funkce. Díky tomu dochází ke snížení času, který je potřebný k implementaci nové uživatelské funkce a jejímu odladění. Pomocí dynamického načítání funkcí je možné uživatelské funkce v aplikaci libovolně kombinovat. Různí uživatelé tak mohou mít různé kombinace dostupných funkcí, které budou zaměřeny na jejich specifické požadavky.

Pro ověření funkčnosti aplikace, byla aplikace spuštěna a připojena k testovací databázi. Pro testování byla použita testovací funkce, která byla umístěna do adresáře, ze kterého byla úspěšně načtena do hlavního okna aplikace. Zobrazení formuláře funkce proběhlo v pořádku. V aplikaci byla uživatelská funkce následně spuštěna. Došlo k úspěšnému vytvoření zálohy testovací databáze, úspěšnému provedení uživatelské funkce a zaznamenání průběhu do souboru ve specifikovaném adresáři.

8 Závěry a doporučení

V této práci byla implementována aplikace pro spouštění uživatelských funkcí v ekonomickém systému Stormware POHODA, jejímž účelem bylo prozkoumat a ověřit možnosti spouštění uživatelských funkcí, které by využívaly přímý přístup do databáze systému. Výsledkem práce je prozkoumání možnosti přístupu do databáze Microsoft Access 2000 [4, s. 26], návrh a implementace aplikace. Spouštění uživatelských funkcí bylo v implementované aplikaci úspěšně otestováno. Stanovené cíle práce tímto byly splněny.

V teoretické části byl představen ekonomický systém Stormware POHODA a jeho použití při zpracování účetnictví. Zároveň byly představeny možné způsoby komunikace se systémem POHODA, včetně prozkoumání možnosti přímého přístupu do databáze systému.

Praktická část představila návrh a samotnou implementaci externí aplikace pro spouštění uživatelských funkcí. Aplikace je vytvořena pomocí jazyka C# a je určena pouze pro operační systém Windows. Implementovaná aplikace se snaží zajistit maximální bezpečnost prováděných akcí v tom smyslu, aby nedošlo k poškození připojené databáze, případně aby bylo možno vzniklé chyby dohledat pomocí záznamů průběhu funkcí či poškozenou databázi obnovit ze zálohy. Implementovaná aplikace využívá přímý přístup do databáze, ke kterému se vztahuje upozornění výrobce [4, s. 26].

Tato cesta spouštění uživatelských funkcí byla ověřena jako funkční a je tak na dalším výzkumu, zda by tato cesta měla být skutečné využívána v reálném prostředí a zda poskytuje dostatečnou úroveň bezpečnosti. V rámci případného dalšího vývoje by mohl být vylepšen vzhled grafického uživatelského rozhraní celé aplikace. Současná aplikace obsahuje plně funkční grafické uživatelské rozhraní, avšak vzhled některých částí je spíše strohý. Dalším směrem, ve kterém by případně bylo možno implementovanou aplikaci vylepšit, je rozšíření seznamu podporovaných databází, se kterými je aplikace schopna pracovat. Díky tomu by se aplikace mohla stát univerzálním nástrojem.

9 Seznam použité literatury

1. *Informace pro účetní a podnikatele - Portál POHODA* [online]. Jihlava: STORMWARE, c2022 [cit. 2022-04-15]. Dostupné z: https://portal.pohoda.cz/

2. *POHODA - ekonomický a informační systém* [online]. Jihlava: STORMWARE, c2022 [cit. 2022-04-15]. Dostupné z: https://www.stormware.cz/

3. TÜV SÜD CZECH S.R.O. ISO 9001: CO JE ISO 9001?. *TÜV SÜD Czech* [online]. Praha: TÜV SÜD Czech, 2021 [cit. 2022-04-16]. Dostupné z: https://www.tuvsud.com/cs-cz/cinnosti/audity-a-certifikace-systemu/iso-9001-certifikace-systemu-managementu-kvality

4. STORMWARE S.R.O. *Ekonomický systém POHODA: Příručka uživatele* [online]. Jihlava: STORMWARE, c2022, 504 s. [cit. 2023-02-18]. Dostupné z: https://www.stormware.cz/dnload/guide_POHODA_2022.pdf

5. SVOBODOVÁ, Libuše a Miloslava ČERNÁ. Accounting, Economic and ERP Systems on the Czech Scene. *Advanced Science Letters* [online]. 2016, 22(5), 1170-1174 [cit. 2022-04-29]. ISSN 1936-6612. Dostupné z: doi:10.1166/asl.2016.6683

6. SVOBODOVÁ, Libuše a Martina HEDVIČÁKOVÁ. Factors Determining Optimal Social Media Network Portfolio for Accounting Firms: The Case of the Czech Republic. In: AL-SHARHAN, Salah A., Antonis C. SIMINTIRAS, Yogesh K. DWIVEDI, et al., ed. *Challenges and Opportunities in the Digital Era* [online]. Cham: Springer International Publishing, 2018, 2018-10-12, s. 425-435 [cit. 2022-10-10]. Lecture Notes in Computer Science. ISBN 978-3-030-02130-6. Dostupné z: doi:10.1007/978-3-030-02131-3_38

7. KOBĚRSKÁ, Lucie. *Výběr softwaru pro vedení daňové evidence a účetnictví.* Ostrava, 2018. Dostupné také z: https://theses.cz/id/xs5gk8/. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava, Ekonomická fakulta.

8. OTTE, Lukáš. *DATABÁZOVÉ SYSTÉMY*. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2013, 189 s. ISBN 978-80-248-3054-4.

9. MICROSOFT. Microsoft Access Database Engine 2010 Redistributable. MICROSOFT. *Microsoft – Cloud, Computers, Apps & Gaming* [online]. Redmond: Microsoft, 2022 [cit. 2022-04-28]. Dostupné z: https://www.microsoft.com/en-us/download/details.aspx?id=13255

10. MICROSOFT. Microsoft Open Database Connectivity (ODBC). *Microsoft – Cloud, Computers, Apps & Gaming* [online]. Redmond: Microsoft, c2022, 2021-02-11 [cit. 2022-04-30]. Dostupné z: https://docs.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc?view=sql-server-ver15

11. MICROSOFT. Microsoft OLE DB. *Microsoft – Cloud, Computers, Apps & Gaming* [online]. Redmond: Microsoft, c2022, 2016-05-07 [cit. 2022-05-18]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms722784(v=vs.85)

12. Microsoft Access ODBC Driver connection strings. CONNECTIONSTRINGS.COM. *ConnectionStrings.com* [online]. -: ConnectionStrings.com, c2022 [cit. 2022-06-05]. Dostupné z: https://www.connectionstrings.com/microsoft-access-odbc-driver/

13. HERNANDEZ, Michael J. a John VIESCAS. *Myslíme v jazyku SQL: tvorba dotazů.* Praha: Grada, 2004. Knihovna programátora (Grada). ISBN 80-247-0899-x.

14. DRAYTON, Peter, Ted NEWARD a Ben ALBAHARI. *C# v kostce: pohotová referenční příručka.* Praha: Grada, 2003. ISBN 80-247-0443-9.

15. ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

16. KENT, Jeffrey A. *Visual C# 2005 - bez předchozích znalostí: průvodce pro samouky.* Brno: Computer Press, 2007. ISBN 978-80-251-1584-8.

10 Seznam webových odkazů

i. https://portal.pohoda.cz/about/o-portalu-pohoda/o-spolecnosti-stormware/
ii. https://www.stormware.cz/o-nas/
iii. https://www.stormware.cz/kontakty/gold-partner.aspx
iv. https://www.stormware.cz/prirucka-pohoda-online/Uvod/Sdeleni_k_plneni_relevantnich_norem/
v. https://www.stormware.cz/pohoda/pohoda.aspx
vi. https://www.stormware.cz/pohoda/pohoda-sql.aspx
vii. https://www.stormware.cz/pohoda/pohoda-e1.aspx
viii. https://www.stormware.cz/podpora/faq/pohoda/184/Na-jakem-Operacnim-systemu-lze-provozovat-POHODU/?id=3229
ix. https://www.stormware.cz/podpora/faq/pohoda/185/Po-aktualizaci-Microsoft-Windows-a-Microsoft-Office-se-zobrazuji-chybova-hlaseni-pri-spusteni-programu-POHODA/?id=3170
x. https://www.stormware.cz/podpora/faq/pohoda/185/Po-instalaci-programu-POHODA-se-pri-spusteni-aplikace-objevi-hlaseni-o-chybejicim-Service-Packu-pro-komponentu-Co-mam-delat/?id=3083
xi. https://portal.pohoda.cz/pro-podnikatele/jak-zacit-podnikat/aplikace-a-software-pro-podnikani/isdoc-jako-samozrejma-soucast-ucetniho-programu/
xii. https://www.stormware.cz/pohoda/xml/dokladyimport/

11 Přílohy

1) Zdrojový kód aplikace

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Jakub Schejbal**
Studium:	I2000409
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Aplikace pro spouštění uživatelských funkcí v ekonomickém systému Stormware POHODA.**
Název bakalářské práce AJ:	User Functions in the Economic System Stormware POHODA.

Cíl, metody, literatura, předpoklady:

Cílem práce je implementace aplikace umožňující spouštění funkcí v systému POHODA podle zadání uživatele s přímým přístupem do databáze systému pomocí jazyka SQL.

1. Popis systému POHODA a jeho využití při zpracování účetnictví.
2. Popis možností datové komunikace se systémem POHODA
3. Analýza možnosti přímého přístupu do databáze systému POHODA,
4. Implementace aplikace umožňující spouštění uživatelských funkcí s přímým přístupem do databáze systému POHODA.

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. RNDr. Petra Poulová, Ph.D.

Datum zadání závěrečné práce: 26.1.2021