TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie

Studijní obor: 1802R007 – Informační technologie

Software pro zdravotní prevenci

Software for health prevention

Bakalářská práce

Autor: **Toma Ondřej**

Vedoucí práce: Ing. Radek Srb

V Liberci 10. 5. 2013

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Chtěl bych tímto poděkovat hlavnímu vedoucímu bakalářské práce, Ing. Radku Srbovi, za jeho pomoc při řešení problémů, cenné nápady a připomínky a za konzultace, které mi poskytl při řešení této práce. Nemalý dík patří také mé rodině, která mi poskytla klid a místo na práci.

Abstrakt

Tématem bakalářské práce je návrh a realizace softwaru pro Projekt podpory zdravotní prevence. Tato aplikace bude sloužit pro správu základních entit, kterými jsou klienti, partnerské organizace, poskytované služby a poukázky.

V úvodu práce jsou popsány požadavky na software, na jeho funkcionalitu a rozšiřitelnost a také výběr vhodného programovacího jazyka a vývojového prostředí pro realizaci softwaru. V další části jsou vysvětleny architektonické prvky, které posloužily při návrhu aplikace, na který byl kladen velký důraz. Samotná realizace řešení je objasněna v následujících kapitolách a výsledná aplikace je popsána v poslední části bakalářské práce.

Klíčová slova

Projekt podpory zdravotní prevence, administrační software, vývoj aplikace, MVC, repositáře

Abstract

The topic of this thesis is software proposal and realisation for Medical prevention support project. Such application will be useful for maintenance of basis entities which are clients, partnership organizations, provided services and vouchers.

In the introduction, software requirements are stated, together with requirements for functionality and spreadfullness. Also, architectonic elements are explained – those that were useful during proposal of application which was of great emphasis. The realization alone is clarified in following chapters, and resulting application is described in the last part of my thesis.

Keywords

Medical prevention support project, administration software, application development, MVC, repositories

Obsah

Seznam obrázků

Použité pojmy a zkratky

- PPZP – Projekt podpory zdravotní prevence
- SPZP – Software pro zdravotní prevenci
- SP – Service Pack
- LINQ – Language Integrated Query
- TPL – Task Parallel Library
- GUI – Graphical User Interface
- IDE – Integrated Development Enviroment
- MVC – Model-View-Controller
- PNG – Portable Network Graphics
- JPG – Joint Photographic Group
- BMP – Bitmap
- SQL – Structured Query Language
- WPF – Windows Presentation Foundation

Úvod

Tato bakalářská práce se zabývá vývojem a realizací aplikace s názvem Software pro zdravotní prevenci určené pro administraci objektů figurujících v Projektu podpory zdravotní prevence. Těmi jsou klienti využívající služeb a výhod projektu, spolupracující organizace, služby poskytované partnerskými organizacemi a poukázky sloužící k získání a využití bodů.

Jako datové úložiště bude software využívat již vytvořenou vzdálenou databázi, která byla vytvořena v rámci bakalářského projektu. Aplikace bude také schopna komunikace s již vytvořeným lékařsko-lékárenským softwarem jménem Praescriptor.

Motivací pro vývoj této aplikace je vytvořit software, který by pomohl nastartovat Projekt podpory zdravotní prevence a dostal ho do podvědomí lidí a organizací, které by mohly v budoucnu spolupracovat. Projekt podpory zdravotní prevence je zaměřen na podporu zdravého životního stylu a měl by pomoci motivovat lidi ke sportování a k udržení se v kondici a předejít tak například nadbytečným kilogramům, které v dnešní době trápí každého třetího člověka a obezita každého pátého. S nadbytečnou váhou jsou spojené určité zdravotní komplikace. Snížením tělesné váhy se může předejít civilizačním chorobám např. cukrovce, hypertenzi, zvýšenému cholesterolu, infarktu atd.

1 Požadavky na software

Aby bylo možné začít s vývojem softwaru, je potřeba si upřesnit pár základních věcí. V této kapitole bude popsáno, jaké požadavky byly kladeny na aplikaci a jak se k nim dospělo.

1.1 Desktopová aplikace

Po krátkém průzkumu bylo zjištěno, že lékaři a organizace, které by mohli spolupracovat v Projektu podpory zdravotní prevence, využívají operační systémy z rodiny Windows. Především jsou to Windows XP a Windows 7. Nejstarší nalezené systémy jsou Windows XP SP2.

Na základě tohoto zjištění bylo rozhodnuto, že aplikace bude desktopová. Bude spustitelná na desktopovém počítači nebo laptopu a primárně bude využívat hardware tohoto zařízení a lokálně instalované nástroje.

1.2 Použitá technologie

K realizaci bakalářské práce budou využívány určité nástroje a technologie. V této kapitole je představím.

1.2.1 Programovací jazyk C# a platforma .NET

Z důvodu produktivity a časové tísně se okruh možných programovacích jazyků pro aplikaci zúžil na jazyky nejvyšší úrovně typu Java, C# a jim podobné. Rozhodl jsem se pro využití jazyka C#, se kterým jsem se seznámil ve výuce.

C# je výkonný vysokoúrovňový objektově orientovaný, ale při tom jednoduchý jazyk zaměřený na vývojáře na platformě .NET Framework vyvinutý firmou Microsoft. Zdědil velké množství dobrých vlastností například z jazyka C++. Výsledkem je tedy čistší a logičtější jazyk. Jazyk C# měl svoji veřejnou premiéru 15. ledna 2002 a to ve verzi 1.0 společně s .NET Frameworkem 1.0 a obsahovala základní podporu objektového programování.

Na druhou verzi se čekalo až do roku 2005, kdy společně s Visual Studiem 2005 přišel i C# 2.0. Přinesl několik důležitých nových prvků, jako jsou například generické typy, iterátory, částečné a statické třídy nebo anonymní metody. Jazyk C# 3.0, který byl vydán společně s Visual Studiem 2008 a .NET Frameworkem 3.5, přidal také několik

nových věcí. Byly jimi například rozšiřující metody, lambda výrazy a technologie LINQ, jež je asi ze všech novinek nejzajímavější.

Následující verze s označením C# 4.0 vyšla v roce 2010 a nabízí také další rozšíření, která zlepšují jeho interoperabilitu s ostatními jazyky a technologiemi. Jazyk C# 4.0 také vyšel s novou verzí .NET Framework se stejným označením 4.0. Toto vydání rozhraní .NET Framework obsahuje řadu přídavků oproti předchozím verzím, z nichž jsou asi nejvýznamnější třídy a typy, které tvoří knihovnu TPL. Pomocí této knihovny lze vytvářet vysoce škálovatelné aplikace, které dokážou velice dobře využít potenciál více jádrových procesorů. Poslední verzí jazyka C# je verze 5.0, která vyšla společně s Visual Studiem 2012. Tato poslední verze přišla se zcela novým způsobem, jak vytvářet a volat asynchronní metody. Více informací o jazyku C# lze nalézt v knihách [1] a [5].

1.2.2 Microsoft Visual C# 2010 Express

Microsoft Visual Studio je vývojové prostředí, které je vytvořeno společností Microsoft. Lze ho využít pro vývoj konzolových aplikací, aplikací s grafickým rozhraním, webových stránek, webových aplikací atd. Programy lze vytvářet na platformách Microsoft Windows, Windows Mobile, Windows CE, .NET a Microsoft Silverlight.

Visual Studio neobsahuje jen editor kódu a debugger, ale i vestavěné nástroje zahrnující designer pro tvorbu aplikací s GUI, designer webu, tříd i databázových schémat. Některé nástroje lze i dodatečně doinstalovat, jako například nástroje pro vývoj aplikací v týmu.

Konkrétně prostředí Microsoft Visual C#, implementuje, jak už je v názvu zřejmé, programovací jazyk C#. Edice Express je odlehčená verze, která je volně ke stažení a je zdarma. Obsahuje tedy jen malou sadu nástrojů oproti verzím Professional a Ultimate. Neobsahuje například podporu vzdálené databáze pro designer dat, designer tříd atd. Vývojová prostředí s přívlastkem Express jsou směřována spíše pro studenty a amatéry, kteří si nemohou dovolit placené verze.

12. 4. 2010 bylo vydáno Microsoft Visual Studio 2010. Nové IDE bylo přepracováno a podle společnosti Microsoft snížili nepřehlednost a složitost starších verzí. Je zde větší podpora pro multi-monitorové programování a lepší přehlednost při více otevřených dokumentech. Nově je tu i přepracována podpora WPF. Co ocení

mnoho programátorů je zvýraznění názvů proměnné na všech místech, kde se nachází. To slouží zejména k rychlému vyhledávání ve zdrojovém kódu. Podobných nástrojů pro usnadnění práce se ve Visual Studiu 2010 nachází mnoho.

1.2.3 Jazyk SQL a databázový systém MySQL

Jazyk SQL z anglického „Structured Query Language“, tedy strukturovaný dotazovací jazyk, je standardizovaný dotazovací jazyk určený pro práci s relačními databázovými systémy. Počátky jazyka jsou v 70. letech 20. století, kdy jej začala vyvíjet společnost IBM. V roce 1986 byl jazyk standardizován americkým institutem ANSI.

MySQL je multiplatformní databázový systém, ve kterém probíhá vlastní ukládání dat nebo práce s daty. Komunikace s databází probíhá pomocí jazyka SQL. Jedná se o volně šiřitelný software a pro svou snadnou implementovatelnost se jedná o velmi oblíbenou databázi. Bylo však od začátku optimalizováno především na rychlost a to i za cenu, že neobsahovala vše, co její konkurence. Pro výběr MySQL jsem se rozhodl z důvodu, že oproti konkurenci je zadarmo.

1.3 Funkcionalita

Funkcionalitou je zde míněno, jak by měl software správně fungovat a jaké by měl splňovat požadavky. Je zapotřebí určit základní seznam úloh, které by měl software nabízet. Všechny tyto funkce musí nakonec tvořit jeden plnohodnotný celek.

Aby byla aplikace pro uživatele příjemnější z hlediska ovládání, je potřeba vytvořit GUI. Jak už bylo zmíněno v kapitole 1.1, software bude muset být podporován minimálně na operačním systému Windows XP SP2. Z tohoto důvodu bude aplikace obsahovat grafické komponenty a bude navržena za použití WinForms. Tato technologie slouží k vývoji formulářových aplikací na platformě .NET Framework. Technologie WinForms obsahuje veškeré komponenty a ovládací prvky, na které jsou uživatelé formulářových aplikací zvyklí.

Software bude obsahovat čtyři základní moduly, které budou sloužit pro správu entit nacházejících se v Projektu podpory zdravotní prevence. Jelikož budou data získávána ze vzdálené databáze, je potřeba při vkládání a úpravě jednotlivých objektů dbát na bezpečnost a správnost vkládaných dat a také na duplicitu u atributů, které musí být unikátní. Při mazání jednotlivých dat je třeba dávat pozor, aby nevznikly v úložišti

díry a poté nedošlo ke kolapsu systému. Mnoho opatření již existuje na straně vzdáleného serveru, ale je vhodné provést určité kroky i na straně aplikace.

Prvním modulem, který by se měl v aplikaci nacházet, je správa uživatelů, kteří vstoupí do projektu. Je důležité určit povinné a nepovinné atributy klientů při administraci. Problém, který v této situaci nastává, je, že lidé nechtějí poskytovat své osobní údaje, jako jméno, příjmení a rodné číslo. Je tedy potřeba zvolit údaj, podle kterého budou jednotlivé osoby rozlišovány. Tímto identifikátorem by mohla být emailová adresa. Je unikátní a umožňuje pozdější komunikaci s klientem. Další jednoznačnou identifikací by mohla být karta, kterou už klient vlastní, například kartička pojišťovny (VZP, …), studentský průkaz ISIC nebo karta, kterou by mohli klienti obdržet po registraci do PPZP. Pomocí karty by se mohli klienti prokazovat například při návštěvách sportovních center. Všechny tyto karty obsahují jednoznačný identifikační kód, takže by neměl nastat žádný problém s duplicitou. Pro lepší a jednodušší identifikaci klienta a pro orientaci v systému by měla být i možnost zadat jméno, příjmení, telefonní číslo a rodné číslo. Administrace nespočívá jen v přidávání údajů, ale musí poskytovat i možnost úprav a mazání. Dále by měl tento modul umožňovat přidávání bodů klientovi za vykonanou činnost související s tímto projektem, jako je například návštěva wellness centra či vykonání sportovní činnosti. Dále by měl nabídnout možnost zobrazení, zda existují nějaké zdravotní záznamy v lékařsko-lékárenském softwaru s názvem Praescriptor.

Dalším modulem v aplikaci by měla být administrace spolupracujících organizací. Jedná se o organizace, které budou klientům poskytovat služby, za které budou získávat body nebo naopak služby, na které budou body využívat. V této části by se mělo řešit přidávání, úpravy a mazání jednotlivých poskytovatelů. Jako jednoznačný identifikační kód by zde mělo sloužit ičo jednotlivých firem. Každá organizace by měla spadat do určité kategorie podle svého zaměření, například pojišťovna, wellness, lékárna, atd. Jako další atributy by zde měly být název společnosti, webová adresa, emailová adresa, popřípadě telefonní číslo.

Třetí část by měla řešit administraci služeb, jež nabízejí jednotliví poskytovatelé. Služba by měla spadat do dvou základních typů a to pro získání bodů a využití bodů. Podle toho se bude se službami dále pracovat. Služby budou moci spadat do více kategorií, podle kterých budou dále děleny. Například kondiční posilování může spadat jak pod posilování ve wellness centru, tak například pod plavání v bazénu. Každá služba

bude někde nabízena, a proto by měla mít přiřazeného svého poskytovatele. U služby pro využití bodů by měla být možnost na určení intervalu platnosti služby. Samozřejmostí by měla být i možnost zadat název služby a nějaký popis pro pozdější využití například na poukázky či na nějakou budoucí propagaci.

Poslední částí SPZP bude administrace poukázek. Za prvé by zde měla existovat možnost pro administraci a pozdější výběr šablony pro poukázku. Šablonou se myslí podklad pro poukázku, na kterém by se mohl nacházet nějaký motiv nebo obrázek. V rámci vytváření vzoru by měla být nabídnuta možnost určit pozici jména klienta, pro kterého bude poukázka tištěna a čárového kódu pro zjednodušení pozdější administrace. Za druhé by měl být k dispozici přehled již přijatých poukázek a také přehled poukázek, které jsou vydané a jsou stále v oběhu mezi lidmi. V neposlední řadě je potřeba poukázky tisknout a dát tak možnost klientům získávat nebo využívat nasbírané body.

1.4 Rozšiřitelnost a znovu použitelnost

Předpokládá se, že Projekt podpory zdravotní prevence bude projektem rozsáhlejšího charakteru a jeho fungování a obsluhu bude zajišťovat několik aplikací. Z tohoto důvodu by se při tvorbě desktopové aplikace SPZP měl klást velký důraz na samotný návrh architektury aplikace. Zaměřit se na striktní oddělení dat, aplikační logiky a grafického uživatelského rozhraní. Navrhnout hierarchickou strukturu za použití abstrakce s cílem eliminovat tak opakovanou implementaci již hotových mechanismů. Dodržování těchto pravidel zajistí snadnou rozšiřitelnost funkčnosti aplikace a minimalizaci počtu nutných změn v případě požadavků na změnu principu fungování dílčích částí.

2 Architektonické prvky

V této kapitole vysvětlím několik abstrakčních technik souvisejících s návrhem aplikace v objektově orientovaném programovacím jazyce, které jsem si nastudoval, abych splnil předešlé požadavky na aplikaci při realizaci softwaru. V další kapitole bude vysvětleno, jak jsem jednotlivé pojmy implementoval do svého řešení.

2.1 Návrhové vzory

Návrhový vzor si lze představit jako soubor pravidel pro obecné řešení problému. Nejedná se tedy o žádnou knihovnu či zdrojový kód, který lze vložit do programu, ale pouze o šablonu určující ověřený obecný postup při tvorbě softwaru, který při řešení problému dodržovat. Dodržováním návrhových vzorů nevznikají chyby, kterých bychom se normálně dopustili.

Dalším důvodem, proč využít návrhové vzory je, že dnes existuje mnoho vývojových prostředí, kde se dá velmi snadno vyvinout aplikace pomocí předpřipravených komponent v nějakém designeru. Jenže časem se přijde na to, že se v aplikaci velice často opakuje stejný kód nebo potřebujeme zobrazit stejná data na několika místech. Při větších projektech tedy zjistíme, že pohodlí takového vývoje s sebou nese určité komplikace, které později stojí mnoho času a úsilí na jejich odstranění.

2.1.1 MVC

MVC neboli Model-View-Controller (česky: Model-Pohled-Kontrolér) je návrhový vzor, který je určen pro komplexnější úlohy. Tento vzor rozděluje aplikační logiku do třech nezávislých celků. Těmi jsou Model, který reprezentuje data, která aplikace využívá a pracuje s nimi, Pohled představující uživatelské rozhraní a Kontrolér určený pro řízení aplikační logiky a chod aplikace. Aplikační logikou může být i řešení zachycených událostí vyvolaných uživatelem. Zajišťuje tedy interakci mezi modely a pohledy. Základní myšlenkou je, že všechny tři celky jsou na sobě v maximální možné míře nezávislé a nemíchá se například vzhled aplikace a práce s daty v jedné třídě. Každou část tedy lze upravovat samostatně a dopad změn na ostatní části je minimální.

2.1.2 Repository Pattern

Máme-li data uložená v jakékoliv databázi, je dobré oddělit operace týkající se přístupu k datům od ostatní aplikační logiky. K tomu právě slouží Repository Pattern neboli Repositáře. O načítání, úpravu a mazání dat se stará jedna konkrétní třída, která obsahuje pouze logiku pro práci s databází. Výhody takové řešení jsou:

- Centrálně řízený přístup ke vzdáleným datům
- Čitelnost kódu oddělením aplikační logiky od logiky přístupu k datům
- Možnost změn pouze na jednom místě

2.2 Genericita

Jazyk C# umožňuje využívat takzvané generické typy. Slouží k tvorbě zobecněných tříd či metod. Dalo by se to chápat jako šablona, díky které je možné daný kód využít pro více datových typů.

Existují dva způsoby, jak vytvořit obecné třídy a metody. V jazyce C# lze vytvořit proměnnou typu object. Do této proměnné pak lze ukládat hodnoty libovolných typů. Pokud se tedy tento typ použije jako parametr v metodě, je možné metodám předávat hodnoty libovolných typů. Když se použije typ object na místo návratového typu, metoda může vracet hodnotu libovolného typu. Tento postup je sice velice pružný, ale zahrnuje několik úskalí. Zatěžuje programátora, který si musí pamatovat, s jakým typem právě pracuje a pokud se zmýlí, vede to k chybám, které se projeví až za běhu programu.

Druhý způsob, kterým lze nadefinovat obecné třídy a metody, jsou právě generické typy, které byly navrženy tak, aby programátorovi pomohly vyvarovat se podobných chyb. Odstraňují nutnost přetypování, vylepšují typovou bezpečnost a usnadňují tak tvorbu zobecněných tříd a metod. Generické třídy a metody mají typové parametry, které jsou nahrazeny konkrétními typy v době použití kódu. Generický typ se uvádí na místě skutečného datového typu v metodě, třídě, rozhraní atd. a je ohraničen špičatými závorkami. Ukázka vytvoření generické třídy a její použití:

```
public class Item<T>
{
    private T _item;

    public Item(T item)
    {
        _item = item;
    }

    public T GetItem()
    {
        return _item;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Item<int> item1 = new Item<int>(10);
        Item<string> item2 = new Item<string>("Item 2");
        Item<DateTime> item3 = new Item<DateTime>(DateTime.Now);

        Console.WriteLine(item1.GetItem());
        Console.WriteLine(item2.GetItem());
        Console.WriteLine(item3.GetItem());

        // Výpis na obrazovce
        // 10
        // Item 2
        // 12.5.2013 11:08:13
    }
}
```

2.3 Dědičnost, Polymorfismus, Rozhraní

Dědičnost je základ objektového programování. Jejím použitím se zamezuje opakovanému psaní stejného kódu v definici různých tříd, které mají mnoho částí společných a jsou mezi sebou provázány. Jako příklad lze uvést třídu dopravní prostředky, jejíž potomky budou osobní automobil, motocykl a autobus. Společným prvkem může být hmotnost, kterou bude mít každý dopravní prostředek a automobil je doplněn o vlastnost objemu kufru, která je specifická pouze pro něj. Právě v takovéto situaci je dědičnost užitečná.

Polymorfismus je úzce spjat s dědičností a je postaven na typové kompatibilitě předka s jeho potomky. Umožňuje ukládat instance potomků do svých předků, což je výhodné například pro parametry funkcí.

Rozhraní neobsahuje žádný kód ani data, ale pouze specifikace metod a vlastností, které musí třída implementující toto rozhraní poskytovat. Díky využití

rozhraní tedy říkáme, co daná třída umí, ale neříkáme, jak to dělá. Odděluje tedy názvy a signatury jednotlivých metod třídy od vlastní implementace.

Hlubší výklad na téma dědičnost, polymorfismus a rozhraní lze najít v knihách [1] a [5].

3 Realizace řešení

V této kapitole bude vysvětleno, jak jsem aplikoval jednotlivé prvky a možnosti objektového programování k realizaci Softwaru pro zdravotní prevenci. V situacích, kdy jsem narazil na dosud neřešený typ problému, jsem hledal inspiraci k jeho řešení na webových stránkách [2], [3] a [4].

3.1 Implementace MVC v SPZP

Protože neexistuje žádný oficiální Framework pro WinForms, který umožňuje snadnou implementaci MVC vzoru, jako je tomu například pro ASP.NET, musel jsem vytvořit implementaci vlastní. Díky jejímu použití má projekt velice dobré rozdělení aplikační logiky. Podle té je také vedena základní strukturu adresářů a jmenný systém udržující snadnou navigaci v projektu.

Prvním adresářem jsou modely. Zde se nacházejí jednotlivé třídy reprezentující entity reálného světa a pomocné třídy k udržení potřebných dat. Modely však mohou obsahovat i vlastní vnitřní logiku, která slouží například pro validaci jednotlivých atributů.

Druhým adresářem, který aplikuje model MVC do mého projektu, jsou pohledy. Jelikož se jedná o formulářovou aplikaci vyvíjenou ve WinForms, tento adresář obsahuje velké množství formulářů reprezentující pohledy určené pro komunikaci s uživatelem. Z důvodu, že je aplikace primárně rozdělena na čtyři části a pohledů je velké množství, jsou i ony rozděleny do čtyř základních adresářů. Objevují se zde pohledy určené pro výpis dat, výběr dat či pro přidání a úpravu jednotlivých objektů. Některé formuláře jsou vázané na konkrétní model a zachytávají jednotlivé události vyvolané uživatelem, které dále předávají kontroléru. Některé pohledy slouží pouze pro navigaci v programu nebo jiné potřebné operace.

Posledním adresářem nacházejícím se v projektu ze struktury MVC, jsou kontroléry. Protože Software pro zdravotní prevenci je funkcionálně velice obsáhlý, v této složce se nachází kontrolérů hned několik. Každý modul je obsluhován jedním kontrolérem, který se stará o potřebnou logiku, jako je otevírání a zavírání jednotlivých formulářů, zachytávání a následná obsluha událostí a podobně.

Už od začátku se muselo počítat s tím, že kvůli použití návrhového vzoru MVC se nebude aplikace pouštět klasickým způsobem. Musím si nejprve inicializovat kontrolér, kterému předám jako parametr pohled, který otevře. Teprve poté se aplikace spustí.

3.2 Implementace repositářů do SPZP

Další nedílnou součástí tohoto softwaru je komunikace se vzdálenou databází. K této komunikaci slouží třída přímo vytvořená k této činnosti, ale dotazy na jednotlivá data se nacházejí v repositářích. Repositáře se nacházejí v adresáři pro ně určeném. Je to tedy stejné jako s implementací MVC. Jednotlivé repositáře v sobě ukrývají dotazy na databázi, čímž tvoří datovou vrstvu, která zajišťuje mapování dat z tabulek do objektů použitých v aplikaci a naopak. Dále se také starají o validaci jednotlivých modelů, které chce uživatel přidat či upravit.

3.3 Objektový návrh

Na architekturu softwaru byl kladen veliký důraz. V situacích, kdy se předpokládá, že bude software obsáhlý, je dobré trávit nad návrhem aplikace spoustu času a pokusit se tak předejít situacím, kdy se v konečných fázích vývoje zjistí, že došlo k fatální chybě a musí se se vším začít znovu.

Vývoj si vyžádal veliké množství abstrakce, protože se od softwaru očekávalo, že bude jednoduše rozšiřitelný a co nejvíce jeho částí znovu použitelných v budoucím vývoji. Muselo se určit, jak budou jednotlivé moduly pracovat a jaké objekty budou využívat. Určit si tak, co mají společného a co by se dalo použít na více místech a předejít tak pozdějším komplikacím s případnou změnou či opravami softwaru.

Aplikace je rozdělena na čtyři základní moduly. Nemyslím tím části, které vyplynuly z návrhového vzoru MVC, ale o celcích, kterými se aplikace zabývá. Jedná se o klienty, poskytovatele služeb, služby a poukázky. Každý modul obsahuje vlastní kontrolér, minimálně jeden repositář, několik pohledů a nemalé množství modelů. V následujících kapitolách bude popsáno, jak byly jednotlivé komponenty navrženy.

3.3.1 Kontroléry

Prvními objekty, které bylo potřeba navrhnout pro aplikaci a pro její správné fungování, byly kontroléry. Jak už bylo zmíněno v předchozích odstavcích, kontrolér se stará o logiku aplikace a její chod. V aplikaci se nachází pět kontrolérů. První z nich

slouží pro inicializaci a zavedení aplikace a pro otevření úvodního formuláře, který slouží jako navigace pro výběr jednoho ze čtyř modulů. Po výběru modulu tento kontrolér vytvoří instanci jednoho ze čtyř zbývajících kontrolérů, které jsou vázány k jednotlivým modulům a starají se o logiku každého z nich. Společně s kontrolérem se musejí vytvořit instance formulářů neboli pohledů, které jsou přístupné ve vybraném segmentu a instance repositářů, kterých bude zapotřebí pro přístup k datům zobrazovaným ve vybrané části. Z tohoto důvodu jsem vytvořil předka, který je uveden v následující ukázce:

```
public abstract class Controller<T,U>
{
    protected T _views;
    protected U _repositories;

    public Controller(T views)
    {
        _views = views;
    }

    public Controller(T views, U repositories)
    {
        _views = views;
        _repositories = repositories;
    }
}
```

Kontrolér je z důvodu zamezení vytvoření instance navržen jako abstraktní a zároveň jako generická třída, které se předávají dva typové parametry. První parametr, který je označen „T“, představuje pohled či pohledy, se kterými bude kontrolér pracovat. Za parametr „T“ lze vložit konkrétní pohled nebo třídu, která obsahuje všechny pohledy potřebné ke správnému fungování vybrané části. Druhým typovým parametrem je „U“, který slouží pro upřesnění, se kterými repositáři bude kontrolér schopný pracovat. Za parametr je možné dát jeden konkrétní repositář nebo třídu, která bude obsahovat veškeré repositáře potřebné pro daný segment aplikace.

V uvedeném předkovi se nacházejí dva konstruktory třídy. Je tomu tak z důvodu, že úvodní kontrolér, který je potomkem této generické třídy, ke své práci nepotřebuje žádný repositář a z toho důvodu je pro něj vhodné použít konstruktor třídy pouze s jedním parametrem. Pro zbytek potomků kontroléru je určen konstruktor se dvěma parametry.

3.3.2 Modely

Modely jsou třídy, které slouží pro uchování dat a pro následnou práci s nimi. Nemají mezi sebou nic společného, proto se zde nedalo využít žádného dědění či nějakého rozhraní, které by specifikovalo, co má daná třída umět. V projektu se nacházejí třídy vycházející z entit reálného světa, ale také třídy, které jsou spíše pomocné a obsahují například parametry určené pro filtraci dat v pohledech nebo třídy, které poskytují lepší rozšiřující popis pro výčtové typy. V modelech, které představují konkrétní entity, jako například třída Klient, je obsažena krom atributů i validační logika.

3.3.3 Pohledy

Protože se jedná o formulářovou aplikaci, očekávalo se velké množství pohledů. Z tohoto důvodu bylo potřeba vymyslet adresářovou hierarchii, která by kopírovala strukturu aplikace a jednoznačně určila, k čemu daný pohled slouží.

Po navržení adresářové struktury bylo potřeba určit, jak budou fungovat jednotlivé pohledy, co budou mít společného a navrhnout podle toho základní rozhraní, která budou určovat, co bude daný pohled umět. Protože je aplikace navržena podle návrhového vzoru MVC, otevírání pohledů obstarává kontrolér. Z tohoto důvodu musí každý formulář obsahovat veřejnou metodu pro zobrazení. Ta byla umístěna do rozhraní s názvem IView, které je implementováno v každém formuláři.

Formuláře jsou rozděleny na dva typy. Na katalogy a detaily. Každý typ má rozdílnou funkcionalitu. Katalogy jsou většinou prvním formulářem, který se uživateli zobrazí, při vybrání kteréhokoliv modulu v aplikaci. Aby se formulář mohl nazvat katalogem, musí být jeho součástí výpis objektů do tabulky. Protože objektů v tabulce může být pokaždé jiné množství, každý katalog obsahuje metodu pro navrácení počtu řádků seznamu. Poslední funkcí základního katalogu je výběr objektu z tabulky. Všechny tyto metody jsou obsaženy v rozhraní s názvem ICatalogView, které je implementováno všemi katalogy v aplikaci. Signaturu metod si lze prohlédnout v následující ukázce:

```
public interface ICatalogView<T> : IView
{
    void LoadData(List<T> ListOfObjects, States state);
    T GetObjectFromDataGrid();
    int GetRowsCountDataGrid();
}
```

Jak je vidět v ukázce, je zde využito genericity. Za parametr „T“ se dosazuje třída, se kterou katalog pracuje. Pokud se například jedná o katalog klientů, modelem dosazeným za parametr „T“ bude třída Klient. Dále je vidět, že ICatalogView je rozšířeno o rozhraní IView a tím získává metodu pro zobrazení formuláře.

Jelikož katalog není jen o výpisu dat, ale patří k němu například i filtrace dat, vytvořil jsem další rozhraní obstarávající události a metody spojené s vyhledáváním. Události a metody si lze prohlédnout v ukázce:

```
public interface ISearchableView<T>
{
    event EventHandler Search;
    event EventHandler ClearSearch;

    void ResetFiltr();
    T GetDataForSearch();
}
```

Opět rozhraní využívá genericity. Parametr „T“ lze nahradit objektem obsahujícím parametry pro hledání. Hledání však nemusí být spojeno pouze s katalogem a proto lze toto rozhraní implementovat v jakémkoliv formuláři.

Další funkcionalita, která se objevuje v katalozích programu je otevření pohledů pro přidání a úpravu nebo mazání konkrétního objektu. Protože se tyto události objevují pouze ve vybraných formulářích s rozšířenou funkčností, je jejich signatura uvedena v samostatném rozhraní:

```
public interface ICatalogWithBasicEventsView<T> : ICatalogView<T>
{
    event EventHandler Add;
    event EventHandler Edit;
    event EventHandler Delete;
}
```

V ukázce je vidět, že jde opravdu jen o definici událostí, které jsou po implementaci svázané s tlačítky nacházejícími se v katalozích.

Druhým typem formulářů v aplikaci jsou detaily. Jedná se o formuláře, které slouží k přidání, úpravě nebo k zobrazení informací o požadovaném objektu. Jsou to i formuláře, které slouží pro výběr objektu či objektů pro další zpracování. Detaily jsou otevírány po vyvolání událostí k tomu určeným. U detailů se však musí řešit i jejich zavírání a čištění jejich komponent. Pro určení signatury slouží rozhraní, které je uvedeno v následující ukázce:

```
public interface IDetailView : IView
{
```

```
        void CloseDialog();
        void ClearAllControls();
    }
```

Jak je vidět, rozhraní IDetailView je rozšířeno o funkcionalitu z IView. To znamená, že formulář implementující IDetailView je možné jak otevírat, tak i zavírat pomocí metod.

Další funkcionalita, která se musí řešit v detailech, je rozdělena podle toho, k čemu je konkrétní formulář určen. Je rozdíl v detailu, který jen data zobrazuje a v detailu, který je určen pro úpravu objektů či v detailu, který je určen jen pro přidání nových dat. Protože s těmito úkony je spojeno několik metod, které se mohou využívat na více místech, rozhodl jsem se to rozdělit do několika rozhraní.

První, co se musí v detailu po otevření řešit, je, zda je potřeba předem vyplnit údaje. Pokud ano, jedná se o formulář, který je určený pro výpis informací o objektu nebo pro úpravu konkrétního objektu.

```
public interface IReadItemDetailView<T> : IDetailView
{
    void SetObjectToControls(T item);
}
```

V kódu je uvedena signatura metody, která se stará o vyplnění informací do komponent k tomu určených. Třída opět využívá genericitu. Za parametr „T“ se očekává objekt, o kterém se mají vyplnit požadované informace.

Pokud se údaje předem vyplnit nemusí a stačí je pouze ukládat po vyplnění uživatelem, stačí implementovat rozhraní, které je uvedeno v další ukázce:

```
public interface IAddItemDetailView<T> : IDetailView
{
    event EventHandler Insert;

    T GetNewObjectFromControls();
}
```

Toto rozhraní je určeno k ukládání dat. Po implementaci tohoto rozhraní je patrné, že formulář bude obsahovat tlačítko, ke kterému bude přidána událost pro ukládání dat. Formulář bude také obsahovat metodu, která vrací objekt určený pro uložení do vzdáleného uložiště. Po kombinaci tohoto a předešlého rozhraní lze vytvořit formulář určený pro úpravu dat. Formulář je poté schopen vypsat informace o objektu pomocí metody a po jejich změně vrátit upravený objekt, který lze následně uložit do vzdálené databáze.

Všechna rozhraní, která jsou uvedena v této kapitole lze implementovat v jakémkoliv novém formuláři podle požadavků na budoucí funkcionalitu. Tím jsem docílil, že je vždy jasné, co konkrétní formulář umí a co se v něm nachází za funkce. Nicméně pokud bylo potřeba doplnit formulář o další funkcionalitu, bylo vytvořeno nové rozhraní, které bylo rozšířeno o potřebné rozhraní uvedené v minulých odstavcích a doplněno o nové události či metody.

3.3.4 Repositáře

Poslední důležitou částí jsou repositáře. Slouží pro komunikaci s perzistentním úložištěm dat. V projektu je nadefinováno rozhraní, které uvádí základní funkcionalitu každého repositáře, který je v projektu vytvořen. Jedná se o metody přidání, úpravu a mazání dat a také o metodu sloužící pro validaci vkládaných dat. Signatura metod je k vidění v následující ukázce:

```
public interface IRepository <T>
{
    void Insert(T ObjectForSave);
    void Update(T ObjectForUpdate);
    void Delete(T ObjectForDelete);
    bool IsValid(T Object);
}
```

Opět je použita genericita. V tomto případě za typový parametr „T“ patří objekt, který bude ukládán nebo mazán ze vzdálené databáze. Chybějící funkčnost je definována v rozhraních určených pro konkrétní repositáře. Jedná se především o signaturu metod pro získání dat ze vzdáleného úložiště.

3.4 Databázový model

V průběhu vývoje vyvstávaly nové požadavky na ukládání dat a bylo tedy nutné provádět změny v databázovém modelu, který byl vytvořen jako bakalářský projekt. Bylo zapotřebí přidat několik atributů, které zapříčinily změnu stávajících pohledů, vstupních parametrů procedur a triggerů, které s těmito atributy souvisí.

4 Popis jednotlivých modulů

V této kapitole popíši jednotlivé moduly, jak fungují a jak vypadají.

4.1 Hlavní okno

Tento formulář slouží k úvodnímu výběru, kde si uživatel může zvolit, jaké sekci se bude věnovat. Na výběr má ze čtyř základních segmentů.

- KLIENTI
- POSKYTOVATELÉ SLUŽEB
- SLUŽBY
- POUKÁZKY

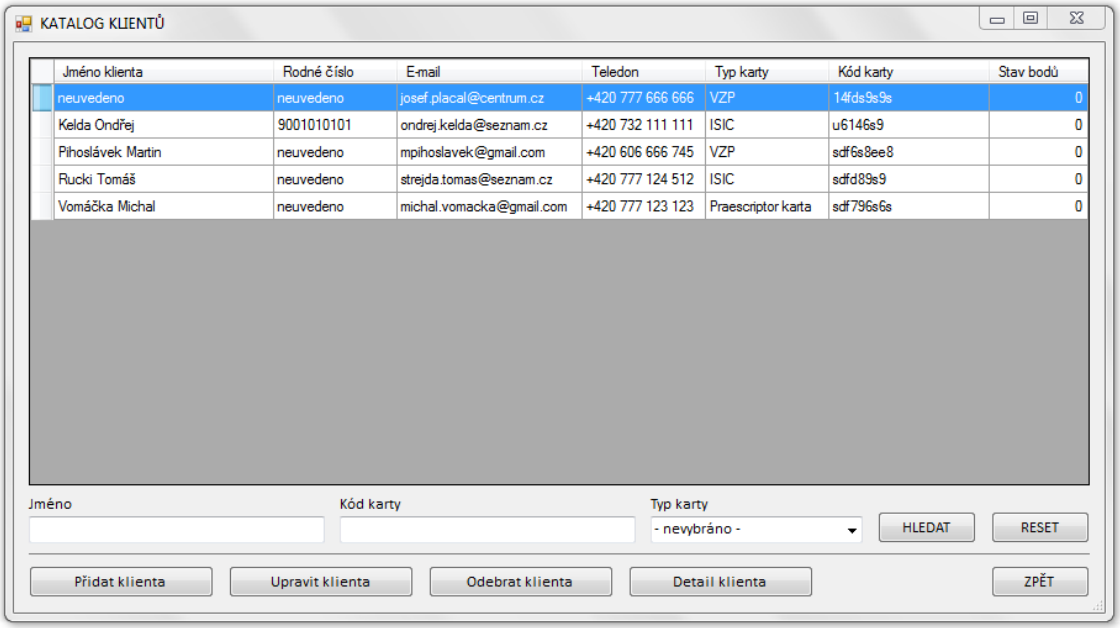
Po kliknutí se uživateli zobrazí úvodní formulář z vybrané části.

Obrázek 4.1 – Úvodní formulář

4.2 Modul KLIENTI

Tento segment je určen pro administraci klientů, kteří se zúčastní Projektu podpory zdravotní prevence. Po vybrání tohoto modulu se jako první zobrazí katalog klientů, kteří jsou zaregistrováni v PPZP. V tabulce lze vyčíst veškeré základní údaje o klientovi (viz Obrázek 4.2). V seznamu lze vyhledávat pomocí zadaných kritérií.

Filtrovat se dají klienti podle jména, kódu karty a typu karty. Po zadání jakéhokoliv kritéria stačí kliknout na tlačítko „HLEDAT“ a požadované záznamy se zobrazí v tabulce katalogu. Pro rychlé smazání dat určených pro filtraci slouží tlačítko „RESET“.

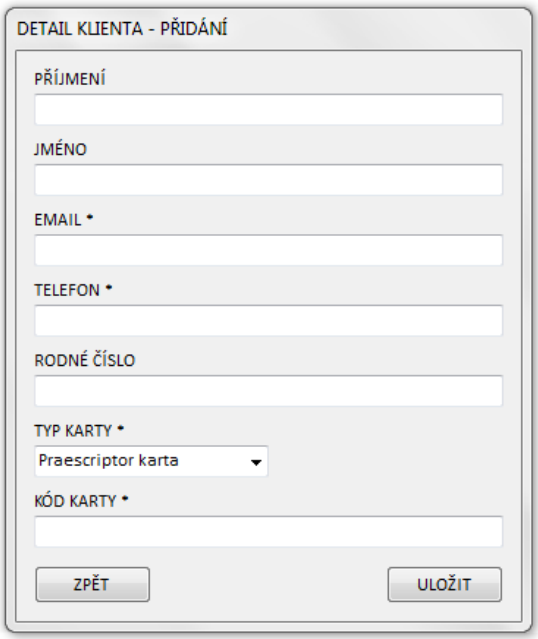


Obrázek 4.2 – Katalog klientů

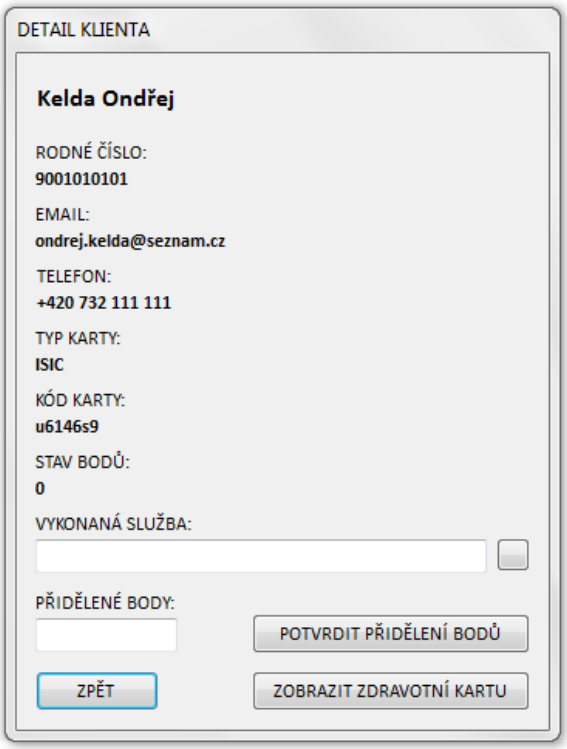
Z katalogu se lze dostat k samotnému přidávání, úpravě nebo detailu klienta. Stačí kliknout na tlačítka umístěna na spodní straně formuláře. Po kliknutí na „Přidat klienta“ se zobrazí formulář pro přidání nového klienta (viz Obrázek 4.4). Tento detail slouží pro vyplnění údajů o novém klientovi a následné uložení. Povinnými atributy pro uložení jsou email, telefon, typ karty a kód karty. Pro zadání správného tvaru telefonního čísla a emailu se po najetí do textového pole zobrazí bublina s nápovědou, kde je uveden konkrétní tvar. Příjmení, jméno a rodné číslo slouží spíše pro upřesnění údajů a pro lepší orientaci v programu.

Pro úpravu klienta stačí vybrat možnost „Upravit klienta“ a otevře se okno, které vypadá shodně s formulářem pro přidání (viz Obrázek 4.4), ale už je vyplněné. Je předvyplněné údaji patřící klientovi, kterého se uživatel rozhodl změnit. Stačí potřebné údaje změnit, nepovinné atributy lze odebrat, či doplnit. Pokud je vše v pořádku vyplněno, lze stisknout tlačítko „Uložit“ a požadovaná změna klienta se okamžitě projeví v katalogu.

Pro odebrání klienta slouží tlačítko „Odebrat klienta“. Po jeho stisknutí se objeví zpráva, kde je uživatel dotázán, zda si je jist s odebráním klienta. Po potvrzení se klient odebere z katalogu.



Obrázek 4.4 – Přidání klienta



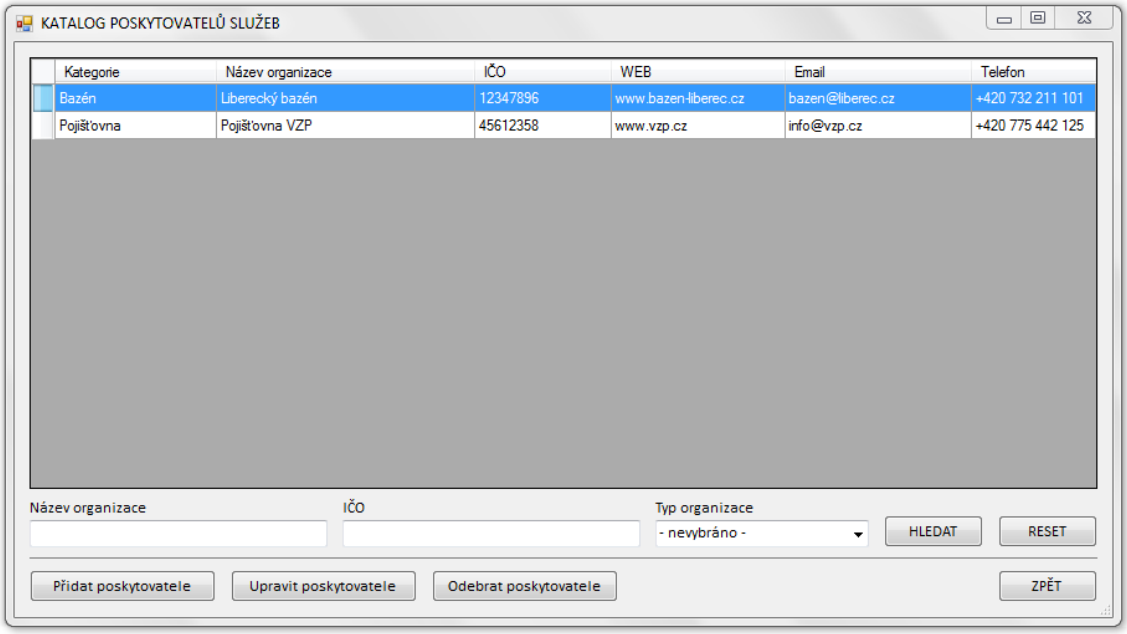
Obrázek 4.3 – Detail klienta

„Detail klienta“, Obrázek 4.3, slouží pro zobrazení veškerých údajů, které jsou známy. Lze zde najít aktuální počet nasbíraných bodů, které slouží pro čerpání slev, které poskytují spolupracující organizace. Detail dále slouží pro přidání bodů za vykonání určité služby. Například pokud klient splní požadovanou délku plavání v bazénu, lze mu přičíst body za plavání. Počet přidávaných bodů je určen konkrétní službou a je zobrazen v textovém poli, které se nachází v dolní části formuláře. Počet bodů nelze měnit. Službu je možno vybrat po kliknutí na tlačítko nacházející se vedle pole určeného pro název služby. Objeví se okno obsahující katalog služeb, ve kterém se dá vyhledávat pro rychlejší výběr. Seznam obsahuje pouze služby, které slouží pro získání bodů. Služby, které jsou pro využití nasbíraných kreditů, se v tomto seznamu nenachází. Jakmile si uživatel vybere požadovanou variantu, potvrdí výběr a posléze se vybraná možnost vyplní do textového pole. Po potvrzení přidělení bodů se změna množství okamžitě projeví na klientově aktuálním zůstatku. Tlačítko „Zobrazit zdravotní kartu“ slouží k propojení Softwaru pro zdravotní prevenci s databázovým uložištěm softwaru Praescriptor a k zobrazení zdravotních záznamů, které se v databázi

nacházejí. Pokud má uživatel uvedeno rodné číslo v Projektu podpory zdravotní prevence a existují zdravotní záznamy nasbírané softwarem Praescriptor, zobrazí se klientova zdravotní karta.

4.3 Modul POSKYTOVATELÉ SLUŽEB

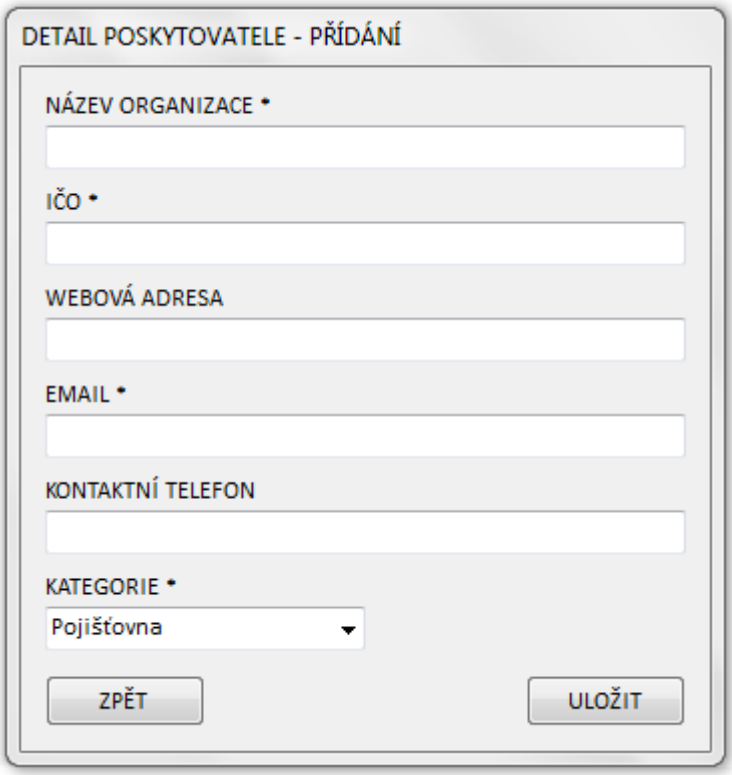
Tato část programu je určena pro administraci organizací, které budou spolupracovat na Projektu podpory zdravotní prevence. Nejedná se jen o organizace, které budou nabízet své služby, ale i o firmy, které budou poskytovat slevy na své výrobky. Mezi organizace se počítají i lékaři, kteří budou spolupracovat a budou ohodnocovat své pacienty body za splnění preventivní prohlídky či jiných lékařských úkonů.



Obrázek 4.5 – Katalog poskytovatelů služeb

První, co se zobrazí po otevření modulu pro poskytovatele služeb, je katalog (viz Obrázek 4.5). V něm lze vidět seznam všech spolupracujících organizací. Ty jsou v tabulce seřazeny podle názvu. Z tabulky lze vyčíst veškeré údaje o organizaci, což je kategorie, do které spadá, název, ičo, web a kontaktní údaje jako jsou email a telefon. V katalogu lze vyhledávat pomocí filtru, do kterého může uživatel vyplnit požadované hodnoty. Jedná se o název organizace, identifikační číslo organizace a typ organizace. Z katalogu lze otevřít formulářové okno pro přidání či úpravu poskytovatele. Stačí kliknout na jedno z tlačítek umístěných na spodní straně katalogového okna.

Po stisku tlačítka „Přidat poskytovatele“ se otevře nové okno, které slouží pro přidání organizace, či jiné spolupracující instituce. Formulář je zobrazen na obrázku. Po úspěšném vyplnění povinných textových polí, kterými jsou název organizace, ičo, email a kategorie, do které poskytovatel spadá, lze organizaci uložit. Pro obsáhlejší informace se mohou vyplnit i kolonky pro webovou adresu a kontaktní telefon. Po zdařilém uložení se objeví zpráva, že byla organizace úspěšně uložena a okno pro přidání se zavře.



Obrázek 4.6 – Pridani poskytovatele služeb

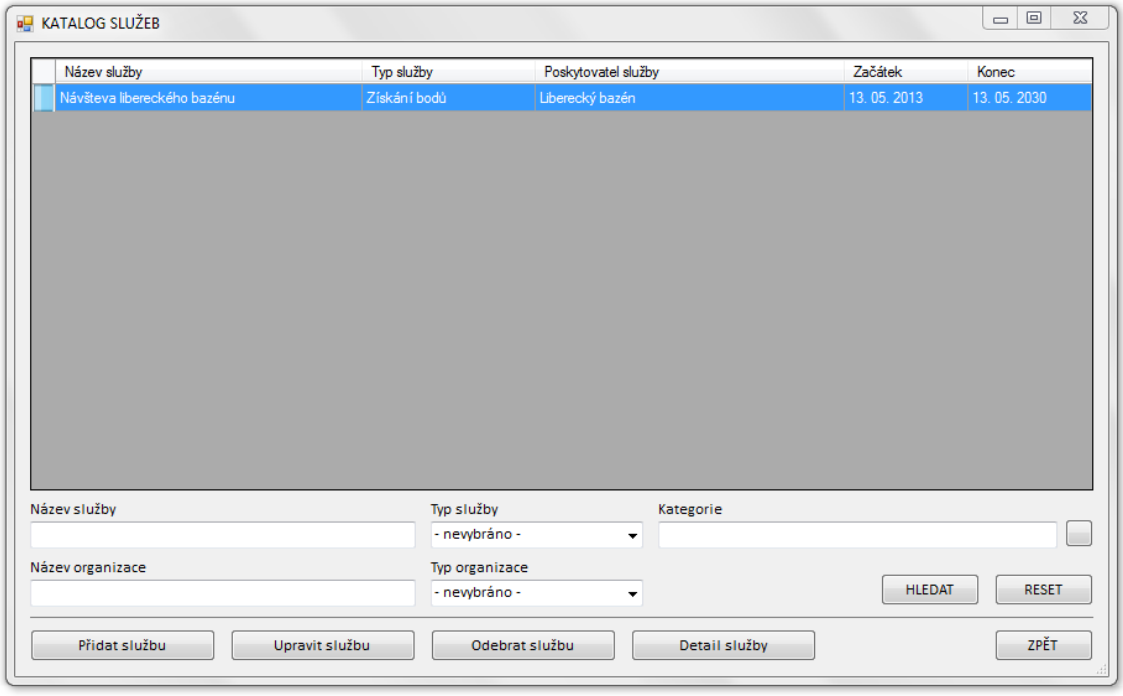
Pokud chce uživatel změnit některé údaje kteréhokoliv poskytovatele služeb, stačí si ho vybrat v tabulce, označit ho a kliknout na tlačítko „Upravit poskytovatele“. Otevře se nový pohled, který je stejný jako okno pro přidání (viz Obrázek 4.6). Rozdílem je, že už je vyplněný. Jsou v něm konkrétní údaje vybrané organizace, kterou se uživatel rozhodl změnit. Povinné atributy zůstávají shodné jako u přidání. Po správném naplnění textových polí se dá poskytovatel uložit.

Jako poslední možnost v katalogu je smazání vybraného poskytovatele. Uživatel vybere konkrétní organizaci a klikne na „Smazat poskytovatele“. Po potvrzení otázky, zda si je uživatel jistý svým výběrem, se poskytovatel služby smaže z katalogu.

4.4 Modul SLUŽBY

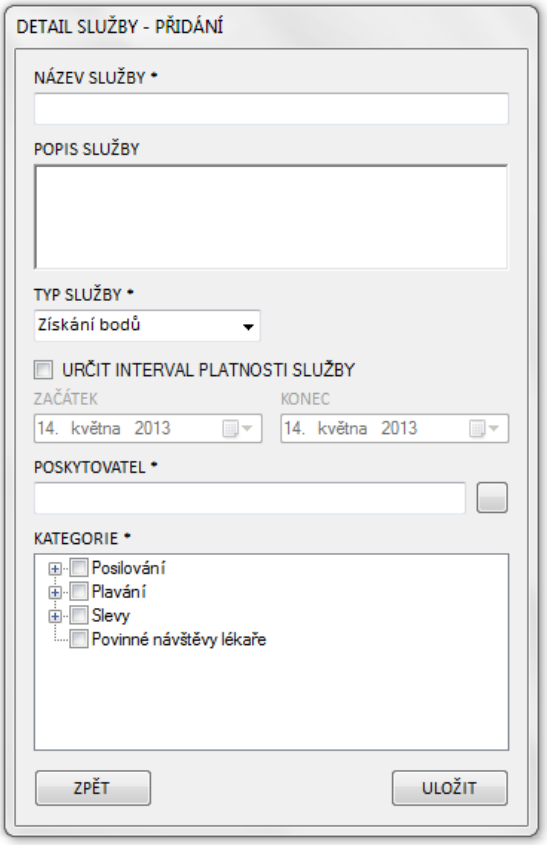
Modul služby slouží pro administraci služeb, které jsou poskytovány v rámci Projektu podpory zdravotní prevence spolupracujícími organizacemi. Neřadí se sem pouze služby pro získání bodů, jako jsou například plavání v bazénu nebo návštěva wellness centra, ale také například služba nabízející slevu na ortopedické pomůcky nebo léky na zhoršenou prostatu. Patří sem tedy vše, na co lze body využít nebo za co lze body získat.

Jako první, co se uživateli zobrazí po výběru administrace služeb, je katalog (viz Obrázek 4.7), ve kterém se nachází úplný výpis všech služeb, které jsou uloženy v PPZP. I zde je možnost vyhledat konkrétní službu pomocí navolených filtrů. Je možné najít požadovanou službu po zadání jejího názvu nebo po výběru typu, zda se jedná o službu pro získání či využití bodů. Dále se po kliknutí na tlačítko, nacházející se vedle textového pole určeného pro kategorie, otevře detail s výpisem všech kategorií, do kterých mohou spadat služby z PPZP. V tomto okně může uživatel zaškrtnout požadované kategorie, ve kterých chce, aby se služba nacházela. Poté potvrdí svůj výběr tlačítkem „Vybrat“. Posléze se uživatelův výběr vyplní do textového pole, kde jsou jednotlivé kategorie odděleny středníkem pro lepší orientaci. Následujícím kritériem může být zadání názvu organizace a konkrétního typu, který organizaci odpovídá.

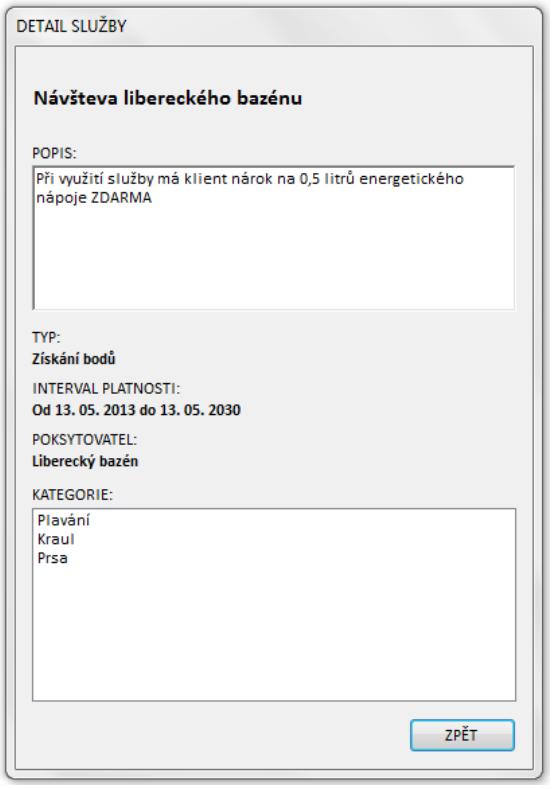


Obrázek 4.7 – Katalog služeb

Podobně jako u katalogů klientů a poskytovatelů služeb má uživatel i zde možnost přidat, upravit, smazat nebo zobrazit detail vybrané služby. Přidání nové položky je možné po klinutí na tlačítko „Přidat službu“. Objeví se nové okno, do kterého je možné vyplnit informace o nové službě (viz Obrázek 4.8). Mezi povinné atributy patří název služby, výběr typu služby, poskytovatel, který službu poskytuje a kategorie, do které služba spadá. Výběr poskytovatele je možný po poklepání myší na tlačítko umístěné vedle textového pole určené pro název poskytovatele. Zobrazí se katalog pro výběr, ve kterém se dá využít hledání pro rychlejší výběr. Po potvrzení výběru se ve formuláři pro přidání služby vyplní název a ičo poskytovatele. Pro výběr kategorie stačí zaškrtnout položky vyhovující požadovaným kritériím. Mezi nepovinné položky patří popis služby a určení intervalu platnosti od kdy a do kdy služba platí. Ten bude vyhovovat v případech, kdy půjde o akční nabídku nebo něco podobného. Pro dokončení úkonu přidání stačí stisknout „Uložit“. Pokud vše proběhne bez problémů, objeví se hláška, že vše proběhlo úspěšně.



Obrázek 4.8 – Přidání služby



Obrázek 4.9 – Detail služby

Pro případné změny je určen formulář, který je stejný jako pro přidání nové služby a je možné ho otevřít po zmáčknutí tlačítka „Upravit službu“. Výběr konkrétní

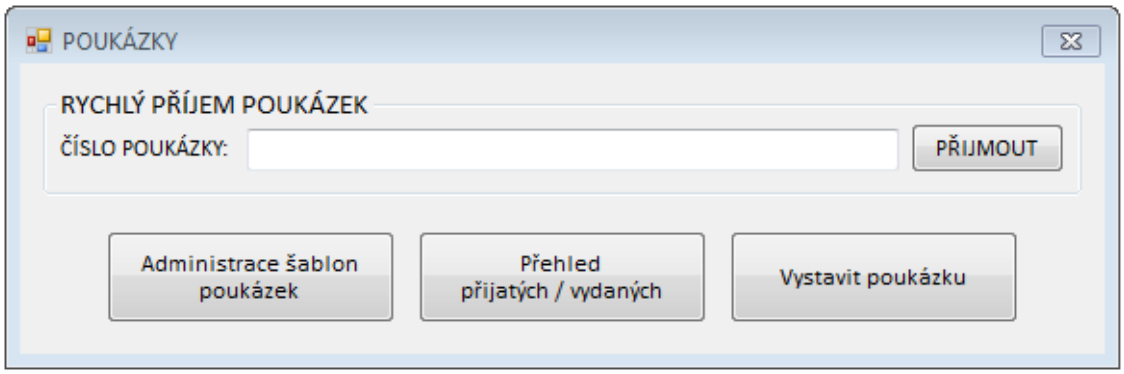
položky pro úpravu je možný označením konkrétního řádku v tabulce. Po výběru se otevře nové okno, které je shodné s formulářem pro přidání služby (viz Obrázek 4.8), a jsou v něm předvyplněné všechny zadané položky vybrané služby. Povinné atributy jsou shodné s přidáním a veškerá funkčnost tlačítek je obdobná. Pro potvrzení změny stačí klinout na „Uložit“.

Pro případné smazání nějaké služby je potřeba službu vybrat označením v tabulce a posléze zmáčknout „Odebrat službu“. Objeví se dotazovací dialog, který se ptá, zda si je uživatel jist s výběrem služby pro smazání. Pokud souhlasí, potvrdí výběr a služba se okamžitě odebere z katalogu.

„Detail služby“ (viz Obrázek 4.9) slouží pro zobrazení všech informací, které jsou uloženy v databázi. Ve výpisu jsou například k vidění všechny kategorie, do kterých služba spadá. Konkrétní službu, kterou chceme zobrazit, opět stačí vybrat označením v tabulce.

4.5 Modul POUKÁZKY

Modul poukázky je poněkud odlišný od ostatních segmentů. Je určen pro administraci voucherů a s tím se pojí několik dalších částí, které je potřeba řešit jednotlivě.

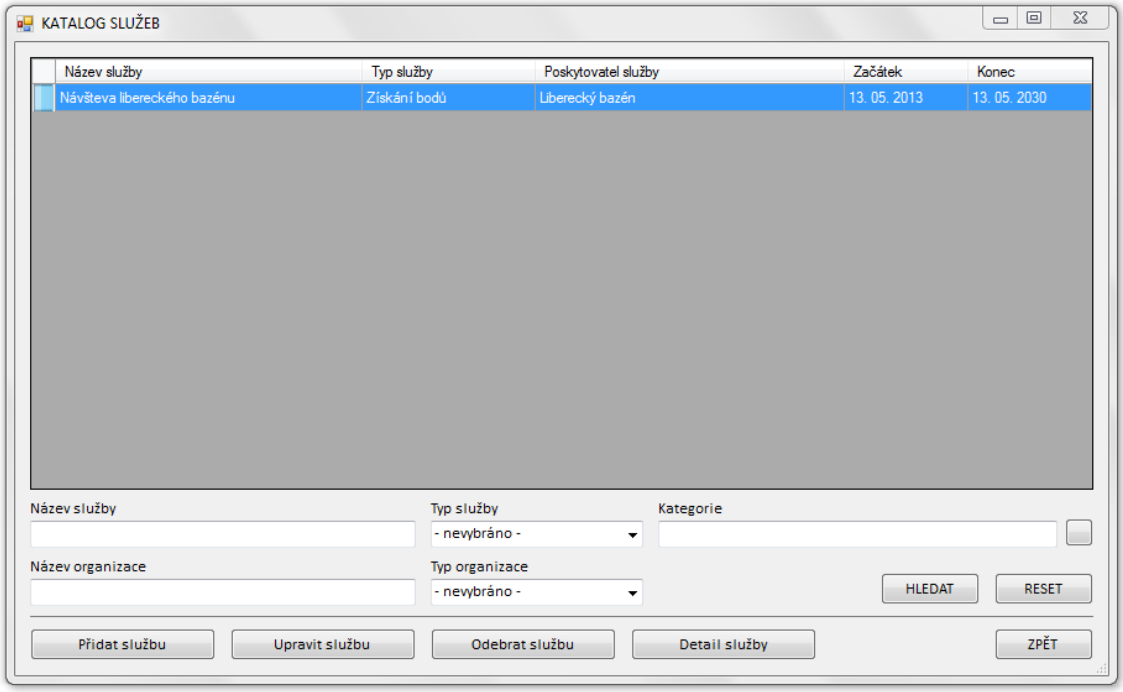


Obrázek 4.10 – Úvodní okno v modulu poukázek

První po výběru se neobjeví katalog jako v ostatních modulech, ale okno, které slouží pro další navigaci v programu (viz Obrázek 4.10). V horní části tohoto formuláře je možnost rychlého příjmu poukázky. Tento úkon je potřeba vyřešit ve chvíli, kdy dorazí klient s poukázkou. Stačí vyplnit do textového pole kód voucheru, který obsahuje, o jaký typ se jedná. Zda jde o poukázku pro využití nasbíraných bodů či slevový kupón, který napomáhá k získání nových klientů pro Projekt podpory zdravotní

prevence. Pro potvrzení kódu stačí stisknout tlačítko „Přijmout“ a poukázka se vzápětí stane neplatnou pro další použití. V dolní části okna se nacházejí tři tlačítka, která slouží pro administraci šablon poukázek, přehled přijatých a vydaných poukázek a pro vystavení nové poukázky.

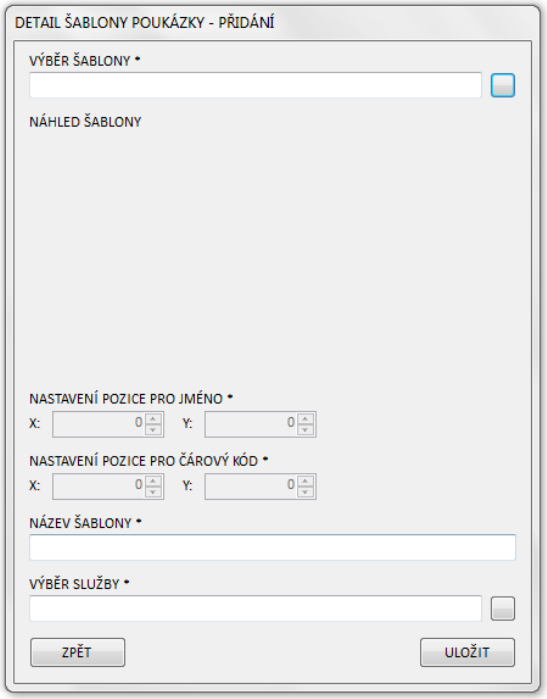
Část určená k administraci šablon slouží k vytvoření vzoru pro tištění poukázek. Po výběru této části, která se objeví po stisknutí tlačítka „Administrace šablon poukázek“, se otevře katalog (viz Obrázek 4.11). Je v něm vypsán seznam všech vytvořených vzorů. Záznamy v tabulce jsou seřazeny podle jediného viditelného sloupce, kterým je název šablony. Pod tabulkou se nachází filtr, určený pro hledání požadovaného záznamu. Stačí vyplnit hledaný název šablony a stisknout tlačítko „Hledat“. Pro znovunačtení dat slouží tlačítko „Reset“. Katalog obsahuje v dolní části okna tlačítka pro přidání, upravení a odebrání šablony.



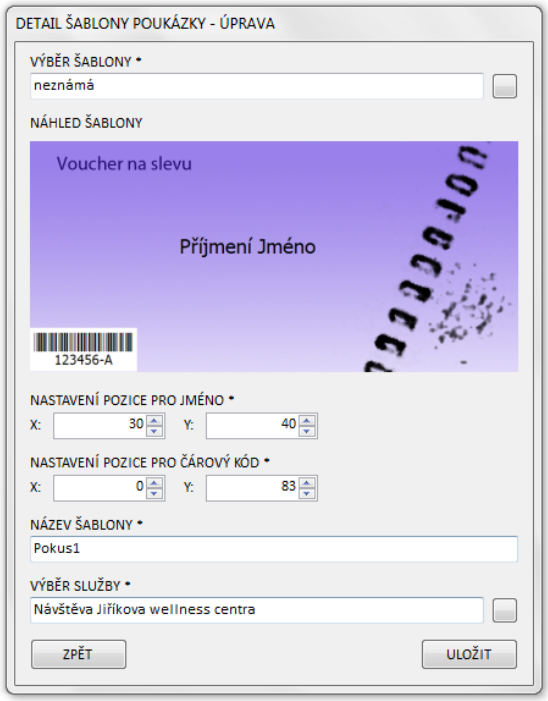
Obrázek 4.11 – Katalog šablon poukázek

Po stisknutí tlačítka „Přidat šablonu“ se zobrazí formulář s názvem „Detail šablony poukázky - přidání“, který slouží pro vytvoření nového vzoru pro budoucí poukázky (viz Obrázek 4.13). Každá organizace, která bude poskytovat nějaké slevy klientům v rámci využívání jejich nasbíraných bodů, si bude muset vytvořit vzhled poukázky včetně textů, které na ni chtějí mít. Tento vzor musí uložit do jednoho z formátů, které jsou v programu podporovány. Těmi jsou jpg, bmp a png. Takto

vytvořený obrázek lze vybrat jako šablonu, se kterou se v programu dále pracuje. K tomu slouží tlačítko umístěné vedle textového pole určeného pro zobrazení cesty k obrázku. Po stisknutí tlačítka se zobrazí klasické dialogové okno, ve kterém uživatel vybere požadovaný obrázek. Ve filtru lze vybrat jeden z podporovaných formátů pro rychlejší hledání. Po potvrzení výběru se vyplní požadovaná cesta do zmiňovaného pole a obrázek se zobrazí v náhledu umístěném pod ním. Tento náhled slouží k dalším možnostem při vytváření nové šablony. Dále se musí určit pozice dvou klíčových údajů, kterými jsou jméno klienta a čárový kód. Pro stanovení jejich pozice, která je podstatná pro tisk poukázek, slouží čtyři číselná pole, které po inkrementaci či dekrementaci hodnoty posouvají pozici údajů na náhledu. Změna hodnoty o jedna znamená posun o 1% původní výšky či šířky v obrázku. Po nastavení pozic už stačí vyplnit jen název šablony a vybrat službu, pro kterou je poukázka určena. Pro výběr služby opět slouží tlačítko umístěné vedle textového pole, po jehož stisknutí se otevře katalog obsahující seznam poskytovaných služeb. Pro rychlejší výběr je možné službu vyhledat podle jména. Po zvolení požadované služby se vyplní název do pole a lze novou šablonu úspěšně uložit. Pokud uložení proběhne v pořádku, zobrazí se upozornění, že vše proběhlo v pořádku a vzor se úspěšně uložil.



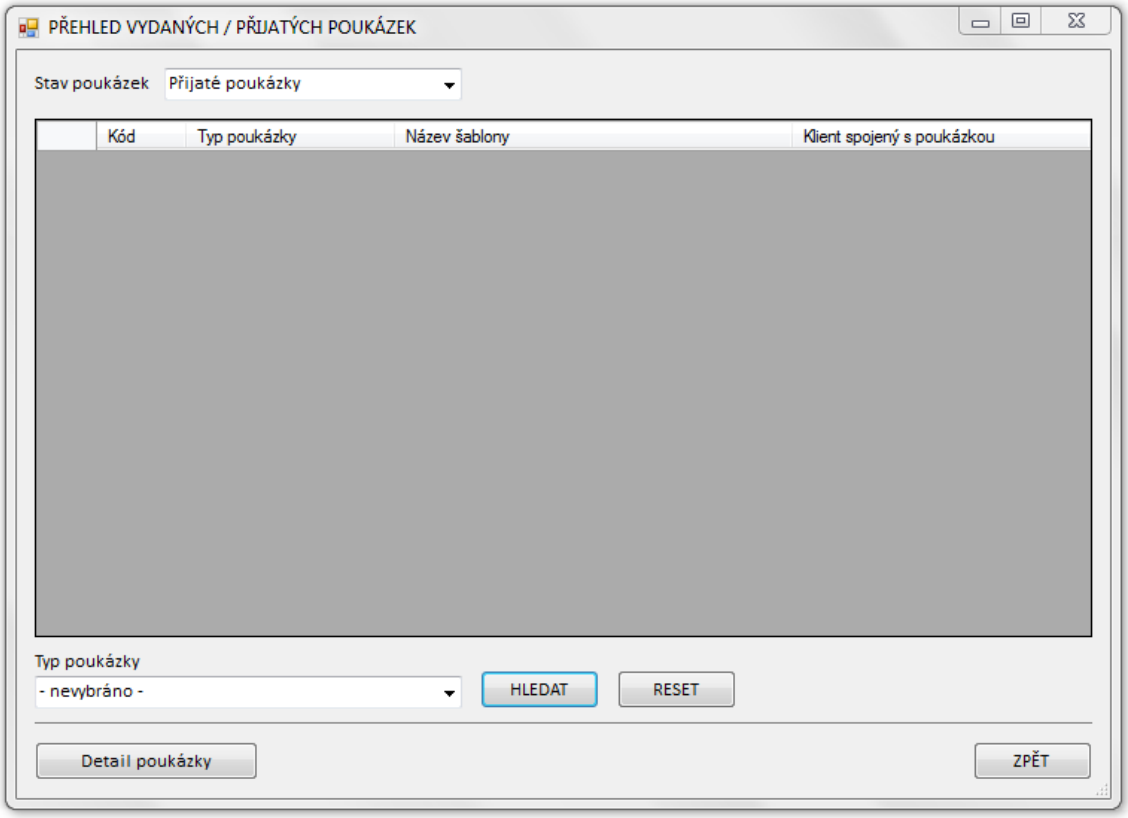
Obrázek 4.13 – Přidání šablony poukázek



Obrázek 4.12 – Úprava šablony poukázek

Nastane-li situace, že bude potřeba vytvořená šablona upravit, stačí ji označit v tabulce a kliknout na „Upravit šablonu“. Zobrazí se stejné okno, jako pro vytvoření nové šablony s tím rozdílem, že už je vyplněné hodnotami vybrané šablony (viz Obrázek 4.12). Náhled obrázku obsahuje i jméno a čárový kód na pozicích, které byly uloženy při vytváření šablony. Pozice lze upravovat. Jakmile uživatel upraví, co potřebuje, stačí potvrdit změnu tlačítkem „Uložit“ a změna šablony poukázky se ihned projeví například při novém tisku poukázek.

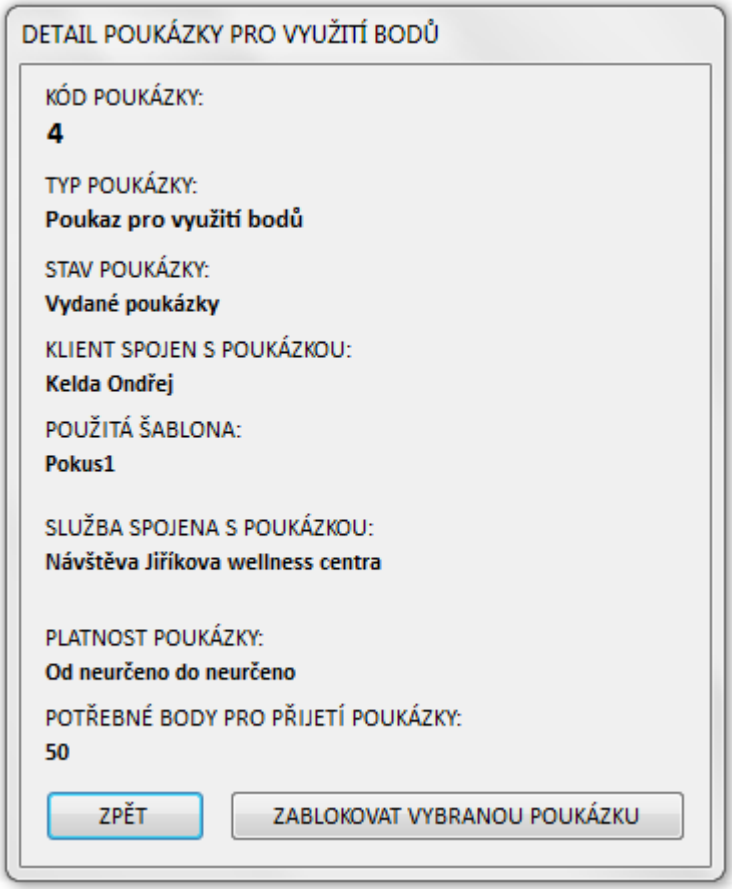
Jakmile se šablona stane nepotřebnou, lze ji z katalogu odebrat. Stačí stisknout „Odebrat šablonu“ a po potvrzení výběru se šablona odebere. Smazaný vzor už nelze vrátit. Tlačítko „Zpět“ slouží k uzavření formuláře s katalogem a k navrácení zpět k úvodnímu oknu v části poukázek.



Obrázek 4.14 – Katalog poukázek

V úvodním formuláři poukázek je dalším tlačítkem „Přehled přijatých / vydaných“. Po kliknutí na toto tlačítko se otevře okno, které slouží pro přehled přijatých i vydaných poukázek (viz Obrázek 4.14). V horním levém rohu formuláře se nachází komponenta, která slouží pro výběr výpisu. Uživatel může zvolit výpis přijatých poukázek, které už nejsou v oběhu a byly za ně připsány či odebrány body klientům

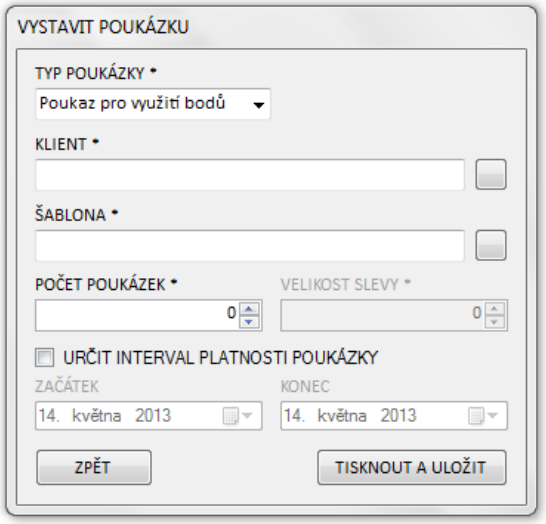
nebo výpis poukázek, které jsou stále mezi lidmi. Po zvolení požadovaného výpisu se vypsaná data dají dále filtrovat. Uživatel smí vybrat, zda bude výpis obsahovat jen slevové poukázky nebo poukázky, které mají klienti pro využití nasbíraných bodů. Pro zrušení filtru a znovu vypsání všech přijatých či vydaných poukázek slouží tlačítko „Reset“. V dolní části okna jsou dvě tlačítka. „Zpět“ je určeno k uzavření současného formuláře a k návratu do úvodního okna v segmentu určeného pro administraci poukázek. Druhým tlačítkem je „Detail poukázky“, které otevře náhled poukázky, kde jsou zobrazeny informace o poukázce (viz Obrázek 4.15). V informacích lze vyčíst kód poukázky, typ poukázky, stav poukázky, jaká byla použita šablona atd. Pokud se jedná o detail slevové poukázky, lze v informacích dále vyčíst velikost slevy, a kým byla poukázka vytvořena. Jedná-li se o poukázku pro využití nasbíraných bodů, je možné v detailu vyčíst, pro koho byla poukázka vytisknuta a kolik potřebuje mít na účtu bodů, aby mohl poukázku využít.



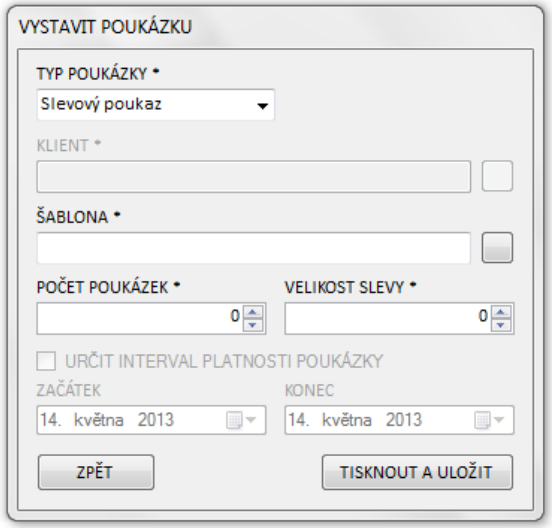
Obrázek 4.15 – Detail poukázly

Posledním tlačítkem nacházejícím se v dolní části úvodního formuláře modulu poukázek je tlačítko „Vystavit poukázku“. Po jeho stisknutí se otevře okno, které slouží

pro tisk poukázek (viz Obrázek 4.16 a Obrázek 4.17). Lze v něm tisknout oba druhy voucherů. Je tedy možné ho využít pro tisk poukázky na využití bodů pro konkrétního klienta nebo vytisknout celou sérii slevových poukázek například na ortopedické pomůcky. Prvním prvkem v okně je výběr typu poukázky pro tisk. Jde tedy vybrat ze dvou druhů. Poukaz pro využití bodů nebo slevový poukaz. Vybráním typu se ovlivní další funkčnost okna. Společným prvkem pro oba typy je výběr šablony. Po stisku tlačítka, které je vedle textového pole pro název šablony, se otevře formulář pro výběr šablony. Pro rychlejší volbu slouží možnost filtrování dat podle názvu. Po výběru šablony se do textového pole vyplní název. Dalším společným prvkem při tisku šablon pro oba typy poukázek je počet poukázek, který se má tisknout.



Obrázek 4.17 – Tisk poukázky pro využití bodů



Obrázek 4.16 – Tisk slevového voucheru

Co pro obě skupiny poukázek společné není, je výběr klienta. K tomu slouží tlačítko umístěné vedle textového pole určeného pro jméno vybraného klienta. V jednom případě jde o klienta, pro kterého je poukázka vystavena. Každý voucher pro využití bodů je totiž vázán na konkrétního klienta, aby při příjmu poukázky nedošlo k odebrání bodů někomu cizímu. U slevových poukázek je tomu jinak. Ty jsou vázány na majitele softwaru, který je vytiskl. Slevové vouchery slouží především k nalákání nových klientů do Projektu podpory zdravotní prevence. Dalším rozdílným prvkem je velikost slevy. Ta je určena pouze pro slevové poukázky. Interval platnosti poukázky, který se nachází v dolní části formuláře, je určen pouze pro vouchery sloužící k čerpání bodů. Tímto intervalem lze vymezit časové období, ve kterém musí klient poukázku využít. Pro dokončení tisku je určeno tlačítko „Tisknout a uložit“. Po jeho zmáčknutí se zkontroluje, zda je vše správně vyplněno. Jestliže je vše v pořádku, objeví se dialogové

okno, které slouží pro nastavení tisku. Jakmile uživatel nastaví potřebné parametry, stačí stisknout „OK“, poukázky se začnou ukládat na vzdálené úložiště a nedojde-li k žádné chybě, spustí se tisk voucherů. Pokud si uživatel tisk poukázek rozmyslí, stačí kliknout na tlačítko „Zpět“, aktuální formulář se uzavře a dojde k návratu do úvodního pohledu v modulu poukázek.

Závěr

Cílem bakalářské práce bylo vytvořit software pro Projekt podpory zdravotní prevence, který bude sloužit pro komunikaci, čili sběr dat a který bude tvořit prostředníka mezi zdravotními pojišťovnami, lékařstvím, lékárenstvím, wellness provozovateli a samotnými klienty.

Nejdříve bylo potřeba vybrat vhodnou technologii pro vývoj softwaru a následně vývojové prostředí, které tuto technologii podporuje. Má volba padla na programovací jazyk C# a platformu .NET, pro které je vhodným vývojovým prostředím Microsoft Visual Studio.

Dalším a to velice důležitým úkolem bylo vytvořit návrh aplikace, který bude splňovat požadavky velkého projektu. Z tohoto důvodu jsem návrhu věnoval nejvíce času a použil jsem několik návrhových vzorů a pravidel objektového programování.

Po výběru vývojových nástrojů a po dokončení návrhu aplikace přišlo na řadu samotné programování. Aplikace byla rozdělena na čtyři základní části, které zajišťují správu klientů, organizací, služeb a poukázek. Pomocí repositářů bylo implementováno připojení a výměna dat se vzdáleným databázovým úložištěm a také s lékařsko-lékárenským softwarem, který umožňuje asociaci klienta PPZP a zdravotní karty uložené v systému Praescriptor.

Všechny body zadání byly úspěšně dokončeny a fungují. Vytvořený software se může nasadit do zkušebního provozu pro analýzu a sběr dat podporujících následný vývoj.

Seznam použité literatury

[1] ALBAHARI, Joseph, Ben ALBAHARI a Peter DRAYTON. *C# 5.0 in a nutshell*. 5th ed. Sebastopol: O'Reilly, 2012, 1042 p. ISBN 978-144-9320-102.

[2] ATWOOD, Jeff a Joel SPOLSKY. *Stack Overflow* [online]. rev 2013.5.9.686. © 2008-2013 [cit. 2013-05-10]. Dostupné z: http://stackoverflow.com

[3] MAUNDER, Chris a David CUNNINGHAM. *CodeProject* [online]. Web03 | 2.6.130501.1 | Last Updated 10 May 2013. © 1999-2013 [cit. 2013-05-10]. Dostupné z: http://www.codeproject.com/

[4] MICROSOFT. *Microsoft Developer Network* [online]. 2008 [cit. 2013-05-10]. Dostupné z: http://msdn.microsoft.com

[5] SHARP, John. *Microsoft Visual C# 2010: krok za krokem*. Vyd. 1. Brno: Computer Press, 2010, 696 s. ISBN 978-80-251-3147-3.

Přílohy

Příloha A

Obsah přiloženého CD:

- BalarskaPrace.pdf: Text bakalářské práce
- Navod.txt: Návod k zprovoznění aplikace
- bakalarska_prace_create_script.sql: Skript pro vytvoření databáze
- bakalarska_prace_insert_script.sql: Skript pro naplnění tabulek
- ZdrojoveKody: složka se zdrojovými kódy
- Aplikace: složka obsahující aplikaci