UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2024 Viacheslav Dmytrenko

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Monitoring a optimalizace vykázaných lékařských úkonů v domově důchodců

Bakalářská práce

2024 Viacheslav Dmytrenko

Univerzita Pardubice

Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Viacheslav Dmytrenko**
Osobní číslo:	**I21138**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Monitoring a optimalizace vykázaných lékařských úkonů v domově důchodců**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Konečná roční úhrada lékařských úkonů realizovaných v domovech důchodců se řídí smlouvou uzavřenou se zdravotní pojišťovnou. Měsíční úkony pojišťovna hradí průběžně dle bodů za provedené úkony. Na konci roku ale dochází ke korekci úhrad dle smlouvy, když se zohlední úkony z referenčního roku. Pro finální vyúčtování je nejdůležitějším parametrem úhrn měsíců za všechny klienty, kdy jim bylo poskytnuto ošetření. Ke krácení bodů dochází, pokud počet kumulovaných měsíců překročil kumulovaný počet těchto měsíců z referenčního roku.
Cílem práce je naprogramovat aplikaci, které umožní sledovat počty úkonů a ošetřených klientů. Databázová část bude obsahovat kartu klienta s údaji o době pobyt v zařízení, datumy ošetření, jejich kódy a počty bodů. Do aplikace bude zakomponován algoritmus optimalizace vykázaných ošetřených klientů v žádostí o měsíční úhrady ve čtvrtém kvartále tak, aby se zabránilo krácení ze strany pojišťovny.

Rozsah pracovní zprávy: **30-40**
Rozsah grafických prací: **5**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

WRÓBLEWSKI, Piotr. Algoritmy: datové struktury a programovací techniky. Vyd. 1. Brno: Computer Press, 2004. ISBN 80-251-0343-9.

ČESKO. Vyhláška č. 482/2021 Sb. Vyhláška, kterou se mění vyhláška č. 134/1998 Sb., kterou se vydává seznam zdravotních výkonů s bodovými hodnotami, ve znění pozdějších předpisů. In: Sbírka zákonů. 2021, částka 215, s. 6497-6536. Dostupné z: https://ftp.aspi.cz/opispdf/2021/215-2021.pdf

BRUTHASOVÁ, Daniela., ČERVENKOVÁ, Anna., JEŘÁBKOVA, Věra. Zmapování nejzávažnějších problémů ve financování ošetřovatelské péče v ústavech sociálních služeb, [online]. Dostupné z: https://katalog.vupsv.cz/Fulltext/vz_294.pdf. [cit. 2013-10-26].

KUČEROVÁ, Zdeňka. Úhrady zdravotní péče v ústavech sociálních služeb, [online]. Dostupné z: https://www.vzp.cz/poskytovatele/informace-pro-praxi/poradna/uhrada-zdravotnich-sluzeb-vpobytovych-zarizenich-socialnich-sluzeb. [cit. 2013-10-26].

Vedoucí bakalářské práce: **Mgr. Jaroslav Marek, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2024

Prohlášení autora

Prohlašuji:

Práci s názvem Monitoring a optimalizace vykázaných lékařských úkonů v domově důchodců jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 07. 05. 2024

Viacheslav Dmytrenko v.r.

PODĚKOVÁNÍ

Tady chtěl bych poděkovat svému vedoucímu Mgr. Jaroslavu Markovi, Ph.D za hezké vedení bakalářské práce a rychle odpovědi. Také rád bych poděkoval svým rodičům a kamarádům za jejich podporu po celou dobu studia.

ANOTACE

Cílem práce je naprogramovat aplikaci, které umožní sledovat počty úkonů a ošetřených klientů. Databázová část bude obsahovat kartu klienta s údaji o době pobytu v zařízení, datumy ošetření, jejich kódy a počty bodů. Do aplikace bude zakomponován algoritmus optimalizace vykázaných ošetřených klientů v žádostí o měsíční úhrady ve čtvrtém kvartále tak, aby se zabránilo krácení ze strany pojišťovny.

KLÍČOVÁ SLOVA

C# .NET, MVVM, MAUI, optimalizace, SQL, desktopová aplikace, body za lékařské úkony, financovaní zdravotních úkonů.

TITLE

Monitoring and optimization of reported medical procedures in the retirement home.

ANNOTATION

The purpose of the work is to program an application that allows to track the number of procedures and clients treated. The database part will contain the client's card with information about the time of stay in the facility, the dates of treatment, their codes and the number of points. An algorithm will be incorporated into the application to optimize the reported treated clients in the fourth quarter monthly reimbursement claims to avoid underpayment by the insurance company.

KEYWORDS

C# .NET, MVVM, MAUI, optimization, SQL, desktop application, points for medical procedures, financing of medical procedures.

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

API – Application Programming Interface

BCL – Base Class Library

CIL – Common Intermediate Language

CLR – Common Language Runtime

DBMS – Database Management Systems

DI – Dependency injection

ERD – Entity relationship diagram

IDE – Integrated Development Environment

JIT – Just-In-Time

MAUI – Multi-platform App UI

MSIL – Microsoft Intermediate Language

MVVM – Model-View-ViewModel

OOP – Object-oriented programming

ORM – Object Relational Mapping

OS – Operating system

SQL – Structured query language

SQLite – Structured Query Language Lite

UI – User Interface

VCS – Version Control System

XAML – Extensible Application Markup Language

OS – Operační systém

VZP – Všeobecná zdravotní pojišťovna

TERMINOLOGIE

Framework: sada nástrojů a knihoven, která usnadňuje proces vývoje software poskytováním struktury, standardizovaných postupů a základních funkcionalit, čímž umožňuje rychlejší a efektivnější implementaci úkolů bez opakovaného vytváření kódu.

Relační databáze: typ databáze, ve kterých informace je ukládána v tabulkách s definovanými vztahy mezi nimi, jako je například vazba jedna k jedné, jedna k mnoha, nebo mnoho k mnoha, umožňující efektivní ukládání, vyhledávání a manipulaci s daty, často využívaný pro ukládání a správu dat v různých aplikacích.

Programovací paradigma: základní principy a metody, které slouží k organizace kódu a definují specifický přístup k psaní softwaru v rámci konkrétního programovacího jazyka nebo platformy.

Uživatelské rozhraní (UI): představuje vizuální a interaktivní část aplikace, která umožňuje uživatelům komunikovat s programem prostřednictvím grafických prvků, jako jsou tlačítka, formuláře a menu.

Integrované vývojové prostředí (IDE): je softwarový nástroj, který kombinuje různé nástroje pro tvorbu programů. IDE zahrnuje editor kódu, kompilátor, ladicí program, systém pro správu verzí a další nástroje potřebné k vytváření programů čím usnadňuje proces vývoje a zvyšuje efektivitu programátora.

Systémy pro správu verzí (VCS): nástroje, které poskytují kontrolu nad změnami kódu, sledují historii změn, umožňují spolupráci mezi vývojáři a v případě potřeby obnovují předchozí verze kódu.

ÚVOD

V domovech důchodců v České republice se úhrady za lékařské úkony řídí smlouvou uzavřenou se zdravotní pojišťovnou [1]. Měsíční úkony jsou hrazeny průběžně podle bodů za provedené úkony, a na konci roku probíhá korekce úhrad dle smlouvy, kde jsou zohledněny úkony z referenčního roku. Financování zdravotní péče v léčebnách dlouhodobě nemocných a domovech důchodců je upraveno vyhláškou Č. 482/2021 [2]. Ve vyhlášce je popsáno, jak se spočítá bodová hodnota různých výkonů. Pro finální vyúčtování je klíčovým parametrem úhrn měsíců za všechny klienty, kterým bylo poskytnuto ošetření. Pokud počet kumulovaných měsíců překročí kumulovaný počet z referenčního roku, dochází ke krácení bodů.

Bakalářská práce se zaměřuje na vytvoření aplikace pro monitorování a optimalizaci vykázaných lékařských úkonů v domovech důchodců. Práce je rozdělena na dvě hlavní části: teoretickou a praktickou.

V teoretické části této bakalářské práce jsou podrobně popsány různé technologie a nástroje které byly použity pro vývoj aplikace. Konkrétně se jedná o jazyky C#, SQL a XAML. C# je zde uveden jako hlavní programovací jazyk pro aplikace, SQL jako jazyk pro práci s databázemi, a XAML jako jazyk pro definování uživatelského rozhraní.

Další část teoretické části se zaměřuje na popis použité databázové technologie SQLite a vývojových prostředí, která byla použita pro vývoj této aplikace. Zmíněny jsou nástroje jako Microsoft Visual Studio, SQLiteStudio a Oracle SQL Developer Data Modeler. Zároveň je zde vysvětlen pojem "framework" a jeho význam v kontextu vývoje aplikace, s důrazem na konkrétní framework MAUI, který byl použit pro vývoj uživatelského rozhraní.

Následuje detailní popis architektury MVVM (Model-View-ViewModel), která byla použita při vývoji této bakalářské práce. Architektura MVVM je klíčová pro oddělení logiky uživatelského rozhraní od obchodní logiky, což usnadňuje testování a údržbu aplikace. V této části je také popsána knihovna SQLite-net-pcl, která slouží jako ORM (Object-Relational Mapping) nástroj, což umožňuje snadnou a efektivní manipulaci s daty v relační databázi z perspektivy objektově orientovaného programování. Tato část také zmiňuje další použité knihovny, které přispívají k funkcionalitě aplikace.

Závěrečná část teoretické části se zaměřuje na matematický algoritmus, který byl použít pro optimalizaci vykázaných lékařských úkonů v prostředí domovů důchodců. Tento algoritmus je

klíčový pro zvýšení efektivity a přesnosti zpracování lékařských záznamů a přispívá k celkovému úspěchu aplikace v daném sektoru.

V praktické části bakalářské práce je popsaná struktura databáze, souborová struktura včetně jednotlivých tříd a je popsáno uživatelské rozhraní (UI), které jsou klíčové pro fungování aplikace. Aplikace je navržena tak, aby umožňovala uživatelům přidávat, upravovat a mazat data, která jsou následně ukládána do relační databáze. Celková struktura aplikace a její funkčnost jsou pečlivě popsány s cílem poskytnout ucelený pohled na implementaci a praktické využití navrženého řešení s pomocí programovacího jazyky C# a databázové technologie SQLite.

1 TEORETICKÁ ČÁST

1.1 Cíle aplikace

Cílem aplikace je sledování a optimalizace lékařských úkonů prováděných v domovech důchodců. Databáze bude obsahovat informace o klientech, jako je doba pobytu, datumy ošetření, kódy úkonů a jejich počty. Aplikace také zahrne algoritmus pro optimalizaci vykázaných ošetření klientů v žádostech o měsíční úhrady, aby se minimalizovala možnost krácení ze strany pojišťovny.

1.2 Evidence bodů za zdravotní péči a obsah aplikace

1.2.1 Body za zdravotní péči

Zdravotní pojišťovna platí za měsíční úkony na základě bodů, které jsou přiřazeny každému úkonu. Tento systém financování zdravotní péče je založen na bodovém ohodnocení poskytnutých lékařských služeb, kde každý lékařský úkon má určitý počet bodů, kterými je vyjádřena jeho hodnota. Tyto body jsou pak převedeny na finanční částku, kterou zdravotní pojišťovna hradí za poskytnuté služby. Každoročně dochází k revizi a korekci úhrad, která zahrnuje úkony z předchozího roku a je prováděna na základě aktualizovaných cenových a bodových hodnot. Na obrázku 1 je zobrazena bodová hodnota úkonů aplikace léků neinvazivní cestou. Tato tabulka poskytuje přehled o jednotlivých léčivech, jejich množství, jednotkové ceně a přiřazených bodech. Je důležité, aby bodová hodnota každého úkonu byla správně stanovena, aby byla zajištěna spravedlivá a transparentní úhrada poskytnutých služeb.

Kód	Název	Doplněk	Množství	Jednotka	Cena	DPH	Body
A000284	pinzeta jednorázová		0,1	1ks	8,20	0,00 %	0,82
02054	STŘÍKAČKA INJ PH 20ML		0,1	ks	2,90		0,29
A000283	ústní lžičky dřevěné		0,5	1ks	0,62	0,00 %	0,31
				Celkem:	**11,72**		**1,42**

Obrázek 1: Bodová hodnota výkonu aplikace léků neinvazivní cestou [3].

1.2.2 Správa lékařských úkonů

V rámci správy lékařských úkonů aplikace umožňuje přidávání nových úkonů, které mohou být prováděny v domovech důchodců. Administrátoři systému mají možnost editovat již existující úkony a také je odstranit v případě potřeby. Tato funkcionalita slouží k efektivnímu sledování a správě všech lékařských úkonů poskytovaných v domovech důchodců, což přispívá ke zlepšení celkové kvality poskytované péče těmto klientům.

1.2.3 Správa ošetřených klientů

V aplikaci je dostupná funkcionalita pro přidávání nových záznamů o ošetřených klientech, úpravu stávajících záznamů a jejich odstranění. Kromě toho, aplikace umožňuje sledování provedených lékařských úkonů u jednotlivých klientů, což usnadňuje sledování a správu poskytované zdravotní péče.

1.2.4 Správa uživatelských rolí a oprávnění

V aplikaci jsou definovány různé uživatelské role s odpovídajícími oprávněními. Administrátor má plný přístup ke všem funkcím aplikace a může provádět jakékoli úpravy a správu dat. Pracovníci mají omezený přístup a mohou pouze přidávat informace o provedených úkonech k jednotlivým klientům.

1.3 Programovací jazyky

1.3.1 Jazyk C#

C# je moderní vysokoúrovňový objektově orientovaný programovací jazyk který vyvinula společností Microsoft v roce 2000 [4]. Jeho vývoj vychází z programovacích jazyků C++ a Java, což mu poskytuje robustní základy a umožňuje efektivní práci s objekty a daty. Kvůli tomu, že C++ je založen na programovacím jazyku C, lze říct, že C# dědí některé charakteristiky a koncepty z jazyka C, což mu dodává flexibilitu a sílu.

Kód napsaný v jazyce C# prochází procesem kompilace pomocí CLR (Common Language Runtime), což je součástí širší platformy .NET. Tato platforma poskytuje prostředí, ve kterém může být kód C# spouštěn a vykonáván. Během kompilace je kód převeden do mezipřekladového jazyka, který se nazývá MSIL (Microsoft Intermediate Language) nebo CIL (Common Intermediate Language). Tento mezipřekladový jazyk je nezávislý na konkrétní platformě, což znamená, že může být použit na různých operačních systémech a hardware. Poté, když je program spuštěn, mezipřekladový jazyk je interpretován nebo předán JIT (Just-In-Time) kompilátoru. JIT kompilátor je součástí CLR a jeho úkolem je převést mezipřekladový jazyk do strojového kódu, který je specifický pro konkrétní hardware a operační systém, na kterém program je spouštěn. Tento proces se nazývá JIT kompilace a umožňuje efektivní provádění kódu C# na různých zařízeních [5].

Celkově platí, že platforma .NET, včetně CLR a JIT kompilátoru, hraje klíčovou roli v procesu kompilace a provádění kódu v jazyce C#. Bez těchto komponent by nebylo možné spouštět a provádět aplikace napsané v tomto jazyce na různých zařízeních a platformách.

C# podporuje několik programovacích paradigmat, včetně objektově orientovaného programování (OOP), které umožňuje organizovat kód do objektů a tříd, což umožňuje pracovat s nimi jako s jednotkami dat. Tímto způsobem je C# schopen abstrahovat a opakovaně používat kód, což usnadňuje vývoj a údržbu složitých softwarových systémů. Kromě toho C# podporuje i další paradigma, jako je například strukturované programování, které umožňuje psát kód s jasně definovanými kontrolními strukturami, jako jsou podmínky a cykly, což zlepšuje čitelnost a údržbu kódu.

Jazyk C# neustále aktualizuje a rozšiřuje své možnosti, což umožňuje vývojářům pracovat s nejnovějšími technologiemi a trendy v softwarovém průmyslu. Díky své široké použitelnosti a podpoře ze strany Microsoftu je C# jedním z nejpopulárnějších programovacích jazyků ve světě softwarového vývoje. Je využíván pro širokou škálu aplikací, včetně desktopových aplikací, webových aplikací, mobilních aplikací, herního vývoje a dalších. Vývojáři mohou využívat bohaté knihovny a frameworky dostupné pro C#, což usnadňuje a urychluje vývoj nových aplikací a projektů. Díky svým vlastnostem a silným základům je C# stále jedním z preferovaných jazyků pro vývoj moderního softwaru.

1.3.2 Jazyk SQL

SQL (Structured Query Language) je standardizovaný databázový programovací jazyk, který byl vyvinut společností IBM v roce 1970 jako součást projektu pro práci s velkými daty a od té doby se stal jedním z nejvíce používaných jazyků pro práci s databázemi [6]. Jeho flexibilita a jednoduchost syntaxe umožňují efektivní manipulaci s daty bez ohledu na velikost databáze či složitost operací. SQL operace jsou rozděleny do několika skupin podle typu operace [7]:

- Data Definition Language (DDL) – tato skupina operací slouží k definici datové struktury databáze, což zahrnuje vytváření, úpravu a mazání objektů databáze, jako jsou tabulky, trigery, funkce a další.
- Data Manipulation Language (DML) – tyto operace umožňují manipulaci s daty v databázi, jako je získávání, ukládání a mazání dat.
- Data Control Language (DCL) – zaměřují se na správu uživatelských rolí a práv k přístupu a manipulaci s daty v databázi.
- Data Transaction Language (DTL) – tato skupina operací je určena pro správu databázových transakcí, které zajišťují konzistenci dat a integritu databáze během provádění transakcí.

- Kromě toho, SQL příkazy jsou prováděny pomocí databázových systémů, které interpretují a vykonávají tyto příkazy. Zapojení se obvykle skládá z připojení k databázovému serveru pomocí specifického připojovacího řetězce a poslání dotazu v jazyce SQL. Databázový server pak interpretuje dotaz, provádí potřebné operace nad databází a vrací výsledky zpět klientovi.

Provedení SQL příkazů probíhá prostřednictvím databázových systémů, které interpretují a vykonávají tyto příkazy. Zapojení se obvykle skládá z připojení k databázovému serveru pomocí specifického připojovacího řetězce a odeslání dotazu v jazyce SQL. Databázový server pak interpretuje dotaz, provádí potřebné operace nad databází a vrací výsledky zpět klientovi. Proces provedení SQL příkazů může zahrnovat různé kroky, jako je analýza dotazu, plánování vykonání, provedení operací nad daty a návrat výsledků.

1.3.4 Jazyk XAML

XAML (Extensible Application Markup Language), byl vytvořen společností Microsoft v roce 2006 [8]. Jedná se o deklarativní značkovací jazyk používaný v různých technologiích vývoje softwaru, jako jsou aplikace pro Windows, webové aplikace a mobilní aplikace.

XAML umožňuje vývojářům vytvářet uživatelská rozhraní (UI) oddělením jejich návrhu od logiky aplikace. Popisuje prvky uživatelského rozhraní a jejich vztahy pomocí značek, díky čemuž je kód čitelnější a přehlednější. Tento přístup zvyšuje efektivitu vývoje softwaru a usnadňuje údržbu aplikací

Příklad syntaxe XAML pro vytvoření tlačítka v aplikaci Windows:

<Button Text="Klikni" Width="100" Height="50"/>

V tomto příkladu je *<Button>* značka, která definuje prvek tlačítka. *Text*, *Width* a *Height* jsou atributy, které určují text, šířku a výšku tlačítka.

Jednou z vlastností jazyka XAML je jeho kompilace do binárního formátu *.baml* (Binary XAML) [9]. Soubory *.baml* obsahují zkompilované značky XAML a používají se v procesu nasazení aplikace. Kompilace jazyka XAML do formátu *.baml* urychluje načítání uživatelského rozhraní aplikace a snižuje zatížení systému.

Jazyk XAML se používá v technologii UWP (Universal Windows Platform) k vytváření uživatelského rozhraní aplikací. V UWP definuje XAML strukturu uživatelského rozhraní a logiku aplikace lze napsat v programovacích jazycích, jako je C# nebo VB.NET [10].

XAML se používá při vývoji webových aplikací pomocí technologie Silverlight, je široce používán v kombinaci s platformami jako je WPF (Windows Presentation Foundation), MAUI (Multi-Platform App UI) a Xamarin pro vývoj nativních aplikací pro různá zařízení a operační systémy, včetně mobilních zařízení. XAML má výkonné možnosti stylování a animace, což vývojářům umožňuje vytvářet atraktivní a interaktivní uživatelská rozhraní.

1.4 Databázové technologie

1.4.1 SQLite

SQLite je vestavěná databáze napsaná v jazyce C, což zajišťuje vysokou rychlost a efektivitu při manipulaci s daty. Tato technologie funguje jako knihovna pro správu dat a systém řízení databází (DBMS), což umožňuje ukládání informací v jediném souboru na disku bez potřeby externího serveru. Byla vyvinuta v roce 2000 D. Richardem Hippem [11]. SQLite se vyznačuje kompaktností a efektivitou, což jej činí ideální volbou pro aplikace, které vyžadují lokální ukládání dat. Tato technologie se těší širokému využití ve světě softwarového vývoje a je podporována na mnoha platformách, včetně Windows, macOS, Linux, iOS a Android. SQLite je často preferovanou volbou pro mobilní aplikace, desktopové programy, webové prohlížeče a další softwarové aplikace, kde je nezbytná efektivní správa dat [12]. SQLite poskytuje spolehlivé a robustní prostředí pro ukládání a manipulaci s daty bez nutnosti složitých konfigurací a externích serverů, což přispívá k jeho popularitě a širokému rozšíření ve světě softwarového vývoje.

1.5 Vývojové prostředí

1.5.1 Microsoft Visual Studio

Microsoft Visual Studio je integrované vývojové prostředí (IDE) vyvinuté společností Microsoft pro vytváření různých aplikací, webových stránek, mobilních aplikací, her a dalších. Microsoft Visual Studio poskytuje vývojářům výkonné nástroje a prostředky pro vytváření, ladění a nasazování softwaru na různých platformách.

Jednou z klíčových vlastností aplikace Microsoft Visual Studio je její multifunkčnost. Prostředí obsahuje různé nástroje a komponenty, které vývojářům umožňují pracovat s různými programovacími jazyky, jako jsou C#, C++, Visual Basic, F# a Python. Díky tomu si mohou vývojáři vybrat nejvhodnější jazyk pro svůj projekt a snadno mezi nimi přepínat v rámci jednoho projektu.

Visual Studio podporuje širokou škálu systémů pro správu verzí (VCS), včetně populárních systémů jako Git, Subversion, Mercurial a Team Foundation Version Control. Vývojáři

si mohou vybrat nejvhodnější systém správy verzí podle svých potřeb a preferencí. Každý z těchto systémů VCS je integrován do vývojového prostředí Visual Studio, což umožňuje snadné používání a efektivní správu verzí kódu v rámci projektů.

Další výhodou aplikace Visual Studio je její integrace s ostatními produkty a službami společnosti Microsoft. Vývojáři mohou snadno integrovat své projekty s cloudovými službami Microsoft Azure, používat nástroje pro vývoj a testování aplikací pro platformu Windows, vytvářet aplikace pro zařízení se systémem Windows a vyvíjet a ladit aplikace pro platformu .NET. Díky podpoře emulátorů mohou vývojáři vytvářet aplikace pro různé platformy.

1.5.2 SQLiteStudio

SQLiteStudio je výkonným multiplatformním nástrojem pro práci s databázemi SQLite. Nabízí širokou škálu funkcí umožňujících správu, úpravu a analýzu SQLite databází. Jednou z hlavních výhod SQLiteStudio je jeho intuitivní uživatelské rozhraní, které zjednodušuje práci s databázemi a zvyšuje efektivitu.

Aplikace podporuje různé typy dat v SQLite, včetně textových, číselných, datových a časových dat, stejně jako BLOB dat. Umožňuje také provádět SQL dotazy přímo z rozhraní aplikace, což usnadňuje analýzu dat a provádění různých operací.

SQLiteStudio umožňuje zobrazení schématu databáze, úpravu tabulek, přidávání a odstraňování záznamů, zálohování a obnovování dat a provádění různých analytických operací, jako je filtrování, třídění a seskupování dat. Díky své široké funkcionalitě je SQLiteStudio nenahraditelným nástrojem pro vývojáře, správce databází a všechny, kdo pracují s SQLite databázemi různých velikostí a složitostí.

1.5.3 Oracle SQL Developer Data Modeler

Oracle SQL Developer Data Modeler je univerzální program pro návrh, modelování a dokumentaci databází, který společnost Oracle vytvořila v roce 2009. Tento nástroj poskytuje mnoho funkcí pro vývoj a správu databázových schémat. Je možné jej použít k vytváření logických i fyzických datových modelů, které umožňují vizualizovat strukturu databáze, vztahy mezi tabulkami a další důležité součásti.

Oracle SQL Developer Data Modeler podporuje řadu standardů a umožňuje vytvářet diagramy entit a vztahů (ERD), definovat tabulky, sloupce, datové typy, indexy, vztahy a klíče. Nástroj nabízí také možnosti reverzního inženýrství, které umožňují odvodit modely

z existujících databází a analyzovat a optimalizovat je. Data Modeler se navíc integruje s dalšími produkty Oracle, což zjednodušuje pracovní postupy a zajišťuje konzistenci dat.

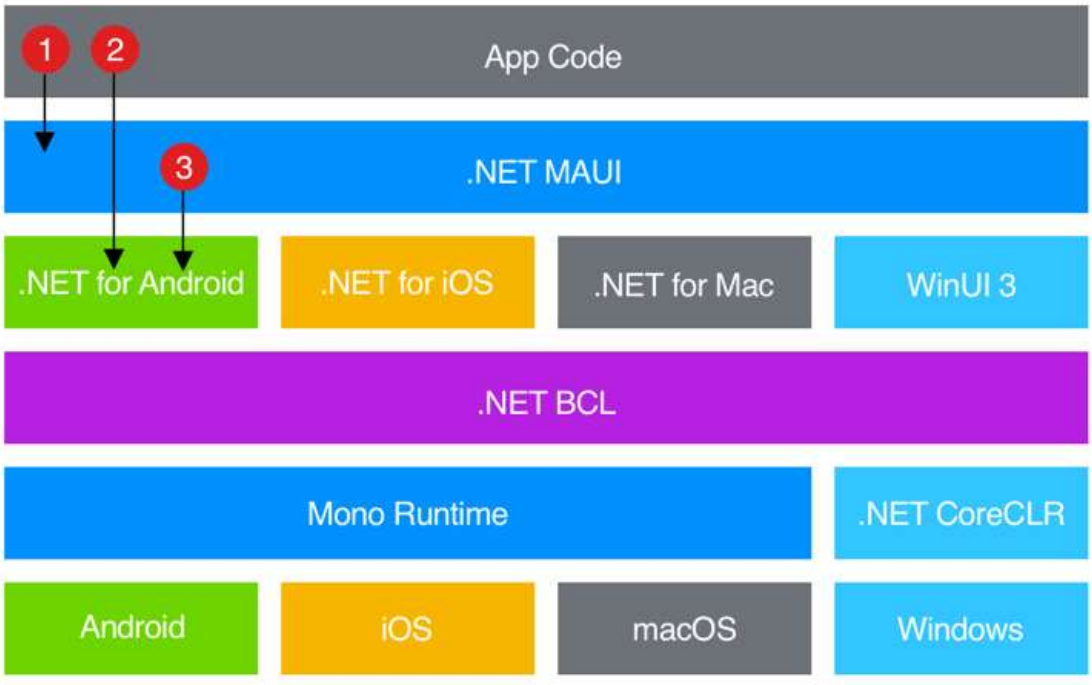
1.6 Frameworky

1.6.1 MAUI

.NET MAUI (Multi-Platform App UI) je moderní framework vyvinutý společností Microsoft v roce 2022 pro vytváření aplikací, které je možné spouštět na různých operačních systémech (OS), jako je Android, iOS, macOS a Windows [13]. MAUI umožňuje vývojářům psát kód v jazyce C# a XAML, který lze použít na všech těchto platformách, což značně usnadňuje a zefektivňuje proces vývoje.

Rozhraní .NET MAUI spojuje rozhraní API pro systémy Android, iOS, macOS a Windows do jediného rozhraní .NET MAUI API a poskytuje přístup ke všem aspektům každé platformy prostřednictvím jednoho společného vstupu.

Níže si můžete prohlédnout podrobný pohled na architekturu aplikace napsané v rozhraní .NET MAUI:



Obrázek 2: Architektura aplikace v .NET MAUI [14].

Při spouštění aplikace napsané s využitím .NET MAUI kód aplikace komunikuje s rozhraním .NET MAUI API (obr. 2).

Toto API následně volá odpovídající rozhraní .NET API pro konkrétní OS, například .NET Android, .NET iOS, .NET macOS nebo WinUI 3, v závislosti na cílové platformě (obr. 2).

V případě potřeby může kód komunikovat přímo s rozhraním API operačního systému bez volání rozhraní API .NET MAUI (obr. 2).

.NET API pro všechny OS využívá základní knihovnu tříd .NET (BCL). BCL umožňuje abstrahovat detaily základní platformy od kódu, což umožňuje sdílení společné obchodní logiky mezi různými aplikacemi. BCL závisí na běhovém prostředí .NET, ve kterém je kód aplikace spuštěn. Například systémy Android, iOS a macOS používají prostředí Mono Runtime, zatímco systém Windows používá prostředí .NET CoreCLR. Dále následuje proces kompilace aplikace pro konkrétní platformu, který se může v jednotlivých OS lišit. [15]

Například aplikace pro Android se kompilují z jazyka C# do mezipřekladového jazyka (CIL), který se pak při spuštění aplikace zkompiluje do vlastní sestavy. Aplikace pro systém iOS jsou kompletně kompilovány z jazyka C# do vlastního kódu ARM. Aplikace pro MacOS používají Mac Catalyst k převodu aplikací pro iOS na desktop a aplikace pro Windows používají knihovnu Windows UI 3 (WinUI 3) k sestavení vlastních aplikací pro desktop systému Windows.

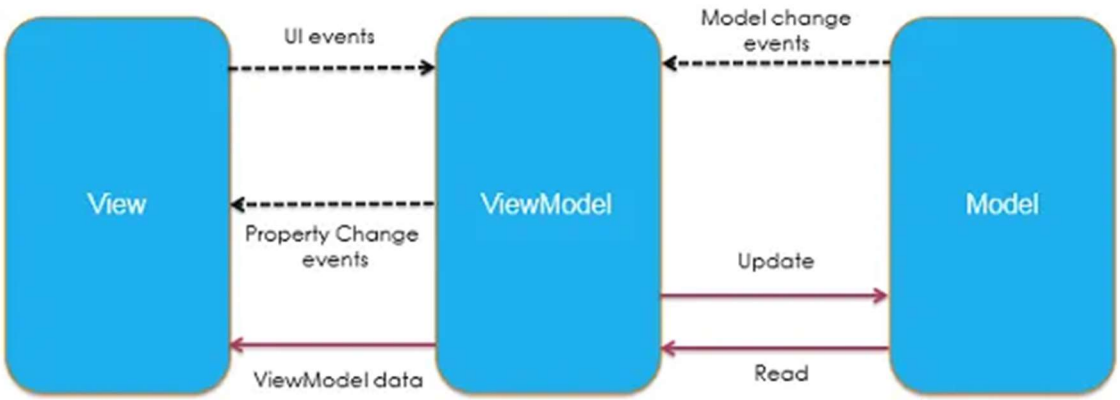
MAUI poskytuje vysokou míru přenositelnosti kódu, což umožňuje vývojářům vytvářet aplikace, které lze bezproblémově spouštět v různých operačních systémech. Díky možnosti vytvářet jediný kód pro více platforem se mohou vývojové týmy zaměřit na zlepšování funkčnosti a uživatelského prostředí, aniž by ztrácely čas přizpůsobováním se různým prostředím. Kromě toho MAUI poskytuje přístup k bohatému ekosystému nástrojů a knihoven, což dále zjednodušuje proces vývoje a zvyšuje efektivitu.

1.7 Architektura Aplikace

Architektura aplikace určuje, jakým způsobem komponenty komunikují, zpracovávají data a vytvářejí uživatelské prostředí. Moderní aplikace potřebují flexibilní a modulární architekturu, například mikroslužby, aby byla zajištěna škálovatelnost a výkonnost.

1.7.1 Architektura MVVM

Architektura MVVM je efektivní metoda organizace kódu v aplikacích uživatelského rozhraní. Rozděluje aplikaci na tři hlavní komponenty: Model, Zobrazení (View) a ViewModel. Na obrázku 3 je zobrazeno architekturu MVVM.



Obrázek 3: Architektura MVVM [16].

Model je zodpovědný za správu dat a obchodní logiky. Je to nezávislá část aplikace, která poskytuje data, jež mají být zobrazena v zobrazení. [17]

Zobrazení je zodpovědné za zobrazení dat a interakci s uživatelem. V nejlepším případě by neměl obsahovat žádnou obchodní logiku. [17]

ViewModel propojuje model a zobrazení. Poskytuje data a příkazy, které mají být zobrazeny v pohledu, a zpracovává uživatelské úkony tak, že je předává modelu. ViewModel může obsahovat logiku související se zobrazováním dat. [17]

V aplikacích využívajících MVVM je View často vytvořeno pomocí značkovacího jazyka XAML a ViewModel je napsán v programovacích jazycích, jako je C# nebo Java. Tím je zajištěno čisté oddělení povinností, kdy View odpovídá pouze za zobrazování a ViewModel za správu stavu aplikace. [17]

Oddělení kódu poskytuje řadu výhod: umožňuje iterativní programování, zjednodušuje testování jednotek, podporuje týmovou spolupráci a zlepšuje udržovatelnost. Použití vzoru MVVM kromě toho pomáhá dlouhodobě zvyšovat výkon a rozšiřitelnost aplikace. [18]

1.8 Knihovny

V této kapitole detailně prozkoumáme knihovny, které byly použity v bakalářské práci, a podrobněji se zaměříme na jejich účel a funkcionality, které přinášejí.

1.8.1 Knihovna SQLite-net-pcl

SQLite-net-pcl je jednoduchá a vhodná knihovna pro práci s databázemi SQLite v aplikacích na platformách .NET a .NET Core. Poskytuje jednoduché a intuitivní rozhraní API pro provádění databázových operací, jako je vytváření tabulek, spouštění dotazů a aktualizace dat.

Vzhledem ke své multiplatformní povaze lze SQLite-net-pcl používat v mobilních i desktopových aplikacích a poskytuje vysoký výkon a spolehlivost. Knihovna také podporuje mechanismy objektových vazeb (ORM), což zjednodušuje interakci s daty a snižuje potřebu ručního psaní dotazů SQL.

1.8.2 Knihovna SQLitePCLRaw.bundle_green

SQLitePCLRaw.bundle_green je rozšiřující knihovna, která rozšiřuje možnosti SQLite-net-pcl o ovladače SQLite pro různé platformy. Obsahuje sadu ovladačů, které lze použít na platformách, jako jsou Android, iOS, macOS a Windows, což z ní činí ideální volbu pro vývoj různé platformy. Díky této knihovně mohou vývojáři zajistit, aby jejich aplikace byly vzájemně kombinovatelné mezi různými operačními systémy a zařízeními, a přitom zůstaly uživatelsky přívětivé a snadno použitelné.

1.8.3 Knihovna ClosedXML

ClosedXML je knihovna pro práci s tabulkami ve formátu Excel, určená pro platformu .NET. Nabízí vývojářům nástroje pro snadné vytváření, úpravy a čtení souborů. Knihovna ClosedXML se vyznačuje tím, že podporuje mnoho základních i pokročilých funkcí Excelu, jako jsou formátování buněk, práce s tabulkami, grafy, kontingenčními tabulkami a dalšími. Uživatelé mohou snadno manipulovat s daty, nastavovat formátování a provádět různé operace, aniž by museli hluboce znát technické detaily formátu Excelu. Knihovna ClosedXML je aktivně udržována a průběžně se rozšiřuje o nové funkce a vylepšení výkonu. Komunita kolem této knihovny také neustále roste, což znamená, že existuje dostatek zdrojů, dokumentace a podpory pro vývojáře, kteří s ClosedXML začínají nebo chtějí řešit specifické problémy.

1.8.3 Knihovna CommunityToolkit.MVVM

Knihovna CommunityToolkit.MVVM je jedním z modulů sady knihoven CommunityToolkit, který se zaměřuje na poskytování podpory pro architekturu MVVM v aplikacích .NET. Architektura MVVM je oblíbeným vzorem pro vývoj uživatelských rozhraní, protože odděluje logiku uživatelského rozhraní od obchodní logiky, což usnadňuje testování, údržbu a rozšiřování aplikací.

CommunityToolkit.MVVM obsahuje mnoho užitečných nástrojů a funkcí, které usnadňují implementaci MVVM v aplikacích, zejména v rámci platformy MAUI, Xamarin.Forms a WinUI. Některé z klíčových prvků zahrnují:

- **Pozorovatelné objekty:** CommunityToolkit.MVVM poskytuje snadný způsob, jak implementovat rozhraní *INotifyPropertyChanged*, což je základ pro vytváření reaktivních uživatelských rozhraní, která se aktualizují při změně dat.
- **Příkazy:** Modul obsahuje nástroje pro implementaci příkazů, což je základní součást vzoru MVVM. Umožňuje oddělit logiku akce od samotného uživatelského rozhraní a přidává podporu pro asynchronní příkazy.
- **Zprostředkovatelé zpráv:** Tato funkce umožňuje komunikaci mezi různými částmi aplikace bez nutnosti přímých vazeb, což usnadňuje udržování a rozšiřování kódu.
- **Atributy a generování kódu:** CommunityToolkit.MVVM nabízí atributy, které mohou být použity k automatizaci některých aspektů MVVM, jako je generování vlastností s oznámením o změně nebo příkazů, což šetří čas a minimalizuje ruční kódování.

CommunityToolkit.MVVM je aktivně vyvíjen a udržován komunitou, což znamená, že se můžete spolehnout na pravidelné aktualizace a podporu pro nové technologie a platformy v ekosystému .NET.

1.8.3 Knihovna DependencyInjection

DependencyInjection (DI), nebo také "injektáž závislostí", je softwarový vzor a klíčová část architektury moderních aplikací. V jednoduchosti se jedná o způsob, jakým se aplikace organizuje tak, že odděluje závislosti tříd, aby byly snadno spravovatelné, testovatelné a rozšiřitelné. Namísto toho, aby třída sama vytvářela nebo vyhledávala své závislosti, jsou jí tyto závislosti předány z vnějšku, což je podstata injektáže závislostí.

Hlavní výhody Dependency Injection zahrnují:

- **Volná vazba:** Třídy nejsou pevně propojeny s konkrétními implementacemi, což umožňuje snadné nahrazení závislostí při testování nebo při změně architektury.
- **Snadné testování:** Díky tomu, že závislosti mohou být injektovány, lze je snadno nahrazovat mock objekty během testování, což usnadňuje vytváření jednotkových testů.
- **Modularita a rozšiřitelnost:** Aplikace postavené s využitím Dependency Injection jsou obecně modulární, což usnadňuje přidávání nových funkcí a přizpůsobování existujících částí.

V prostředí .NET je Dependency Injection často zajištěna pomocí kontejnerů pro správu závislostí. Tyto kontejnery umožňují definovat, jaké závislosti by měly být injektovány do konkrétních tříd, a také zajišťují jejich životní cyklus. V rámci .NET Core a novějších verzí .NET je Dependency Injection zabudována jako základní součást frameworku, s podporou pro různé scénáře, jako jsou singletony, transienci a scoped závislosti.

Implementace Dependency Injection v .NET obvykle zahrnuje několik kroků:

- **Registrace závislostí:** V kontejneru pro správu závislostí se definují třídy a jejich vztahy, včetně toho, zda se jedná o singletony, transienci nebo scoped instance.
- **Injektáž závislostí:** Závislosti jsou injektovány do tříd buď prostřednictvím konstruktorů, metod nebo vlastností. Konstruktorová injektáž je obecně preferována, protože je nejvíce explicitní.
- **Správa životního cyklu:** Kontejner pro správu závislostí zajišťuje správné vytváření a uvolňování závislostí podle jejich typu (singleton, transient, scoped).

1.9 Smluvní podmínky a zpráva bodů

Stěžejním cílem této práce je vytvořit aplikaci pro sledování realizovaných zdravotnických úkonů v domově důchodců. Tyto úkony zřizovateli proplácí zdravotní pojišťovny dle vyhlášky Č. 482/2021 [2]. na základě uzavřené smlouvy. Obvykle tato smlouva o zajištění úhrady zdravotní péče v domovech důchodců, viz [19], obsahuje následující podmínky:

Smluvní podmínky z návrhu dodatku ke Zvláštní smlouvě o poskytování a úhradě ošetřovatelské péče v zařízeních sociálních služeb poskytujících pobytové sociální služby u VZP:

1. Smluvní strany se dohodly, že výše úhrady se stanoví podle seznamu zdravotních výkonů za poskytnuté zdravotní výkony s hodnotou bodu ve výši 1,28 Kč.
2. V případě, že podíl počtu ošetřených a Pojišťovnou uznaných unikátních pojištěnců v hodnoceném období s některou z diagnóz C00 až C97, E10.3 až E10.7, E11.3 až E11.7, F00 až F99, G09 až G99 nebo I60 – I69 podle mezinárodní klasifikace nemocí na celkovém počtu ošetřených a Pojišťovnou uznaných unikátních pojištěnců v hodnoceném období překročí 25 %, navyšuje se hodnota bodu o 0,02 Kč.
3. Celková výše úhrady za výkony Pobytovému zařízení (dále jen „celková výše úhrady“) nepřekročí částku, která se vypočte takto:

$$\max\{\mathrm{PMUP_{ref}} * (\textstyle\sum_{j=1\ldots m} \mathrm{PUM_{ho,j}}) * 1{,}20 * \mathrm{KN};\ \mathrm{PB_{ho}} * \mathrm{HB_{min}} + \mathrm{KP_{ho}}\},$$

kde:

$\mathrm{PUM_{ho,j}}$ je počet vykázaných kalendářních měsíců, v nichž byly poskytovány unikátnímu pojištěnci zdravotní služby v hodnoceném období.

KN je koeficient navýšení, který nabývá hodnoty 1,02 v případě, že podíl počtu ošetřených a Pojišťovnou uznaných unikátních pojištěnců v hodnoceném období s některou z diagnóz C00 až C97, E10.3 až E10.7, E11.3 až E11.7, F00 až F99, G09 až G99 nebo I60 – I69 podle mezinárodní klasifikace nemocí na celkovém počtu ošetřených a Pojišťovnou uznaných unikátních pojištěnců v hodnoceném období překročí 25 %, a hodnoty 1,00 v ostatních případech.

$\mathrm{PB_{ho}}$ je celkový počet Pobytovým zařízením vykázaných a Pojišťovnou uznaných bodů v hodnoceném období.

$\mathrm{HB_{min}}$ je minimální hodnota bodu, která se stanoví ve výši 1,03 Kč.

KP_{ho} je hodnota korunových položek v hodnoceném období.

max funkce maximum, která vybere z oboru hodnot hodnotu nejvyšší.

j nabývá hodnot 1 až m, kde m je počet unikátních pojištěnců ošetřených v hodnoceném období.

$\mathrm{PMUP_{ref}}$ je průměrná měsíční úhrada za unikátního pojištěnce v referenčním období vypočtená jako:

$$\mathrm{PMUP_{ref}} = \frac{\mathrm{Uhr_{ref}}}{\sum_{\mathrm{i=1...n}} \mathrm{PUM_{ref,i}}},$$

kde:

$\mathrm{Uhr_{ref}}$ je celková úhrada Pobytového zařízení za výkony, včetně zvlášť účtovaného materiálu a zvlášť účtovaných léčivých přípravků v referenčním období.

$\mathrm{PUM_{ref,i}}$ je počet vykázaných kalendářních měsíců, v nichž byly poskytovány unikátnímu pojištěnci i zdravotní služby v referenčním období.

i nabývá hodnot 1 až n, kde n je počet unikátních pojištěnců ošetřených v referenčním období.

4. Pokud je Pobytovému zařízení oproti referenčnímu období nově nasmlouván výkon č. 06648 (bonifikační výkon za práci v nepřetržitém nebo v třísměnném pracovním

režimu), přičemž Pobytové zařízení v referenčním období vykazovalo hrazené služby poskytnuté v době od 22:00 do 06:00 hodin výkonem č. 06645 (bonifikační výkon za práci v době od 22:00 do 06:00 hodin) a hrazené služby poskytnuté v době pracovního volna nebo pracovního klidu vykazovalo výkonem č. 06649 (bonifikační výkon za práci v době pracovního volna nebo pracovního klidu), hodnota $\mathrm{PMUP_{ref}}$ se navýší o částku vypočtenou takto:

$$\left\{\frac{\mathrm{PV_{výk2021}} * (\mathrm{BH_{výk2023}} - \mathrm{BH_{výk2021}})}{\mathrm{PUM_{ref,i}}}\right\} * \mathrm{HB_{ref}},$$

kde:

$\mathrm{PV_{výk2021}}$ celkový počet Pobytovým zařízením vykázaných a Pojišťovnou uznaných výkonů č. 06645 – bonifikační výkon za práci v době od 22:00 do 06:00 hodin a výkonů č. 06649 – bonifikační výkon za práci v době pracovního volna nebo pracovního klidu. Do celkového počtu výkonů jsou zařazeny výkony uvedené ve větě první poskytnuté v roce 2021, Pobytovým zařízením vykázané do 31. 3. 2022 a Pojišťovnou uznané do 31. 5. 2022.

$\mathrm{BH_{výk2021}}$ bodová hodnota výkonu č. 06645 – bonifikační výkon za práci v době od 22:00 do 06:00 hodin respektive výkonu č. 06649 – bonifikační výkon za práci v době pracovního volna nebo pracovního klidu, uvedená v seznamu zdravotních výkonů platném pro rok 2021.

$\mathrm{BH_{výk2023}}$ bodová hodnota výkonu č. 06645 - bonifikační výkon za práci v době od 22:00 do 06:00 hodin respektive výkonu č. 06649 – bonifikační výkon za práci v době pracovního volna nebo pracovního klidu uvedená v seznamu zdravotních výkonů platném pro rok 2021, přičemž bodová hodnota výkonu č. 06645 a výkonu č. 06649 je navýšena o bodovou hodnotu výkonu č. 06648 - bonifikační výkon za práci v nepřetržitém nebo v třísměnném pracovním režimu, uvedeném v seznamu zdravotních výkonů platném pro rok 2023.

$\mathrm{HB_{ref}}$ hodnota bodu v referenčním období, která je stanovena ve výši 1,16 Kč.

2 PRAKTICKÁ ČÁST

Tato kapitola bakalářské práce se zaměřuje na implementaci a návrh aplikace. Obsahuje popis databáze, která obsahuje klíčová data potřebná pro fungování aplikace, včetně informací o zaměstnancích, výkonech, zákaznících a diagnózách. Tato data jsou uložena v jednoduché databázi SQLite, která byla speciálně navržena pro tuto aplikaci.

Dále kapitola popisuje také souborovou strukturu aplikace. Tato struktura zahrnuje různé soubory a adresáře, které společně tvoří kompletní aplikaci. Soubory zahrnují kód pro jednotlivé funkce aplikace, jako je správa databáze, logika uživatelského rozhraní, a další komponenty, které jsou klíčové pro fungování aplikace. Adresáře jsou organizovány tak, aby poskytovaly jasnou strukturu a usnadňovaly navigaci mezi různými částmi kódu.

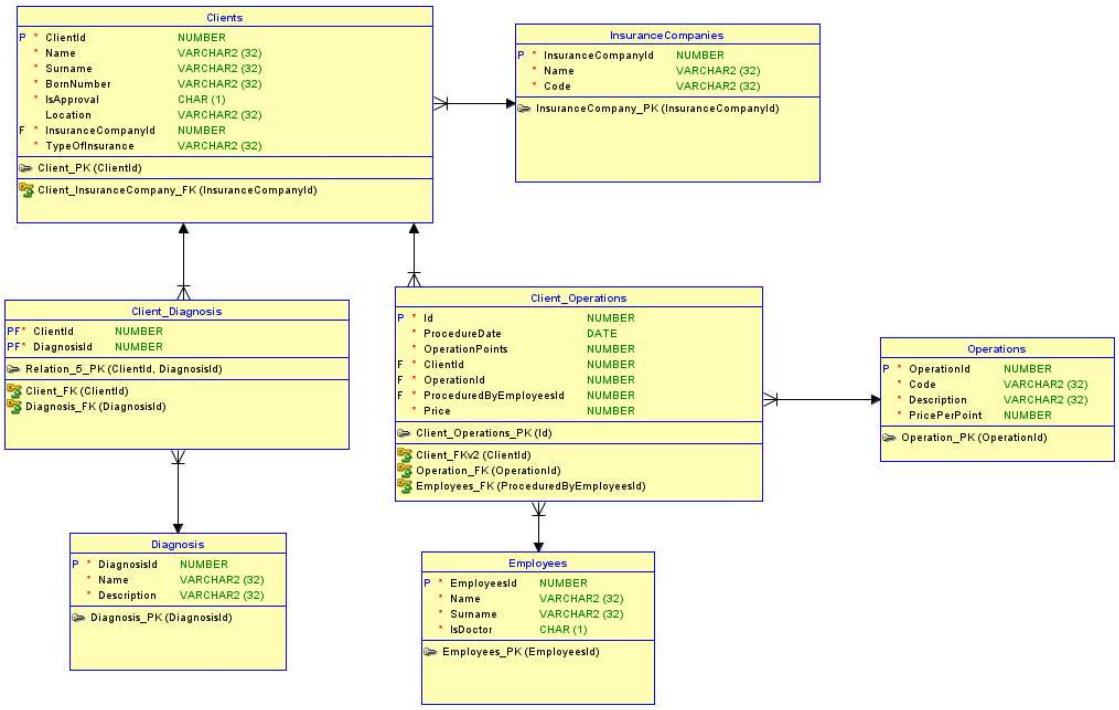
Kapitola se také zabývá uživatelským rozhraním, které je navrženo tak, aby bylo snadno použitelné a poskytovalo všechny potřebné funkce pro správu dat uložených v databázi. Návrh uživatelského rozhraní byl vytvořen pomocí jazyka XAML, který umožňuje vytvořit bohaté a intuitivní grafické rozhraní. Díky tomu je aplikace uživatelsky přívětivá a poskytuje nástroje pro efektivní práci se zákazníky a dalšími daty.

Celkově tato kapitola poskytuje ucelený pohled na implementaci aplikace, včetně designu databáze, souborové struktury a uživatelského rozhraní.

2.1 Struktura databáze aplikace

Základem každé aplikace je její databáze, a proto se v této části bakalářské práce zaměříme na její strukturu a design. Obrázek 4 detailně popisuje, jaké tabulky jsou v databázi přítomny a jaká pole obsahují. Navíc jasně ukazuje, jak jsou jednotlivé entity v databázi propojeny a jaké jsou mezi nimi vztahy. Toto schéma poskytuje ucelený přehled o tom, jak je databáze navržena a jaké informace v ní mohou být uloženy.

Databáze pro tuto aplikaci je navržena tak, aby spravovala různé aspekty související se zdravotní péčí a souvisejícími procesy. Zahrnuje několik tabulek, z nichž každá má specifickou roli a klíčové vztahy s ostatními částmi systému. Tyto tabulky společně tvoří základ databáze a zajišťují efektivní správu informací o klientech, diagnózách, prováděných operacích, pojišťovnách a zaměstnancích.



Obrázek 4: Struktura databáze.

2.1.1 Tabulka Client

Tabulka *Client* obsahuje základní informace o klientech. Každý záznam má unikátní identifikátor *clientId*, což je primární klíč, a další informace, jako je jméno (*name*), příjmení (*surname*), a rodné číslo (*bornNumber*). Dále obsahuje pole *isApproval*, které indikuje, zda musíme klienta vykazovat, a jeho hodnota musí být 0 nebo 1, což je vynuceno omezením *CHECK*. Tabulka také obsahuje údaje o umístění (*location*), typ pojištění (*typeOfInsurance*) a identifikátor pojišťovny (*insuranceCompanyId*). Tento poslední sloupec má cizí klíč, který odkazuje na tabulku *InsuranceCompany*, což znamená, že každý klient je spojen s konkrétní pojišťovnou.

2.1.2 Tabulka ClientsDiagnosis

Tabulka *ClientsDiagnosis* zachycuje vztah mezi klienty a jejich diagnózami. Primární klíč tvoří kombinace *clientId* a *diagnosisId*, což znamená, že jeden klient může mít více diagnóz a jedna diagnóza může být přidělena více klientům. Tato tabulka má cizí klíče odkazující na tabulky *Client* a *Diagnosis*, čímž se vytváří propojení mezi těmito entitami.

2.1.3 Tabulka ClientsOperations

Tabulka *ClientsOperations* sleduje operace prováděné na klientech. Každý záznam má unikátní identifikátor *id* a obsahuje data, jako je datum výkonu (*procedureDate*), počet bodů za výkon (*operationPoints*), a cena (*price*). Dále jsou zde cizí klíče, které odkazují na klienta,

typ operace a zaměstnance, který výkon provedl. To zajišťuje propojení mezi klienty, operacemi a zaměstnanci.

2.1.4 Tabulka Diagnosis

Tabulka *Diagnosis* uchovává informace o diagnózách. Každý záznam má unikátní identifikátor *diagnosisId*, což je primární klíč. Obsahuje také název diagnózy (*name*) a její popis (*description*). Tato tabulka slouží jako základní rejstřík diagnóz, které mohou být přiděleny klientům.

2.1.5 Tabulka Employee

Tabulka *Employee* obsahuje údaje o zaměstnancích, včetně jejich jména (*name*), příjmení (*surname*), a indikátoru, zda jsou lékaři (*isDoctor*). Tento indikátor je omezen hodnotami 0 nebo 1. Každý záznam má unikátní identifikátor *employeesId*, což je primární klíč.

2.1.6 Tabulka InsuranceCompany

Tabulka *InsuranceCompany* obsahuje informace o pojišťovnách. Každý záznam má unikátní identifikátor *insuranceCompanyId*, spolu s názvem (*name*) a kódem pojišťovny (*code*). Tato tabulka propojuje klienty s jejich pojišťovnami.

2.1.7 Tabulka Operation

Tabulka *Operation* zahrnuje informace o typech výkonů. Každý záznam má unikátní identifikátor *operationId*, spolu s kódem výkonu (*code*), popisem (*description*), a cenou za bod (*pricePerPoint*). Tato tabulka je základem pro sledování výkonů prováděných na klientech a je propojena s tabulkou *ClientsOperations*.

Každá tabulka má svou vlastní strukturu a propojení s ostatními, což umožňuje vytvářet komplexní vztahy mezi různými prvky systému. Výsledkem je databáze, která podporuje klíčové funkce aplikace a zajišťuje integritu a konzistenci uložených dat.

2.2 Souborová struktura aplikace

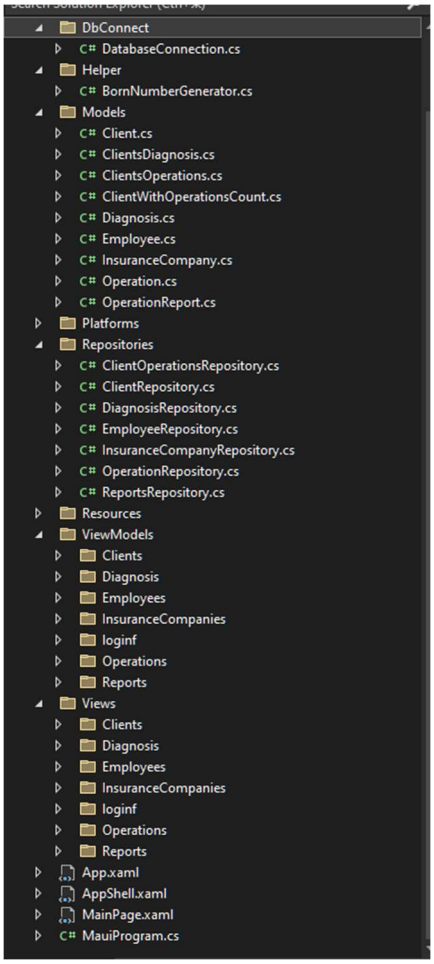
Souborová struktura aplikace je rozdělena do několika složek: DbContext, Helper, Platforms, Repositories, Resources, ViewModels a Views. Každá z těchto složek má specifickou funkci a obsahuje soubory, které se týkají dané části aplikace. Kromě těchto složek obsahuje kořen projektu několik klíčových souborů, jako jsou App.xaml, AppShell.xaml, MainPage.xaml a MauiProgram.cs.

Na obrázku 5 můžeme vidět celkový přehled souborové struktury této aplikace. Z tohoto obrázku je jasně patrné rozdělení projektu do jednotlivých složek a umístění hlavních

souborů. Složky DbContext, Helper, Platforms, Repositories, Resources, ViewModels a Views jsou uspořádány tak, aby poskytovaly logickou organizaci kódu a usnadňovaly práci vývojářům.

Kořen projektu, který obsahuje soubory jako App.xaml, AppShell.xaml, MainPage.xaml a MauiProgram.cs, slouží jako základní bod pro konfiguraci a nastavení aplikace. Tyto soubory jsou klíčové pro inicializaci a fungování aplikace, a proto jsou umístěny v kořenové složce, což je vidět na obrázku 5.

Celkově tato souborová struktura umožňuje přehledné a organizované uspořádání kódu, což je zásadní pro efektivní vývoj a údržbu aplikace. Obrázek 5 poskytuje vizuální reprezentaci této struktury, což usnadňuje pochopení různých částí projektu a jejich propojení.



Obrázek 5: Souborová struktura aplikace.

2.2.1 Složka DbContext

Složka *DbContext* obsahuje jeden důležitý soubor s názvem *DatabaseConnection.cs* (obr. 6), který je zodpovědný za vytvoření připojení k databázi *Clienttracker.sqlite*. Tato třída

poskytuje základní rozhraní pro připojení k databázi a zajišťuje, že aplikace může pracovat s daty uloženými v SQLite.

```
9 references
public class DatabaseConnection
{
    private readonly SQLiteAsyncConnection _connection;

    1 reference
    public DatabaseConnection()
    {
        var dbPath = Path.Combine(FileSystem.AppDataDirectory, "Clienttracker.sqlite");
        _connection = new SQLiteAsyncConnection(dbPath);
    }

    7 references
    public SQLiteAsyncConnection GetConnection()
    {
        return _connection;
    }
}
```

Obrázek 6: Ukázka kódu třídy DatabaseConnection.

Třída *DatabaseConnection* používá knihovnu SQLite a vytváří asynchronní připojení k databázi. Při inicializaci se databázový soubor umístí do adresáře pro aplikační data (*AppDataDirectory*), což je typický postup pro mobilní a desktopové aplikace. Metoda *GetConnection* vrací vytvořené připojení, které lze použít pro provádění databázových operací, jako je čtení, zápis nebo aktualizace dat.

2.2.2 Složka Helper

Složka *Helper* obsahuje jeden pomocný soubor s názvem *BornNumberGenerator.cs*, který se používá při importu testovacích dat. Tento soubor poskytuje nástroje pro generování rodných čísel, které se používají při simulaci a testování aplikace.

```
public static class BornNumberGenerator
 {
    private static Random random = new Random();

    public static string Generate(int orderNumber)
    {
        int age = random.Next(60, 81);
        int currentYear = DateTime.Now.Year;
        int birthYear = (currentYear - age) % 100;
        int randomMonth = random.Next(1, 13);
        int randomDay = random.Next(1, 30);

        string birthDatePart = $"{birthYear:00}{randomMonth:00}{randomDay:00}";

        string uniquePart = orderNumber.ToString("000");
```

```
        long baseNumber = long.Parse(birthDatePart + uniquePart);

        int checkDigit = CalculateCheckDigit(baseNumber);

        return $"{birthDatePart}/{uniquePart}{checkDigit}";
    }

    private static int CalculateCheckDigit(long baseNumber)
    {
        long remainder = baseNumber % 11;
        if (remainder == 10)
        {
            return 0;
        }
        return (int)remainder;
    }

}
```

Třída *BornNumberGenerator*, obsahuje metodu *Generate*, která vygeneruje rodné číslo na základě vstupního pořadového čísla (*orderNumber*). Tato metoda nejprve vypočítá náhodný věk mezi 60 a 81 lety a odhadne rok narození z aktuálního roku. Poté vygeneruje náhodný měsíc a den. Tímto způsobem vytvoří řetězec představující část data narození.

Unikátní část rodného čísla je generována z pořadového čísla a přidává se k datumu narození, aby vytvořila základní číslo. Nakonec se vypočítá kontrolní číslice pomocí metody *CalculateCheckDigit*. Pokud je zbytek dělení základního čísla 11 desítka, vrátí se kontrolní číslice 0. Jinak se použije zbytek jako kontrolní číslice.

Výsledkem této metody je vygenerované rodné číslo ve formátu YYMMDD/XXXK, kde YYMMDD představuje zjednodušené datum narození, XXX je unikátní část a K je kontrolní číslice.

2.2.3 Složka Platforms

Složka *Platforms* obsahuje soubory a konfigurace specifické pro jednotlivé platformy, na kterých aplikace je spuštěna. V projektu MAUI jsou podporovány různé platformy, jako jsou iOS, Android, Windows a další. Každá z těchto platforem má své specifické požadavky a nastavení, a proto tato složka zahrnuje příslušné soubory a konfigurace pro každou z těchto platforem.

2.2.4 Složka Resources

Složka *Resources* je určena pro všechny statické zdroje, které aplikace používá. To zahrnuje různé typy zdrojů, jako jsou obrázky, ikony, soubory se styly, písma a další. V projektu MAUI

je tato složka klíčová pro zajištění konzistentního vzhledu a chování aplikace napříč různými platformami.

2.2.5 Složka Repositories

Složka *Repositories* obsahuje soubory, které zajišťují přístup k datům uloženým v databázi a jejich správu. Repositories slouží jako vrstva mezi databázovým připojením a aplikační logikou, což umožňuje oddělit způsob přístupu k datům od ostatních částí aplikace. Tímto způsobem je dosaženo vyšší flexibility a snazší údržby kódu, protože změny v databázi nebo přístupových metodách neovlivní přímo ostatní části aplikace.

V této složce najdeme soubory, které poskytují metody pro práci s různými datovými entitami. Typicky obsahují CRUD operace (Create, Read, Update, Delete), které umožňují vytváření nových záznamů, čtení existujících, jejich aktualizaci a mazání. Soubory ve složce Repositories mohou také zahrnovat další metody pro složitější dotazy nebo jinou logiku týkající se práce s daty.

Celkově složka Repositories zajišťuje, že přístup k datům je dobře organizovaný a izolovaný od zbytku aplikace. To přispívá k lepší struktuře aplikace a usnadňuje budoucí rozšiřování a údržbu.

2.2.6 Složka ViewModels

Složka *ViewModels* hraje klíčovou roli ve struktuře aplikací, které používají architekturu MVVM. Tato složka obsahuje soubory, které fungují jako most mezi datovou vrstvou a uživatelským rozhraním. Ve složce *ViewModels* jsou umístěny třídy, které představují logiku pro jednotlivé části uživatelského rozhraní a zajišťují komunikaci s daty.

Třídy ve složce *ViewModels* mají několik klíčových funkcí:

- **Správa dat:** Třídy ve složce ViewModels přebírají data z modelů nebo repositářů a připravují je pro zobrazení v uživatelském rozhraní. Tím se odděluje datová logika od samotného zobrazení.
- **Reaktivita**: Umožňuje uživatelskému rozhraní automaticky reagovat na změny dat ve *ViewModelu*. Tím se dosahuje interaktivního a responzivního uživatelského zážitku.
- **Zpracování akcí:** *ViewModely* obsahují příkazy (*Relay Commands*), které jsou připojeny k akcím v uživatelském rozhraní, jako je kliknutí na tlačítko nebo checkbox. To umožňuje oddělit logiku uživatelského rozhraní od samotného zpracování akcí.

2.2.7 Složka Views

Složka *Views* je základní součástí architektury aplikace, která definuje vizuální prezentaci uživatelského rozhraní. Obsahuje soubory ve formátu XAML, které popisují rozložení a vzhled jednotlivých obrazovek nebo prvků UI. Tato složka slouží k oddělení prezentace od logiky aplikace, což usnadňuje úpravy vzhledu bez zásahu do datové logiky. Významně spolupracuje se složkou *ViewModels*, která řídí logiku a interakci s daty.

2.2.8 Soubor App.xaml

Soubor *App.xaml* je jedním z klíčových souborů v projektech založených na .NET MAUI. Obsahuje základní konfigurace a nastavení, které mají vliv na celou aplikaci. Mezi tyto konfigurace patří definování globálních zdrojů, což může zahrnovat věci jako jsou barvy, písma nebo další grafické prvky, které se používají napříč aplikací. Dále jsou v něm uloženy styly a šablony, které určují, jak budou vypadat jednotlivé komponenty uživatelského rozhraní, jako jsou tlačítka, textová pole, nebo seznamy.

App.xaml pomáhá zajistit konzistentní vzhled a chování aplikace, protože umožňuje sdílet nastavení napříč různými částmi projektu. Díky tomu, že se zde definují globální zdroje a styly, je možné snadno udržovat jednotný design, aniž by museli upravovat každou stránku nebo komponentu samostatně. Celkově vzato, soubor App.xaml představuje centralizovaný bod pro konfiguraci aplikace a hraje zásadní roli při vytváření a správě globálních nastavení.

2.2.9 Soubor AppShell.xaml

Soubor AppShell.xaml obsahuje konfigurace pro hlavní strukturu a navigaci aplikace v prostředí .NET MAUI. Tento soubor využívá jazyk XAML k definování výchozí stránky uživatelského rozhraní aplikace a celkového chování navigace.

```
<?xml version="1.0" encoding="utf-8"?>
<Shell
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:views="clr-namespace:ClientCareTracker.Views.Employees"
    xmlns:views1="clr-namespace:ClientCareTracker.Views.Operations"
    xmlns:views2="clr-namespace:ClientCareTracker.Views.Diagnosis"
    xmlns:views3="clr-namespace:ClientCareTracker.Views.Clients"
    xmlns:views4="clr-namespace:ClientCareTracker.Views.InsuranceCompanies"
    xmlns:views5="clr-namespace:ClientCareTracker.Views.Reports"

    x:Class="ClientCareTracker.MainPage">

    <TabBar>
        <ShellContent
```

```
            Title="Klienti"
            ContentTemplate="{DataTemplate views3:ClientDetailPage}"/>

        <ShellContent
            Title="Zaměstnanci"
            ContentTemplate="{DataTemplate views:EmployeeDetailPage}"/>

        <ShellContent
            Title="Výkony"
            ContentTemplate="{DataTemplate views1:OperationDetailPage}"/>

        <ShellContent
            Title="Diagnózy"
            ContentTemplate="{DataTemplate views2:DiagnosisDetailPage}"/>

        <ShellContent
            Title="Pojišťovny"
            ContentTemplate="{DataTemplate views4:InsuranceCompanyDetailPage}"/>


        <ShellContent
            Title="Zprávy"
            ContentTemplate="{DataTemplate views5:InsuraceReportPage}"/>
    </TabBar>

</Shell>
```

Kód začíná definicí hlavní komponenty *<Shell>*, která je základním stavebním kamenem pro definování navigační struktury v .NET MAUI. Součástí této definice jsou různé XML jmenné prostory (xmlns), které umožňují přístup k různým třídám a funkcím, jako je základní prostor pro MAUI a vlastní jmenný prostor (*views)*, který odkazuje na *ClientCareTracker.Views.loginf*.

Atribut *Shell.FlyoutBehavior="Disabled"* naznačuje, že aplikace nepoužívá tradiční boční menu.

Nadpis *Title="Sledování péče o klienty"* je zobrazen jako hlavní název aplikace nebo aktuální sekce. Tento název je zobrazen uživatelům v horní části uživatelského rozhraní, což poskytuje kontext a orientaci.

V rámci komponenty *<ShellContent>* je nastaven atribut *ContentTemplate="{DataTemplate views:WelcomePage}",* což znamená, že při spuštění aplikace se zobrazí stránka *WelcomePage*. Tato část kódu je zodpovědná za určení výchozí stránky aplikace a jejího obsahu.

2.2.10 Soubor MainPage.xaml

Soubor *MainPage.xaml* definuje strukturu stránky, která se zobrazuje po úspěšném přihlášeni do aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<Shell
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:views="clr-namespace:ClientCareTracker.Views.Employees"
    xmlns:views1="clr-namespace:ClientCareTracker.Views.Operations"
    xmlns:views2="clr-namespace:ClientCareTracker.Views.Diagnosis"
    xmlns:views3="clr-namespace:ClientCareTracker.Views.Clients"
    xmlns:views4="clr-namespace:ClientCareTracker.Views.InsuranceCompanies"
    xmlns:views5="clr-namespace:ClientCareTracker.Views.Reports"

    x:Class="ClientCareTracker.MainPage">

    <TabBar>
        <ShellContent
            Title="Klienti"
            ContentTemplate="{DataTemplate views3:ClientDetailPage}"/>

        <ShellContent
            Title="Zaměstnanci"
            ContentTemplate="{DataTemplate views:EmployeeDetailPage}"/>

        <ShellContent
            Title="Výkony"
            ContentTemplate="{DataTemplate views1:OperationDetailPage}"/>

        <ShellContent
            Title="Diagnózy"
            ContentTemplate="{DataTemplate views2:DiagnosisDetailPage}"/>

        <ShellContent
            Title="Pojišťovny"
            ContentTemplate="{DataTemplate views4:InsuranceCompanyDetailPage}"/>


        <ShellContent
            Title="Zprávy"
            ContentTemplate="{DataTemplate views5:InsuraceReportPage}"/>
    </TabBar>

</Shell>
```

Soubor začíná komponentou *<Shell>*, která je základem pro definování hlavní struktury aplikace. Uvnitř této komponenty je definována část *<TabBar>*, což je mechanismus, který umožňuje navigaci mezi různými sekcemi aplikace pomocí záložek. Každá záložka je

reprezentována komponentou <*ShellContent*>, která specifikuje, na jakou stránku se záložka odkazuje a jaký má název.

V této ukázce *MainPage.xaml* obsahuje několik záložek, které poskytují přístup k různým částem aplikace. Například záložka *Klienti* s atributem *ContentTemplate="{DataTemplate views3:ClientDetailPage}"* odkazuje na stránku s detailními informacemi o klientech. Podobně záložka *Zaměstnanci* vede na stránku *EmployeeDetailPage*, která se zaměřuje na detaily o zaměstnancích.

Další záložky zahrnují *Výkony* (s odkazem na *OperationDetailPage*), *Diagnózy* (*DiagnosisDetailPage*), Pojišťovny (*InsuranceCompanyDetailPage*), a *Zprávy* (*InsuraceReportPage*). Každá z těchto záložek poskytuje uživatelům přístup k jiné části aplikace, což umožňuje snadnou navigaci mezi různými funkcemi a sekcemi.

Celkově tento soubor určuje základní rozložení stránky aplikace pomocí záložek. Díky komponentě <*TabBar*> mohou uživatelé snadno přepínat mezi různými částmi aplikace, což poskytuje intuitivní a pohodlný uživatelský zážitek. Soubor *MainPage.xaml* je tedy klíčovým prvkem pro definování hlavní struktury aplikace a pro zajištění, že navigace bude uživatelsky přívětivá a efektivní.

2.2.11 Soubor MauiProgram.cs

Soubor *MauiProgram.cs* (obr. 7) je hlavním konfiguračním souborem ve projektech .NET MAUI. Metoda *CreateMauiApp* vytváří instanci aplikace pomocí *MauiApp.CreateBuilder()* a určuje základní nastavení, jako je použití aplikace *App*, přidání písem a další konfigurační prvky. V tomto souboru se provádí registrace služeb pomocí Dependency Injection (DI), což umožňuje snadné vytváření instancí a práci s různými komponentami aplikace. Například registrace *DatabaseConnection* pomocí *AddSingleton* zajišťuje, že tato instance bude v aplikaci existovat jako singleton, což je ideální pro připojení k databázi.

Soubor dále registruje různé repositáře, jako jsou *EmployeeRepository*, *OperationRepository*, *DiagnosisRepository* a další, které poskytují přístup k datům a spravují operace spojené s databází. Kromě toho jsou zde registrovány i *ViewModely*. Jsou registrovány jako "transient", což znamená, že pro každé použití se vytvoří nová instance, což je vhodné pro scénáře s krátkodobými instancemi, jako jsou ViewModely pro různé stránky.

Soubor také obsahuje registraci stránek (*Views*), jako jsou *EmployeeDetailPage*, *OperationDetailPage*, *ClientDetailPage* a další, což umožňuje aplikaci vědět, jaké stránky má

vytvořit a jak je propojit s odpovídajícími *ViewModely*. Tyto registrace umožňují aplikaci pracovat s různými prvky uživatelského rozhraní a logikou dat. Celkově soubor *MauiProgram.cs* zajišťuje, že aplikace má správně zaregistrované všechny potřebné služby, repositáře, *ViewModely* a stránky, což podporuje efektivní vývoj a udržitelnost kódu.

```
public static class MauiProgram
{
    4 references
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });
        // db
        builder.Services.AddSingleton(x => new DatabaseConnection());
        // Repository
        builder.Services.AddSingleton<EmployeeRepository>();
        builder.Services.AddSingleton<OperationRepository>();
        builder.Services.AddSingleton<DiagnosisRepository>();
        builder.Services.AddSingleton<ClientRepository>();
        builder.Services.AddSingleton<InsuranceCompanyRepository>();
        builder.Services.AddSingleton<ClientOperationsRepository>();
        builder.Services.AddSingleton<ReportsRepository>();
        // ViewModel
        builder.Services.AddTransient<EmployeesViewModel>();
        builder.Services.AddTransient<OperationViewModel>();
        builder.Services.AddTransient<DiagnosisViewModel>();
        builder.Services.AddTransient<InsuranceCompanyViewModel>();
        builder.Services.AddTransient<ClientInfoViewModel>();
        builder.Services.AddTransient<LoginViewModel>();
        builder.Services.AddTransient<ReportsViewModel>();
        builder.Services.AddTransient<ClientViewModel>();
        // View
        builder.Services.AddTransient<EmployeeDetailPage>();
        builder.Services.AddTransient<OperationDetailPage>();
        builder.Services.AddTransient<DiagnosisDetailPage>();
        builder.Services.AddTransient<ClientDetailPage>();
        builder.Services.AddTransient<InsuranceCompanyDetailPage>();
        builder.Services.AddTransient<ClientInfoPage>();
        builder.Services.AddTransient<LoginPage>();
        builder.Services.AddTransient<WelcomePage>();
        builder.Services.AddTransient<MainPage>();
        builder.Services.AddTransient<AddDataPage>();
        builder.Services.AddTransient<InsuraceReportPage>();
        builder.Services.AddTransient<CountPage>();
        return builder.Build();
    }
}
```

Obrázek 7: Ukázka kódu třídy MauiProgram.

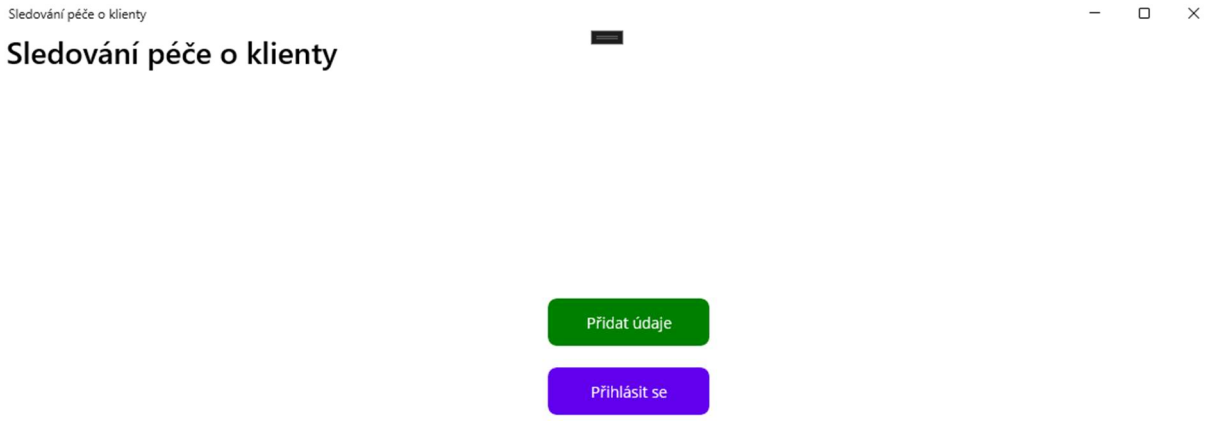
2.3 Uživatelské rozhraní aplikace

Uživatelské rozhraní aplikace je klíčovou součástí, která určuje, jak uživatelé interagují s aplikací a jak ji vnímají. Dobrý design uživatelského rozhraní zajišťuje, že aplikace je intuitivní, snadno ovladatelná a atraktivní. To znamená, že uživatelé mohou rychle pochopit, jak aplikace funguje, a mohou efektivně používat její funkce.

Důležitost uživatelského rozhraní spočívá v tom, že je to první věc, kterou uživatelé vidí, když aplikaci otevřou. Pokud je rozhraní nepřehledné nebo obtížné na ovládání, může to negativně ovlivnit uživatelský zážitek a vést k frustraci. Naopak, dobře navržené uživatelské rozhraní s jasnou strukturou a konzistentním designem může zvýšit spokojenost uživatelů a podporovat jejich dlouhodobé používání aplikace. Klíčovými prvky dobrého uživatelského rozhraní jsou jednoduchost, přehlednost a konzistence.

2.3.1 Úvodní stránka aplikace

Po spuštění aplikace uživatel uvidí úvodní stránku (obr. 8) s dvěma tlačítky: „Přidat údaje“ a „Přihlásit se“. Tlačítko „Přidat údaje“ je určeno pro zaměstnance domova důchodců, kteří potřebují zapsat provedené úkony u klientů. Naopak tlačítko „Přihlásit se“ je určeno pro správce systému, kteří se mohou po přihlášení dostat do rozhraní pro správu všech dat uložených v aplikaci. Tímto způsobem je úvodní stránka navržena tak, aby okamžitě poskytla uživateli jasné možnosti podle jeho role v systému.



Obrázek 8: Úvodní stránka aplikace.

2.3.2 Stránka přidáni informace

Na této stránce (obr. 9) může zaměstnanec domova důchodců přidat do databáze provedený výkon. Pro úspěšné zadání musí zaměstnanec zadat počet bodů, vybrat klienta, kterému byl výkon proveden, zvolit typ výkonu, vybrat sebe ze seznamu zaměstnanců a kliknout na tlačítko „Přidat“. Tím se zadané informace uloží do databáze a uživatel je přesměrován na úvodní stránku.

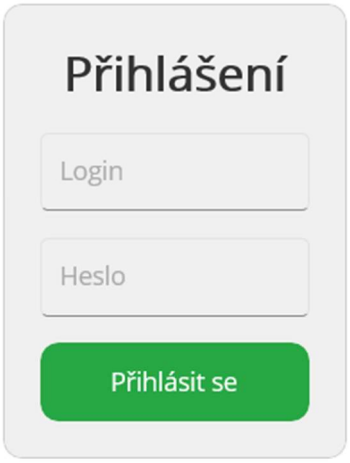
Obrázek 9: Stránka přidáni informace.

2.3.3 Stránka přihlášeni do systému

Když uživatel na úvodní stránce klikne na tlačítko „Přihlásit se“, bude přesměrován na stránku pro přihlášení jako administrátor (obr. 10). Na této stránce musí zadat přihlašovací jméno a heslo spojené s administrátorským účtem. Pro úspěšné přihlášení musí uživatel zadat správné údaje, například přihlašovací jméno "admin" a heslo "heslo".

Pokud jsou zadané údaje správné, uživatel bude přesměrován na hlavní stránku pro administrátory.

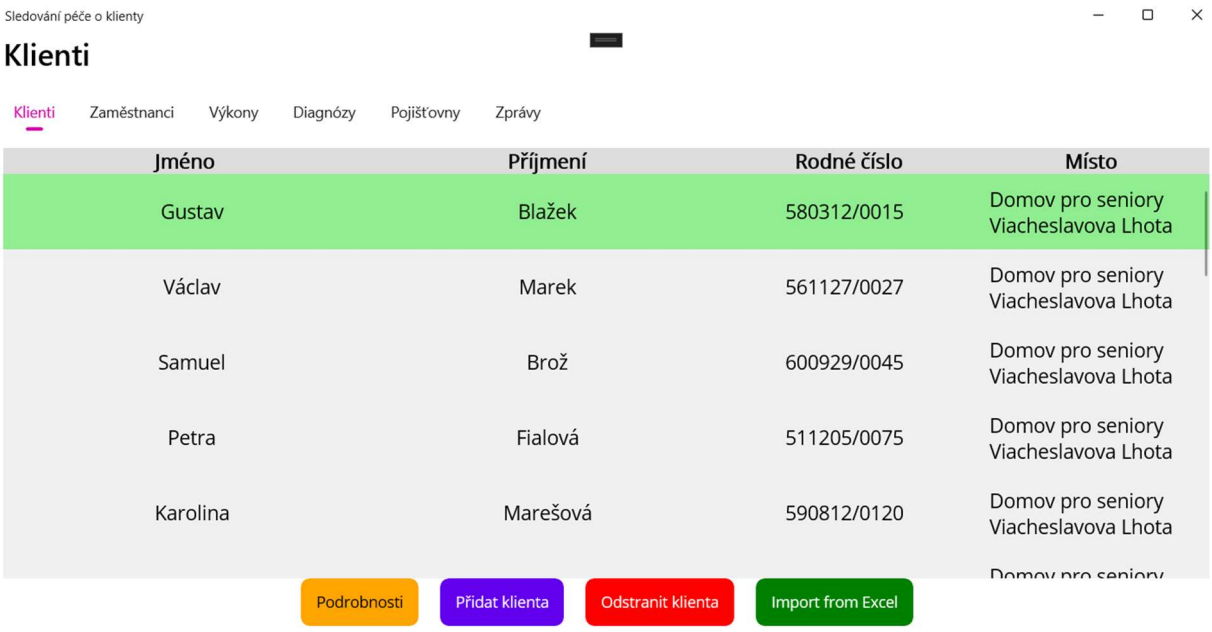
Celkově tlačítko „Přihlásit se“ slouží jako brána do administrátorské části aplikace. Proces přihlášení je navržen tak, aby zajistil, že přístup do této sekce mají pouze autorizovaní uživatelé s platnými přihlašovacími údaji. Po úspěšném přihlášení má administrátor plnou kontrolu nad daty a funkcemi spojenými s aplikací.



Obrázek 10: Stránka přihlášeni do systému.

2.3.4 Hlavní stránka pro administrátory

Tato stránka (obr. 11) slouží jako hlavní rozhraní pro správu databáze. Zde mohou administrátoři spravovat všechny potřebné údaje, včetně informací o klientech, výkonech, zaměstnancích, diagnózách, zprávách a dalších. Tato stránka poskytuje administrátorům kompletní přístup k různým datům v databázi a umožňuje provádět změny, přidávat nové záznamy nebo upravovat stávající informace.



Obrázek 11: Hlavní stránka pro administrátory.

V horní části stránky se zobrazují dostupné záložky: Klienti, Zaměstnanci, Výkony, Diagnózy, Pojišťovny a Zprávy. V každé z těchto záložek lze provádět různé operace s daty. Výchozí záložka je Klienti, kde se zobrazují jména a příjmení klientů, jejich rodná čísla a místo, kde se nacházejí. Když vyberete položku, zvýrazní se zelenou barvou a aktivují se dvě tlačítka: „Podrobnosti“ a „Odstranit“. Tato tlačítka umožňují buď smazat položku, nebo zobrazit podrobnější informace o klientovi.

Na hlavní stránce pro administrátory, konkrétně na stránce *Klienti*, se nachází tlačítko „Import z Excelu“, které umožňuje načíst data z excelového souboru a uložit je do databáze. Pro úspěšné zpracování musí data ve zdrojovém souboru být ve specifickém formátu, který zahrnuje následující položky:

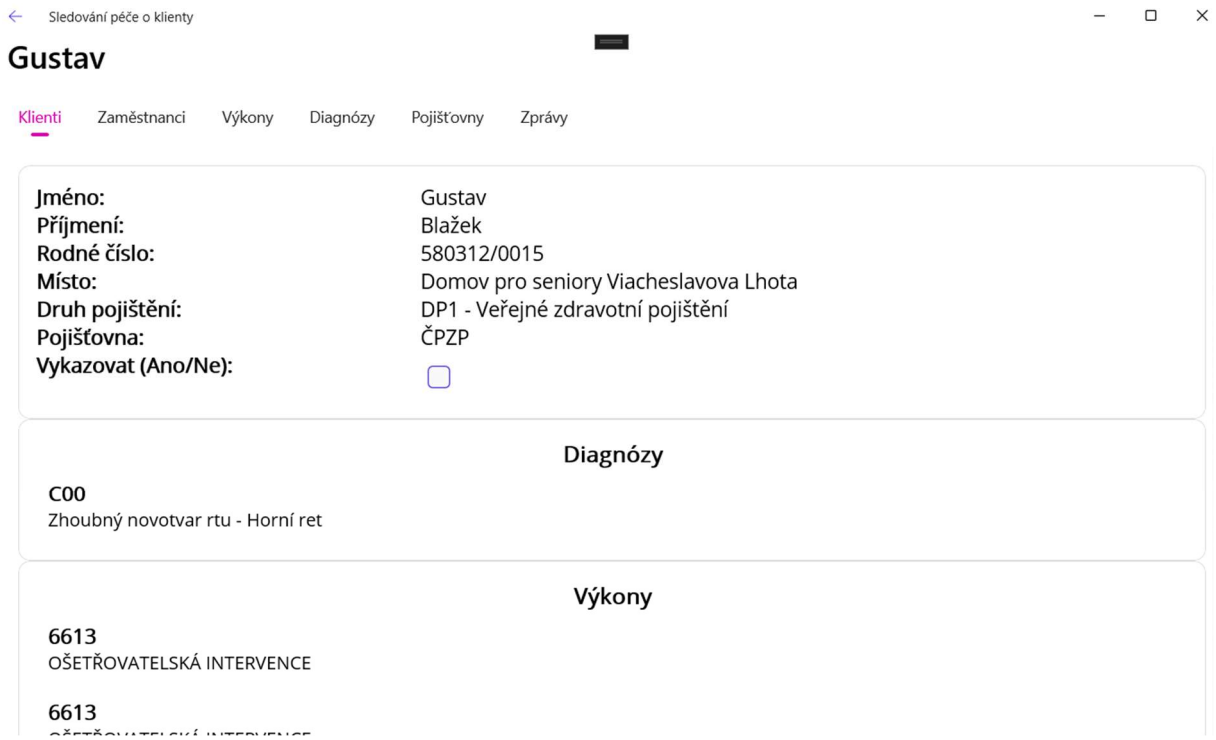
- Datum – datum, kdy byl úkon proveden,
- Klient (Jméno, Příjmení) – jméno a příjmení klienta,
- Pojišťovna (Kód – Název) – kód a název pojišťovny, která zajišťuje pokrytí,
- Zaměstnanec (Jméno, Příjmení) – jméno a příjmení zaměstnance, který úkon provedl,
- Výkon – typ provedeného úkonu,
- Body – počet bodů za úkon,
- Cena – cena spojená s úkonem,
- Pracoviště – místo, kde byl úkon proveden,
- Druh pojištění – typ pojištění klienta.

Tato funkce je mimořádně užitečná, protože umožňuje rychlé a efektivní importování většího množství dat do aplikace, aniž by bylo nutné provádět ruční zadávání. Administrátoři mohou takto snadno aktualizovat informace v databázi a udržovat přesné a aktuální záznamy o klientech, pojišťovnách, zaměstnancích a výkonech. Pro úspěšný import je však nutné zajistit, aby data v excelovém souboru byla v souladu s požadovaným formátem. Tímto způsobem lze zaručit bezproblémové zpracování a správné uložení dat do databáze.

Celkově záložky v horní části stránky umožňují rychle přepínat mezi různými sekcemi aplikace, což dává přehled o celkovém fungování systému. Výchozí záložka pro klienty je obzvláště užitečná, protože zde můžete spravovat informace o klientech a provádět různé operace podle potřeb. Tímto způsobem je možné efektivně pracovat s daty a zajišťovat, že informace o klientech jsou aktuální a přesné.

2.3.4 Detailní profil klienta

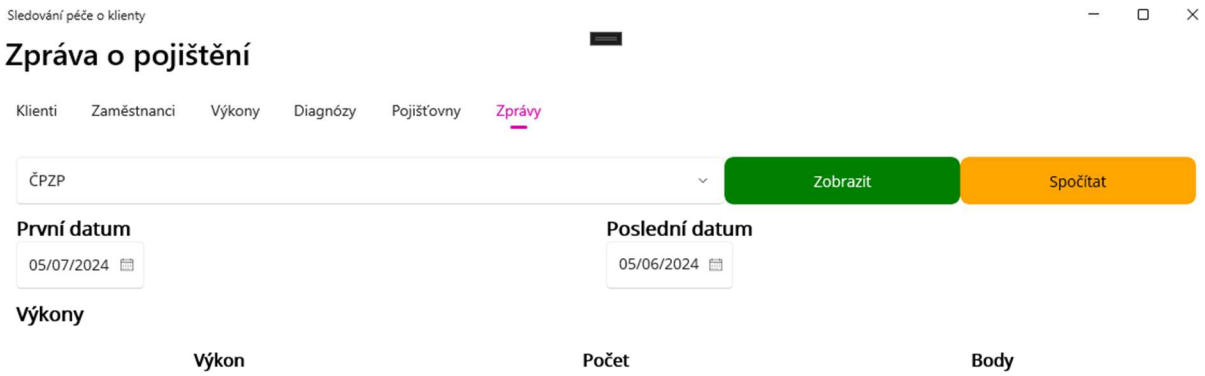
Pokud klikneme na tlačítko „Podrobnosti“, zobrazí se detailní profil klienta (obr. 12), kde můžeme uvidět jeho základní údaje, jako je jméno, příjmení, typ pojištění, rodné číslo, místo pobytu a pojišťovací společnost klienta. Dále zde najdeme atribut, který určuje musíme jsme vykazovat klienta nebo ne. Dále je zobrazeno diagnózy klienta. Kromě toho jsou v této sekci uvedeny výkony, které byly u klienta provedeny během jeho pobytu v domově důchodců.



Obrázek 12: Detailní profil klienta.

2.3.4 Zprávy

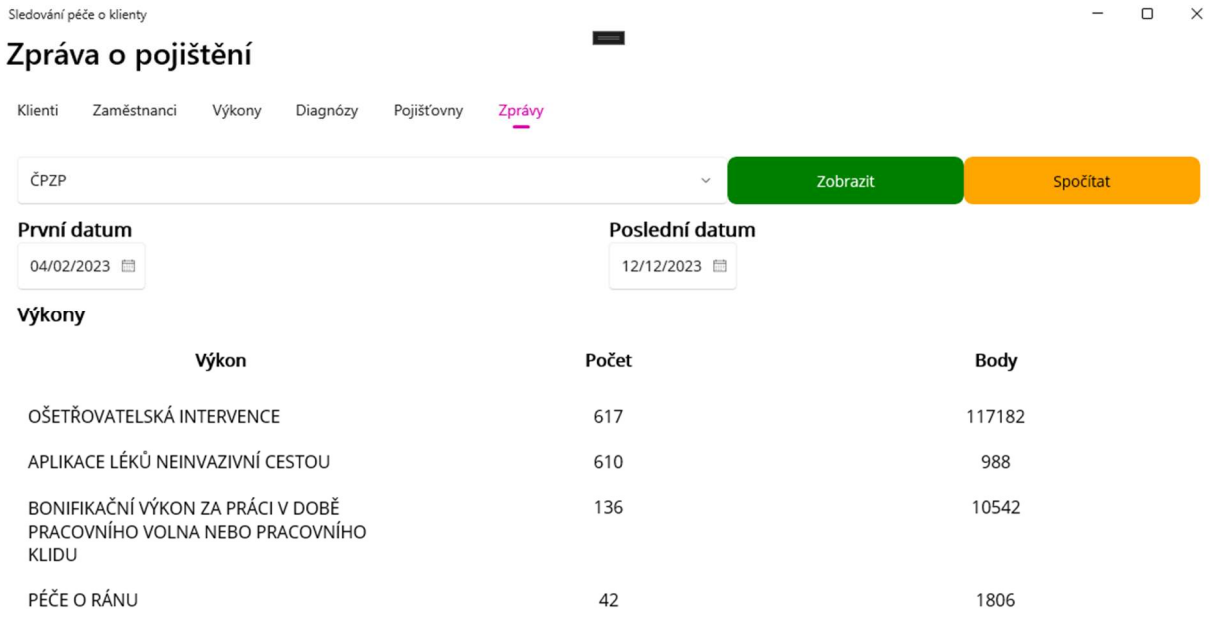
Po otevření stránky Zprávy uvidíme obrazovku (obr. 13), kde lze zobrazit operace provedené během zadaného období. Tato funkce umožňuje zjistit celkový počet provedených operací, počet získaných bodů a názvy jednotlivých operací. Tímto způsobem si administrátory mohou udělat ucelený přehled o výkonech za určité časové období.



Obrázek 13: Stránka zprávy.

Aby bylo možné zobrazit požadované informace, je nutné nejprve vybrat časový rozsah pomocí prvků "První datum" a "Poslední datum". Tyto prvky umožňují přesně definovat, od kdy do kdy chcete zobrazit data. Kromě toho je třeba vybrat pojišťovací společnost, které se zadané operace týkají.

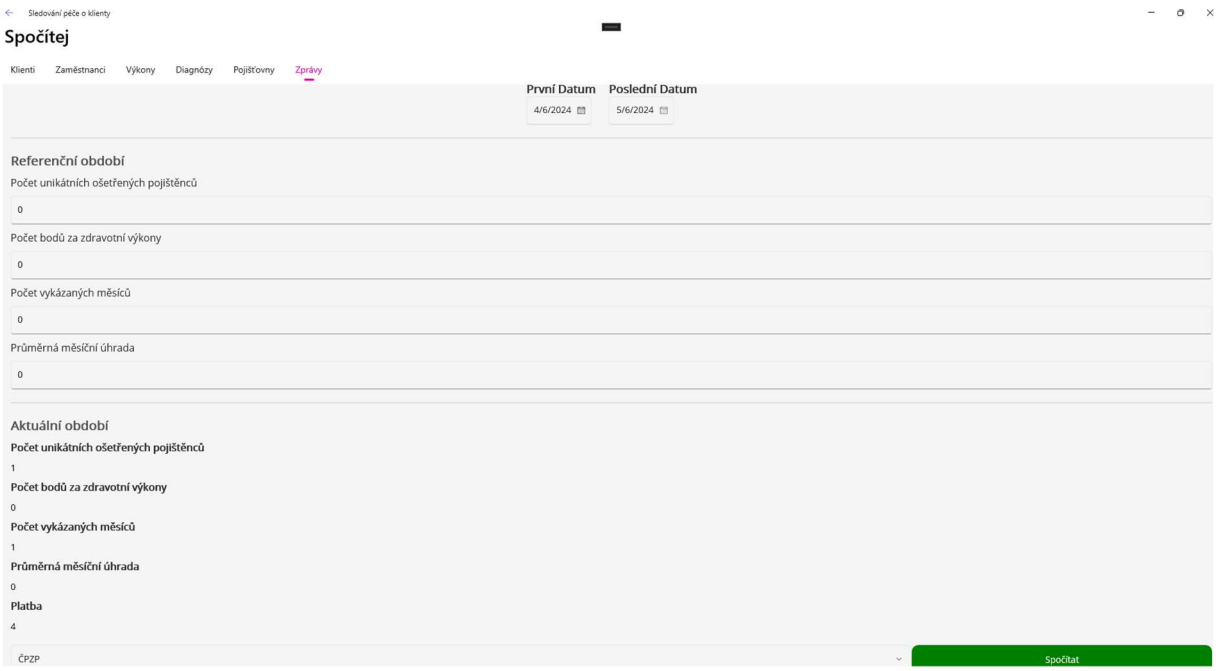
Jakmile jsou všechny parametry nastaveny, stačí kliknout na tlačítko „Zobrazit“. Po jeho stisknutí aplikace zobrazí seznam operací provedených v určeném období (obr. 14), včetně počtu operací, celkového počtu bodů a názvů jednotlivých operací. Tento přehled je užitečný pro administrátory, kteří potřebují sledovat výkony v daném období, a poskytuje jasný pohled na výsledky práce.



Obrázek 14: Seznam operací provedených v určeném období.

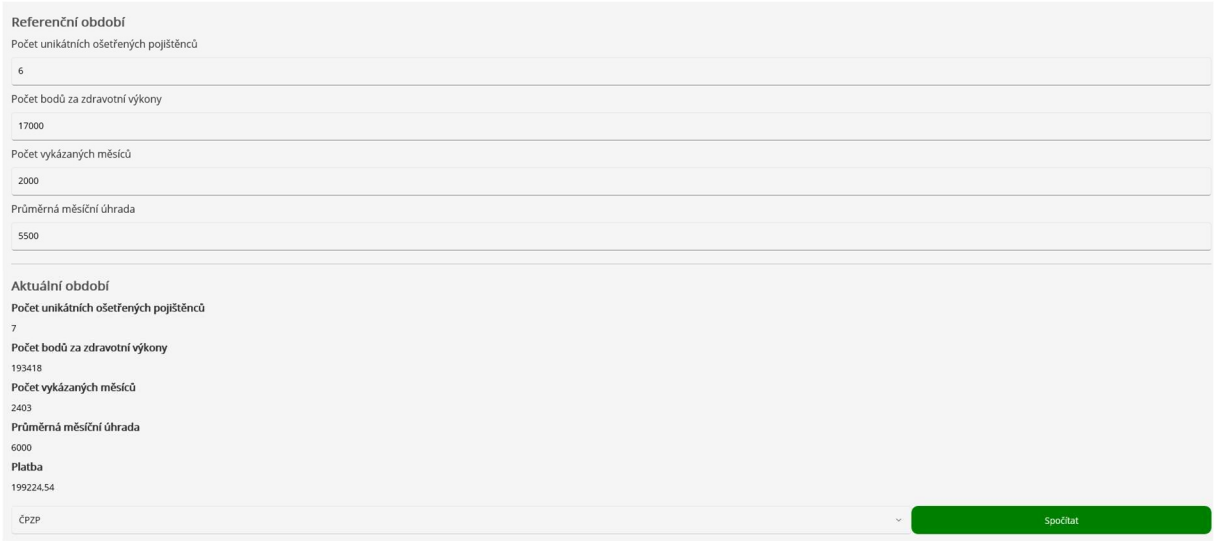
2.3.4 Zprávy Spočítej

Po otevření stránky Zprávy se zobrazí obrazovka (obr. 13), kde můžete kliknout na tlačítko "Spočítat". Tímto kliknutím budete přesměrováni na stránku pro výpočet dat pro konkrétní pojišťovací společnost (obr. 15).



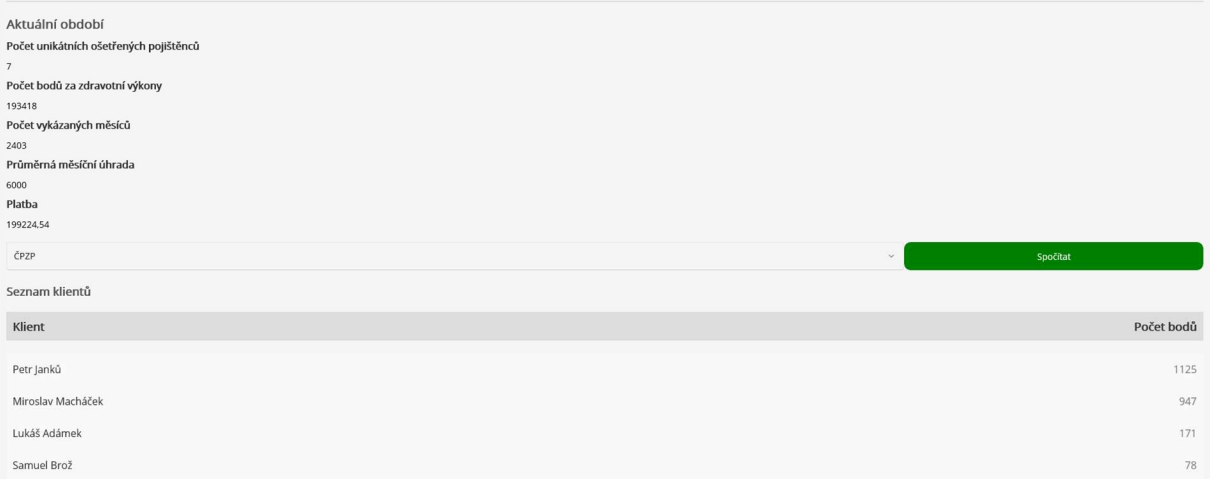
Obrázek 15: Stránka spočítej ve zprávách.

Zde můžete zadat časové období, počet unikátních pojištěnců, kteří byli ošetřeni, počet bodů za zdravotní výkony, průměrnou měsíční úhradu, a také počet vykázaných měsíců v referenčním roce a poté kliknout na tlačítko „Spočítat“. Matematický algoritmus, který je součástí této aplikace, vypočítá hodnoty pro zvolené období a zobrazí výsledky v části stránky s názvem „Aktuální období“ (obr. 16).



Obrázek 16: Vypočítané hodnoty.

Po spočítáni hodnot pod tlačítkem „Spočítat“ zobrazí se seznam klientů s jejich počtem výkonů během zadaného období (obr. 17). Po stisknutí na nadpis “Počet bodů” tabulka je seřazena podle počtu bodů. Tímto způsobem lze identifikovat klienty, jejichž výkony by mohly být optimalizovány.



Obrázek 17: Seznam klientů s jejich počtem výkonů.

ZÁVĚR

Cílem této aplikace bylo vytvořit aplikace, který usnadní správu lékařských úkonů v domovech důchodců. Vyvinutá aplikace umožňuje efektivní správu lékařských úkonů a poskytuje nástroje pro jejich optimalizaci, což je důležité pro přesné vykazování zdravotním pojišťovnám a minimalizaci rizika krácení úhrad.

Práce byla rozdělena do dvou hlavních částí: teoretické a praktické. V teoretické části jsme podrobně prozkoumali technologie a nástroje použité při vývoji aplikace. Byly zde popsány klíčové jazyky jako C#, SQL a XAML, které byly využity pro různé aspekty vývoje. Tato část také objasnila význam databázové technologie SQLite a uvedla prostředí jako Microsoft Visual Studio a Oracle SQL Developer Data Modeler, které se použily během vývoje. Architektura MVVM (Model-View-ViewModel) byla popsána jako základní struktura, která odděluje uživatelské rozhraní od obchodní logiky a umožňuje snadnější testování a údržbu. Popis knihovny SQLite-net-pcl ukázal, jak aplikace pracuje s relačními databázemi a umožňuje snadnou manipulaci s daty.

V praktické části práce jsme se zaměřili na implementaci aplikace. Byla zde popsaná struktura databáze a souborová organizace projektu, včetně popisu jednotlivých tříd a jejich interakce. Důraz byl kladen na uživatelské rozhraní (UI), které je klíčové pro interakci uživatelů s aplikací. Navržená aplikace umožňuje přidávat, upravovat a mazat data, která jsou následně ukládána do relační databáze. Tato část také popsala matematický algoritmus pro optimalizaci lékařských úkonů, který je nezbytný pro zvýšení přesnosti zpracování lékařských záznamů a snížení možnosti chyb při vykazování úkonů pojišťovnám.

Tato aplikace poskytuje efektivní nástroje pro správu lékařských úkonů v domovech důchodců a výrazně přispívá k optimalizaci vykázaných úkonů. Díky této aplikaci mohou administrátoři domovů důchodců lépe spravovat data a minimalizovat riziko krácení úhrad ze strany pojišťoven, což je zásadní pro finanční stabilitu a efektivní provoz těchto institucí. Navržené řešení by tak mohlo sloužit jako užitečný nástroj pro organizace působící v této oblasti a poskytovat jim lepší kontrolu nad procesy spojenými s lékařskou péčí.

POUŽITÁ LITERATURA

1. KUČEROVÁ, Zdeňka, 2015. Úhrada zdravotních služeb v pobytových zařízeních sociálních služeb. In: *vzp.cz* [online]. VZP, 2015 [cit. 2024-02-15]. Dostupné z: https://www.vzp.cz/poskytovatele/informace-pro-praxi/poradna/uhrada-zdravotnich-sluzeb-v-pobytovych-zarizenich-socialnich-sluzeb
2. ČESKO, 2021. Vyhláška č. 482/2021 Sb. Vyhláška, kterou se mění vyhláška č. 134/1998 Sb., kterou se vydává seznam zdravotních výkonů s bodovými hodnotami, ve znění pozdějších předpisů. In: *ftp.aspi.cz* [online]. ČESKO, 2021 [cit. 2024-02-18]. Dostupné z: https://ftp.aspi.cz/opispdf/2021/215-2021.pdf
3. ÚZIS ČR, 2016. Registrační list - 06620. In: *szv.mzcr.cz* [online]. Seznam zdravotních výkonů 2.0.193.7, 2016 [cit. 2024-04-01]. Dostupné z: https://szv.mzcr.cz/Vykon/Detail/06620
4. MICROSOFT, 2024. C# Introduction. In: *learn.microsoft.com* [online]. Microsoft, 2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction
5. MICROSOFT, 2024. Prohlídka jazyka C#. In: *learn.microsoft.com* [online]. Microsoft, 2024 [cit. 2024-03-11]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/csharp/tour-of-csharp
6. FUTURELEARN, 2023. What is SQL and what is it used for? In: *futurelearn.com* [online]. FutureLearn, 2023 [cit. 2024-03-17]. Dostupné z: https://www.futurelearn.com/info/blog/what-sql-used-for
7. AWS, 2024. What is SQL? In: *aws.amazon.com* [online]. Amazon Web Services, Inc. or its affiliates, 2024 [cit. 2024-03-17]. Dostupné z: https://aws.amazon.com/what-is/sql/?nc1=h_ls
8. FILEFORMAT, 2024. What is a XAML file? In: *docs.fileformat.com* [online]. Openize Pty Ltd, 2024 [cit. 2024-04-06]. Dostupné z: https://docs.fileformat.com/web/xaml/
9. MICROSOFT, 2022. WPF Globalization and Localization Overview. In: *learn.microsoft.com* [online]. Microsoft, 2022 [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/wpf-globalization-and-localization-overview?view=netframeworkdesktop-4.8&redirectedfrom=MSDN#Localize_a_WPF_Application
10. MICROSOFT, 2023. What's a Universal Windows Platform (UWP) app? In: *learn.microsoft.com* [online]. Microsoft, 2023 [cit. 2024-04-06]. Dostupné z:

https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

11. TUTORIALSPOINT, 2024. SQLite – Overview. In: *tutorialspoint.com* [online]. Tutorialspoint, 2024 [cit. 2024-03-23]. Dostupné z: https://www.tutorialspoint.com/sqlite/sqlite_overview.htm
12. ITNETWORK, 2024. Úvod do SQLite. In: *itnetwork.cz* [online]. Itnetwork, 2024 [cit. 2024-03-23]. Dostupné z: https://www.itnetwork.cz/sqlite/sqlite-tutorial-uvod-a-priprava-prostredi
13. BRAINHUB, 2024. .NET MAUI in a nutshell. In: *brainhub.eu* [online]. Brainhub, 2024 [cit. 2024-04-13]. Dostupné z: https://brainhub.eu/library/net-maui-in-nutshell#:~:text=NET%20MAUI%20is%20a%20cross,ideal%20for%20multi%2Ddevice%20users
14. MICROSOFT, 2023. What is .NET MAUI? In: *learn.microsoft.com* [online]. Microsoft, 2023 [cit. 2024-04-13]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0
15. VAHID CHESHMY, 2019. CLR, BCL, CIL, and assemblies in .Net. In: *medium.com* [online]. Medium, 2019 [cit. 2024-04-13]. Dostupné z: https://medium.com/@v.cheshmi/clr-bcl-cil-and-assemblies-in-net-framework-3de8bf09e781
16. EDIN ŠAHBAZ, 2023. MVVM Implementation Using .NET. In: medium.com [online]. Medium, 2023 [cit. 2024-05-05]. Dostupné z: https://medium.com/@edin.sahbaz/mvvm-implementation-using-net-community-toolkit-mvvm-source-generator-in-net-maui-810a137af47e
17. MICROSOFT, 2023. Model-View-ViewModel (MVVM). In: *learn.microsoft.com* [online]. Microsoft, 2023 [cit. 2024-04-13]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/architecture/maui/mvvm
18. MICROSOFT, 2022. Data binding and MVVM. In: *learn.microsoft.com* [online]. Microsoft, 2022 [cit. 2024-04-13]. Dostupné z: https://learn.microsoft.com/en-us/windows/uwp/data-binding/data-binding-and-mvvm
19. VZP, 2023. Dodatek č. 1 ke Zvláštní smlouvě o poskytování a úhradě ošetřovatelské péče v zařízeních sociálních služeb poskytujících pobytové sociální služby. In: *www.vzp.cz* [online]. VZP, 2023 [cit. 2024-02-28]. Dostupné z: https://www.vzp.cz/Contract/DownloadFile/1542571

PŘÍLOHY

Příloha A: Zdrojový kód aplikace ClientCareTracker.

Příloha B: DDL Script pro vytváření databáze.

Příloha A: Zdrojový kód aplikace ClientCareTracker

Tato příloha obsahuje *zip* archiv s projektem z Visual Studia 2022 jménem ClientCareTracker.

Příloha B: DDL Scripty pro vytváření databáze

Tato příloha obsahuje DDL script pro vytváření databáze se jménem vytvoreniDatabaze.ddl