Vývoj docházkové aplikace pomocí WPF s použitím MVVM vzoru

Bc. Tomáš Ševců

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tomáš Ševců**
Osobní číslo:	**A21507**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Vývoj docházkové aplikace pomocí WPF s použitím MVVM vzoru**
Téma práce anglicky:	**Development of Attendance Application in WPF Using MVVM Pattern**

Zásady pro vypracování

1. Proveďte rešerši existujících řešení zabývajících se aplikacemi na docházku.
2. Popište framework WPF a MVVM vzor.
3. Ukažte postupy/části kódu WPF a MVVM vzoru pro nejdůležitější části.
4. Vytvořte návrh jednotlivých částí a pracovní postupy – UML.
5. Naprogramujte docházkovou aplikaci.
6. Prezentujte vytvořenou aplikaci.
7. Vyhodnoťte vytvořenou aplikaci a její reálné použití.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. YUEN, Sheridan. *Mastering Windows Presentation Foundation: Build responsive UIs for desktop applications with WPF*. 2nd Edition. Birmingham: Packt Publishing, 2020. ISBN 978-1838643416.
2. CHOWDHURY, Kunal. *Windows Presentation Foundation Development Cookbook: 100 recipes to build rich desktop client applications on Windows*. 2nd Edition. Birmingham: Packt Publishing, 2018. ISBN 978-1788399807.
3. WEIL, Arnaud. *Learn WPF MVVM: XAML, C# and the MVVM pattern*. Lulu.com, 2016. ISBN 978-1326847999.
4. SMITH, Jon P. *Entity Framework Core in Action*. 2nd Edition. Shelter Island: Manning Publications, 2018. ISBN 978--1617294563.
5. BOCK, Lisa. *Identity Management with Biometrics: Explore the latest innovative solutions to provide secure identification and authentication*. Birmingham: Packt Publishing, 2020. ISBN 978-1838988388.
6. JIANG, Richard, Danny CROOKES, Ahmed BOURIDANE, Somaya AL-MAADEED a Azeddine BEGHDADI. *Biometric Security and Privacy: Opportunities & Challenges in The Big Data Era*. Springer, 2017. ISBN 978-3319837031.

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Jméno, příjmení: Tomáš Ševců

Název diplomové práce: Vývoj docházkové aplikace pomocí WPF s použitím MVVM vzoru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 5. 5. 2023

Tomáš Ševců, v.r.
.....
podpis studenta

ABSTRAKT

Diplomová práce nazvaná „Vývoj docházkové aplikace pomocí WPF s použitím MVVM vzoru“ se zaměřuje na popis a vývoj aplikace pomocí Windows Presentation Foundation s použitím vzoru Model-View-ViewModel. Cílem práce je popsat framework WPF a návrhový vzor MVVM, předvést postupy a kód pro nejdůležitější části WPF a MVVM vzoru a pomocí těchto postupů naprogramovat docházkovou aplikaci. Před vytvořením výsledné aplikace jsou provedeny požadavky a cíle aplikace, které jsou následně naprogramovány a popsány. V úvodních kapitolách teoretické části jsou popsány použité technologie .NET, WPF, EF a návrhový vzor MVVM. Následně práce popisuje vývoj pomocí WPF a jeho hlavní postupy a části. Nakonec je pomocí zmíněných postupů vytvořena šablona WPF aplikace, která obsahuje základní funkcionality jako navigaci, příkazy, služby a skladiště. V praktické části se práce nejprve věnuje docházkové aplikaci obecně a existujícím řešením. Následně jsou provedeny plány a cíle funkcí a požadavků vytvářené docházkové aplikace. Docházková aplikace je v dalším kroku realizována, je popsán její vývoj a představení jejích funkcí a možností. Nakonec je aplikace zhodnocena, zda splňuje určené plány a cíle a jaké má další možnosti rozvoje.

Klíčová slova: vývoj softwaru, docházková aplikace, WPF, MVVM vzor, technologie .NET, EF

ABSTRACT

The thesis entitled "Development of Attendance Application in WPF Using MVVM Pattern" focuses on the description and development of an application using Windows Presentation Foundation with the use of the Model-View-ViewModel pattern. The aim of the thesis is to describe the WPF framework and MVVM design pattern, demonstrate the procedures and code for the most important parts of the WPF and MVVM pattern, and program an attendance application using these procedures. Before creating the final application, the requirements and goals of the application are defined, which are then programmed and described. In the introductory chapters of the theoretical part, the used technologies are

described such as .NET, WPF, EF and the MVVM design pattern. The thesis then describes development using WPF and its main procedures and parts. Finally, using the mentioned procedures, a WPF application template is created, which includes basic functionalities such as navigation, commands, services and storage. In the practical part, the thesis first deals with the attendance application in general and existing solutions. Subsequently, the plans and goals for the functions and requirements of the created attendance application are established. The attendance application is implemented in the next step, and its development and introduction of its functions and capabilities are described. Finally, the application is evaluated to determine if it meets the defined plans and goals and what other development possibilities it may have.

Keywords: software development, attendance application, Windows Presentation Foundation, Model-View-ViewModel pattern, NET technologies, Entity Framework

Rád bych poděkoval vedoucímu diplomové práce Ing. Pavlu Vařachovi, Ph.D. za jeho cenné rady, připomínky a vstřícnost při vypracovávání diplomové práce.

Také bych rád poděkoval své družce za pomoc, podporu a stravu při vypracovávání diplomové práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Sledování docházky je v mnoha organizacích zásadním úkolem, proto je nezbytné mít spolehlivý a účinný systém. V posledních letech využití technologie transformovalo sledování docházky z manuálních na automatizované systémy. Tento vývoj vedl k vytvoření docházkových aplikací, které dokážou přesně evidovat docházku zaměstnanců, snižují chybovost a šetří čas. Softwarové aplikace výrazně usnadnily a zefektivnily sledování docházky a práce "Vývoj docházkové aplikace pomocí WPF s použitím vzoru MVVM" se zaměřuje na vývoj takové aplikace.

Vývoj takových aplikací vyžaduje použití robustních rámců a vzorů, které mohou zajistit efektivitu a spolehlivost. Jedním z takových frameworků je .NET, široce používaná platforma pro vytváření aplikací pro Windows. Grafický subsystém .NET Windows Presentation Foundation (WPF) nabízí výkonnou platformu pro vytváření desktopových aplikací s bohatým uživatelským rozhraním. Návrhový vzor Model-View-ViewModel (MVVM) je oblíbený vzor používaný k vývoji aplikací WPF, který umožňuje jasné oddělení aplikační logiky a uživatelského rozhraní.

Hlavním cílem této práce je popsat vývoj docházkové aplikace využívající WPF se vzorem MVVM. Teoretická část práce představuje relevantní technologie a koncepty, jako jsou .NET, WPF, Entity Framework (EF) a vzor MVVM. Následně jsou popsány nejdůležitější části WPF a MVVM, pomocí kterých je vytvořena aplikační šablona WPF, která obsahuje navigaci, příkazy, služby a skladiště. Praktická část práce je zaměřena na vývoj docházkové aplikace, pokrytí jejích požadavků, proces vývoje a vyhodnocení. Nejprve je docházková aplikace představena obecně a jsou popsány existující řešení zabývající se docházkovými aplikacemi. Následně jsou provedeny plány a cíle funkcí a požadavků vytvářené docházkové aplikace. Dále bude popsán proces vývoje spolu s představením aplikace. Na závěr zhodnocení vytvořené aplikace a její reálné využití, zjištění splnění zadaných plánů a cílů a možnosti jejího dalšího rozvoje. Prostřednictvím této práce čtenář porozumí frameworku WPF, vzoru MVVM a vývoji docházkové aplikace, kterou lze v budoucnu reálně využít jako docházkový systém ve firmě.

I. TEORETICKÁ ČÁST

1 POUŽITÉ TECHNOLOGIE

1.1 Platforma .NET

.NET je bezplatná multiplatformní open source vývojářská platforma vyvinutá společností Microsoft pro vytváření a provozování moderních aplikací, včetně aplikací pro stolní počítače, webových aplikací a mobilních aplikací. .NET poskytuje sadu nástrojů, knihoven a jazyků, včetně C# a Visual Basic .NET (VB.NET), které mohou vývojáři používat k vytváření široké škály aplikací. 94[1]

Platforma .NET obsahuje běhové prostředí (runtime) .NET, které poskytuje společné spouštěcí prostředí pro aplikace vytvořené pomocí technologií .NET, a sadu knihoven a služeb, které usnadňují vytváření a spouštění těchto aplikací. Runtime .NET poskytuje aplikacím zabezpečení, správu paměti a další systémové služby a umožňuje vývojářům soustředit se na psaní kódu, který implementuje funkce jejich aplikací. [2]

.NET také obsahuje sadu knihoven a nástrojů pro vytváření webových aplikací, jako je ASP.NET, a sadu knihoven a nástrojů pro vytváření mobilních aplikací, jako je Xamarin. .NET navíc poskytuje sadu knihoven pro přístup k datům, včetně Entity Framework pro přístup k relačním databázím a LINQ pro dotazování a manipulaci s daty. [3]

Framework .NET poskytuje bohatou sadu knihoven a nástrojů pro vytváření cloudových aplikací, včetně ASP.NET pro vývoj webu, Azure SDK pro vytváření cloudových aplikací a Entity Framework pro přístup k datům. .NET Core, multiplatformní verze .NET, umožňuje vývojářům vytvářet cloudové aplikace, které lze provozovat na více platformách, včetně Windows, Linuxu a macOS.

Microsoft Azure, platforma cloud computingu od společnosti Microsoft, poskytuje řadu služeb pro vývojáře .NET, včetně virtuálních strojů, úložiště, databází a bezserverových počítačů, což vývojářům umožňuje snadno vytvářet a nasazovat cloudové aplikace.

Kromě toho .NET poskytuje řadu funkcí, které jsou zásadní pro vytváření cloudových aplikací, jako je podpora architektury mikroslužeb, kontejnerizace a nasazení do cloudové infrastruktury. [4]

.NET je oblíbená platforma pro vývoj her, protože poskytuje řadu nástrojů, knihoven a frameworků pro vytváření vysoce výkonných her pro různé platformy, včetně Windows, Xbox a Playstation. Framework .NET poskytuje bohatou sadu knihoven pro vývoj her, včetně DirectX pro grafiku, XNA pro vývoj her a programovacího jazyka C#. Některým

populární herní enginy, které používají .NET, zahrnují Unity a Unreal Engine, které lze použít k vývoji 2D a 3D her pro stolní, mobilní a webové platformy. Kromě toho lze běhové prostředí a knihovny .NET Core použít k vývoji multiplatformních her, což vývojářům umožňuje zacílit na více platforem pomocí jediné kódové základny. [5]

Celkově .NET poskytuje komplexní a flexibilní platformu pro vytváření a provoz moderních aplikací a je široce používán vývojáři pro vytváření široké škály aplikací v různých doménách.

[6]

1.2 Windows Presentation Foundation

Windows Presentation Foundation zkráceně WPF je architektura uživatelského rozhraní, která je nezávislá na rozlišení a používá vektorový vykreslovací modul navržený tak, aby využíval moderní grafický hardware. WPF nabízí ucelenou sadu funkcí pro vývoj aplikací, mezi které patří jazyk XAML[1], ovládací prvky, datové vazby, rozložení, 2D a 3D grafika, animace, styly, šablony, dokumenty, média, text a typografie.

WPF je součástí platformy .NET. WPF byl poprvé představen s .NET Framework 3.0 v roce 2006. V současnosti[2] nejnovější stabilní verze WPF funguje pod .NET Framework 6.0 z roku 2022. [7]

WPF používá jako svůj značkovací jazyk XAML, který umožňuje vývojářům deklarativně popsat strukturu a vzhled uživatelského rozhraní. WPF také podporuje použití programovacích jazyků .NET, jako je C# a Visual Basic, pro psaní logiky a přidávání funkcí do aplikací. [8]

Jednou z klíčových výhod WPF je jeho schopnost oddělit uživatelské rozhraní od základního kódu, což umožňuje vývojářům snadněji upravovat vzhled aplikace, aniž by to ovlivnilo její funkčnost. To také usnadňuje údržbu a aktualizaci aplikací v průběhu času.

WPF nahrazuje starší technologii Windows Forms. Windows Forms byl představen jako součást .NET Framework 1.0, která byla vydána v roce 2002, a byla navržena tak, aby

[1] XAML - eXtensible Application Markup Language
[2] Leden 2023

vývojářům poskytla způsob, jak vytvářet tradiční aplikace založené na Windows s jednoduchým a přímočarým uživatelským rozhraním.

WPF i Windows Forms jsou dnes stále široce používány. WPF je obecně považován za nástupce Windows Forms, protože poskytuje flexibilnější a výkonnější způsob vytváření uživatelských rozhraní. WPF je také vhodnější pro vytváření moderních aplikací s pokročilou grafikou, animacemi a multimédii, zatímco Windows Forms je vhodnější pro jednodušší aplikace s přímočarým uživatelským rozhraním.

[9] [10] [11]

1.2.1 Vývojové prostředí

Jako vývojové prostředí (IDE) se pro vývoj aplikací WPF nejčastěji využívá Visual Studio. Visual Studio je nejrozšířenější integrované vývojové prostředí (IDE) pro vývoj aplikací WPF. Visual Studio poskytuje řadu funkcí a nástrojů, které usnadňují vytváření aplikací WPF, včetně vizuálního editoru, editoru kódu, ladicích nástrojů a řady šablon a startovacích projektů.

Visual Studio podporuje vývoj WPF poskytováním vizuálního editoru pro vytváření uživatelských rozhraní pomocí XAML a podporuje použití C# a Visual Basic jako programovacích jazyků. Vizuální editor poskytuje rozhraní přetahování pro vytváření uživatelských rozhraní a editor kódu poskytuje bohatou sadu funkcí pro psaní, testování a ladění kódu.

Visual Studio navíc obsahuje řadu nástrojů pro správu závislostí, testování a nasazování aplikací a integraci s dalšími vývojovými nástroji a službami. Díky své výkonné sadě funkcí, podpoře široké škály vývojových technologií .NET a integraci s dalšími technologiemi společnosti Microsoft je Visual Studio preferovanou volbou pro mnoho vývojářů WPF.

[12] [13]

1.2.2 C#

C# je jedním z programovacích jazyků, které lze použít při vývoji WPF. WPF je součástí .NET frameworku a je navržen pro použití s jakýmkoliv programovacím jazykem .NET, včetně C#, Visual Basic a F#.

C# je oblíbenou volbou pro vývoj WPF díky své moderní syntaxi, silné typové bezpečnosti a podpoře objektově orientovaného programování. Ve WPF je uživatelské rozhraní definováno v XAML, zatímco kód za ním je napsán v C#. To umožňuje vývojářům používat C# k vytváření dynamických, interaktivních uživatelských rozhraní s komplexním chováním a využívat celou řadu knihoven a frameworků .NET pro vytváření robustních a škálovatelných aplikací.

C# je také výkonný a všestranný jazyk s velkou a aktivní komunitou vývojářů a je široce používán pro vývoj různých aplikací, od desktopových aplikací, jako je WPF, po webové aplikace, mobilní aplikace a hry. Všechny tyto faktory činí C# oblíbenou volbou pro vývoj WPF a je široce používán mnoha profesionálními vývojáři a organizacemi.

[6] [14]

1.2.3 XAML

XAML (eXtensible Application Markup Language) je značkovací jazyk používaný k definování struktury a vzhledu uživatelských rozhraní v aplikacích WPF. XAML je podobný HTML, ale místo toho, aby definoval strukturu webové stránky, definuje strukturu a vzhled prvků uživatelského rozhraní, jako jsou okna, tlačítka, textová pole a další ovládací prvky.

Ve WPF se XAML používá k definování rozvržení a vzhledu uživatelského rozhraní, zatímco chování rozhraní je definováno v kódu. Kód XAML definuje strukturu uživatelského rozhraní, jako je uspořádání ovládacích prvků a vztahy mezi nimi. Kód definuje vlastnosti, události a chování ovládacích prvků, jako je jejich vzhled, chování a interakce s uživatelem.

Kód XAML je kombinován s kódem na pozadí, aby se vytvořila konečná aplikace. WPF používá kód XAML k vytvoření stromu objektů, který představuje uživatelské rozhraní. Tyto objekty se pak používají k vykreslení rozhraní na obrazovku a k reakci na interakce uživatele.

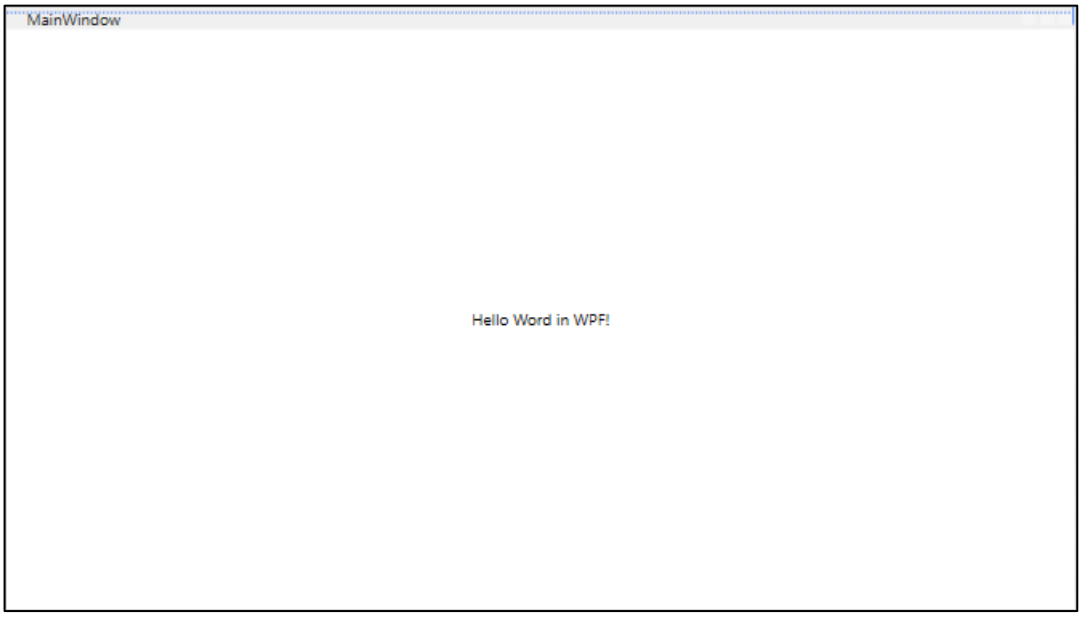
[9] [10] [15]

Na Obrázek 1 lze vidět stromovou strukturu, kde Window je kořenový element. Elementy, někdy označovány jako tagy, jsou zapisovány do lomených závorek. Element Window

obsahuje atribut x:Class, který spojuje XAML s částečným kódem třídy definovaným za ním, atributy xmlns jsou jmenné prostory. V kořenovém elementu je Grid, který dál obsahuje TextBlock, který vypisuje textový popis, v tomto případě Hello Word in WPF!. Obrázek 2 zobrazuje, jak napsaný kód XAML vypadá.

```
1  <Window
2      x:Class="HelloWordWPF.MainWindow"
3      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
6      xmlns:local="clr-namespace:HelloWordWPF"
7      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8      Title="MainWindow"
9      Width="800"
10     Height="450"
11     mc:Ignorable="d">
12     <Grid>
13         <TextBlock Text="Hello Word in WPF!"
14                    VerticalAlignment="Center"
15                    HorizontalAlignment="Center"/>
16     </Grid>
17 </Window>
```

Obrázek 1 Ukázka XAML Hello Word aplikace



Obrázek 2 Vizualizace Hello Word aplikace

1.3 Model–View–Viewmodel

Model-View-ViewModel (MVVM) je architektonický vzor používaný ve WPF a dalších platformách založených na XAML pro vytváření aplikací. MVVM je variací vzoru Model-View-Controller (MVC), který se běžně používá při vývoji uživatelského rozhraní.

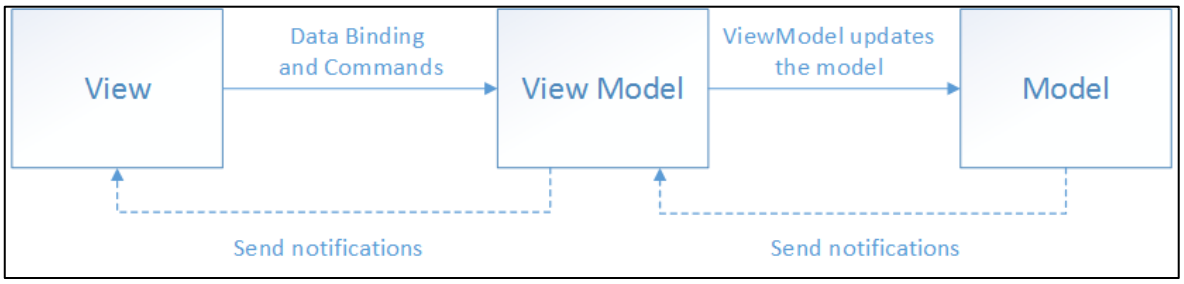
V MVVM Model představuje data a obchodní logiku aplikace. View (pohled) je uživatelské rozhraní a zobrazuje data z Model (modelu). ViewModel funguje jako prostředník mezi modelem a pohledem a je zodpovědný za transformaci dat z modelu do podoby, kterou lze snadno zobrazit v pohledu.

Hlavní výhodou MVVM je, že poskytuje jasné oddělení zájmů mezi Model, View a ViewModel. To usnadňuje vývoj, testování a údržbu aplikace, protože každou komponentu lze vyvíjet a testovat nezávisle. MVVM navíc usnadňuje vytváření opakovaně použitelných komponent, protože model a pohled lze snadno zaměnit, aniž by to ovlivnilo zbytek aplikace.

MVVM také umožňuje flexibilnější a dynamičtější uživatelské rozhraní, protože pohled lze snadno aktualizovat, aby odrážel změny v modelu, aniž by to ovlivnilo podkladová data. MVVM navíc poskytuje podporu pro datové vazby, což usnadňuje zobrazení a manipulaci s daty v uživatelském rozhraní.

Celkově je MVVM výkonná a flexibilní architektura, která se dobře hodí pro vývoj komplexních a dynamických uživatelských rozhraní ve WPF a dalších platformách založených na XAML.

[9] [16] [18]



Obrázek 3 MVVM vzor [17]

1.4 Entity Framework

Entity Framework (EF) je open source framework pro objektově relační mapování (ORM) pro platformu .NET. Poskytuje způsob, jak interagovat s relačními databázemi více objektově orientovaným způsobem, což umožňuje vývojářům pracovat s entitami a vztahy namísto psaní přímo SQL dotazů.

Entity Framework pomáhá vývojářům abstrahovat základní databázovou strukturu, což usnadňuje zaměřit se na obchodní logiku aplikace spíše než na nízko úrovňové detaily databázových interakcí. Tato abstrakce umožňuje vývojářům pracovat s objektovým modelem vyšší úrovně, což snižuje množství kódu potřebného k interakci s databází a zvyšuje udržovatelnost a testovatelnost aplikace.

Entity Framework podporuje řadu různých databázových systémů, včetně Microsoft SQL Server, Oracle, SQLite a dalších. Kromě toho poskytuje podporu pro LINQ (Language Integrated Query), což usnadňuje psaní silně typovaných, skládaných dotazů do databáze.

Celkově je Entity Framework výkonným nástrojem pro práci s relačními databázemi v .NET a poskytuje vývojářům řadu výhod, včetně zkrácení doby vývoje, lepší udržovatelnosti a snadnějšího testování.

[19] [20] [21]

2 VÝVOJ POMOCÍ WPF

Při vývoji jakékoliv aplikace je dobré dodržovat určité zásady a pravidla. V objektově orientovaném programování (OOP), což C# je, existuje sada návrhových principů SOLID, který představuje pět principů návrhu, aby byl návrh softwaru srozumitelnější, flexibilnější a udržitelnější. SOLID je zkratkou pro:

1. **S**ingle Responsibility Principle – Princip jednotné odpovědnosti – SRP

   Třídy by měly mít jedinou zodpovědnost / jediný důvod ke změně. Zodpovědnost právě za jednu věc, která by měla být vystižena názvem třídy.

2. **O**pen Closed Principle – Princip otevřenosti a uzavřenosti – OCP

   Třídy by měly být otevřené pro rozšiřování, ale uzavřené pro změny. Rozšíření funkčnosti by mělo být možné tím, že po přidání nového kódu není nutné zasahovat do existujícího kódu. Společné funkce jsou abstrahovány a definovány v odvozených třídách.

3. **L**iskov Subtitution Principle – Liskovové princip zaměnitelnosti – LSP

   Podtřídy by měly být zaměnitelné s jejich bázovými třídami.

4. **I**nterface Segregation Principle – Princip oddělení rozhraní – ISP

   Více specifických rozhraní je lepší než jedno univerzální rozhraní. Třídy by měly záviset pouze na rozhraních, která používají.

5. **D**ependency Inversion Principle – Princip obrácení závislostí – DIP

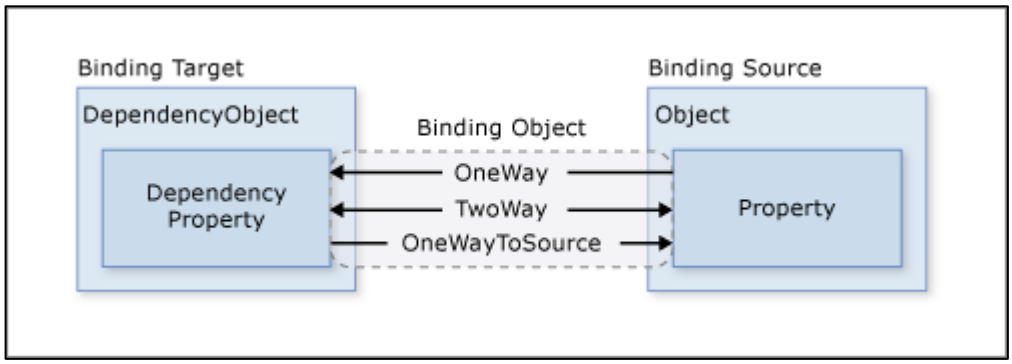
   Závislost by vždy měla být na abstraktním ne na konkrétním. Konkrétnější musí záviset na abstraktnějším ne naopak. Třídy, které závisejí na abstraktních rozhraních a třídách je snadné nahradit jinou konkrétní implementací.

[22]

2.1 DataBinding

DataBinding neboli vazba dat je funkce ve WPF, která umožňuje svázat data mezi prvky uživatelského rozhraní XAML a daty ze zdroje, jako je objekt v C#. Zároveň udržuje prvek aktualizovaný, když se data v objektu změní. Díky tomu je oddělená datová logika od logiky

uživatelského prostředí pro lepší údržnost a škálovatelnost. Data v XAML lze svázat například s prvky jako jsou textová pole, seznamy, tlačítka.



Obrázek 4 DataBinding [23]

Každá vazba má obvykle čtyři součásti: cílový objekt vazby, cílová vlastnost, zdrojový objekt a cesta k hodnotě ve zdroji vazby, která se má použít. Pokud bychom chtěli svázat text prvku `TextBox` s jménem zaměstnance `Employee.Name`, vazba by byla nastavena jako cílový objekt = `TextBox`, cílová vlastnost = `Text`, zdrojový objekt = `Employee` a cesta k hodnotě zdrojového objektu = `Name`.

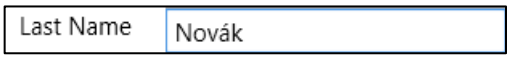
Vazba mezi objekty určuje i směr toku dat. Jsou čtyři možnosti: jednosměrná (OneWay) vazba, obousměrná vazba (TwoWay), opak jednosměrné (OneWayToSource) a OneTime.

- **OneWay** vazba automaticky aktualizuje cílovou vlastnost, tedy například text v TextBoxu, ale změny se nepřenesou zpět do zdroje. Ideální pro prvky, které jsou implicitně pro čtení.
- **TwoWay** vazba automaticky aktualizuje změny buď zdrojové vlastnosti nebo cílové vlastnosti, podle toho, kde nastala změna. Ideální pro upravitelné formuláře.
- **OneWayToSource** je opakem vazby OneWay a aktualizuje zdrojovou vlastnost, při změně cílové vlastnosti.
- **OneTime** vazba způsobí, že zdrojová vlastnost inicializuje cílovou vlastnost, ale následné změny se nešíří. Ideální pro statická data.

Aby bylo možné detekovat změny zdroje ve vazbě OneWay a TwoWay musí zdroj implementovat mechanismus oznámení změny jako je `INotifyPropertyChanged`. Ten zajišťuje, že při změně zdroje je o ní informován i cíl.

```
<TextBlock
    Grid.Row="5"
    Grid.Column="0"
    Text="Last Name" />
<TextBox
    Grid.Row="5"
    Grid.Column="1"
    Width="170"
    Text="{Binding LastName, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
```

Obrázek 5 DataBinding: XAML cílový objekt a cílová vlastnost



Obrázek 6 DataBinding: cílový objekt a cílová vlastnost

```
private string _lastName;

public string LastName
{
    get
    {
        return _lastName;
    }
    set
    {
        _lastName = value;
        OnPropertyChanged(nameof(LastName));
    }
}
```

Obrázek 7 DataBinding: C# zdrojový objekt a zdrojová vlastnost

```
public event PropertyChangedEventHandler? PropertyChanged;


protected void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Obrázek 8 DataBinding: OnPropertyChanged

Obrázek 5 zobrazuje vazbu dat v XAML cílový objekt je `TextBox` a cílová vlastnost je `Text`. Je spárován se zdrojovým objektem `LastName`, který je v tomto případě zároveň i zdrojová vlastnost. `Mode` je `TwoWay`, to znamená, že změny provedené ve formuláři (cíli) se přímo projevují i ve zdroji. Díky použití `TwoWay`, lze použít tento formulář i pro úpravu existujícího Last Name, tzn. například při úpravě vybraného kontaktu, a nejen při jeho vytváření. `UpdateSourceTrigger` s nastavením `PropertyChanged` zajišťuje, že se změna v UI projeví okamžitě. `TextBlock`, který prvku s vazbou předchází pouze zobrazuje text Last Name. Obrázek 6 zobrazuje náhled, jak vytvořené XAML vypadá. Obrázek 7 ukazuje zdrojový objekt. Funkce `OnPropertyChanged` zajišťuje, že i při vnější změně `LastName` je cílový objekt a cílová vlastnost upozorněna o změně. Na Obrázek 8 je znázorněno, jak funkce `OnPropertyChanged` může vypadat.

[9] [10] [23]

2.2 Dependency Injection

Dependency Injection (DI), česky vkládání závislostí, je návrhový vzor pro předávání objektů druhými objekty, aniž by na ni měla v době sestavování programu referenci. Bez využití DI objekt, který chce využít služby jiného objektu, Zároveň je spuštěna asynchronně, díky čemuž UI aplikace nezamrzne zodpovídá za celý jeho životní cyklus, tedy inicializaci a destrukci. V aplikaci s DI je objekt oproštěn od této správy, protože ji zařizuje poskytovatel závislostí (Dependency provider) neboli kontejner. Objektu, pro využití služeb jiných objektů, stačí pouze reference na poskytovatele závislostí, který poskytne očekávaný objekt a jeho služby. Poskytovatel závislostí zároveň zodpovídá za celý životní cyklus objektů, které poskytuje.

Ve WPF lze Dependency Injection použít k vytvoření volně propojených a vysoce udržovatelných aplikací. WPF podporuje DI pomocí kontejneru Inversion of Control (IoC), což je komponenta, která spravuje vytváření, řešení a životní cyklus objektů v aplikaci. Kontejner IoC je zodpovědný za vložení požadovaných závislostí do třídy, když je instanciována, čímž třídu osvobodí od nutnosti vytvářet nebo spravovat své vlastní závislosti.

Ve WPF existují dva hlavní typy DI:

Constructor Injection: Nejběžněji používaný typ Dependency Injection ve WPF. V této metodě jsou požadované závislosti předány třídě jako parametry v konstruktoru. Kontejner IoC je zodpovědný za vytvoření požadovaných závislostí a jejich předání třídě, když je vytvořena instance. To zajišťuje, že třída má vždy závislosti, které potřebuje ke správnému fungování.

```
public class ExampleClass
{
    private readonly IDataRepository _exampleRepository;

    public ExampleClass(IDataRepository customerRepository)
    {
        _exampleRepository = customerRepository;
    }

    public void SaveData(Data data)
    {
        _exampleRepository.SaveData(data);
    }
}
```

Obrázek 9 DI: Constructor Injection

V tomto příkladu má třída `ExampleClass` závislost na rozhraní `IDataRepository`. Tato závislost je předána třídě `ExampleClass` prostřednictvím konstruktoru. Kontejner IoC je zodpovědný za vytvoření instance `IDataRepository` a její předání do třídy `ExampleClass`, když je instance vytvořena.

Property Injection: Tento typ Dependency Injection zahrnuje předání požadovaných závislostí třídě prostřednictvím jejích vlastností. Kontejner IoC nastavuje hodnoty těchto vlastností při vytváření instance třídy. Tato metoda je méně běžně používaná než Constructor Injection, ale může být užitečná v určitých případech, například když lze změnit závislosti za běhu.

```
public class ExampleClass
{

    private IDataRepository ExampleRepository { get; set; }

    public void SaveData(Data data)
    {
        ExampleRepository.SaveData(data);
    }
}
```

Obrázek 10 DI: Property Injection

V tomto příkladu má třída `ExampleClass` závislost na rozhraní `IDataRepository`, které je třídě předáno prostřednictvím vlastnosti `ExampleRepository`. Kontejner IoC je zodpovědný za nastavení hodnoty této vlastnosti při vytváření instance třídy `ExampleClass`.

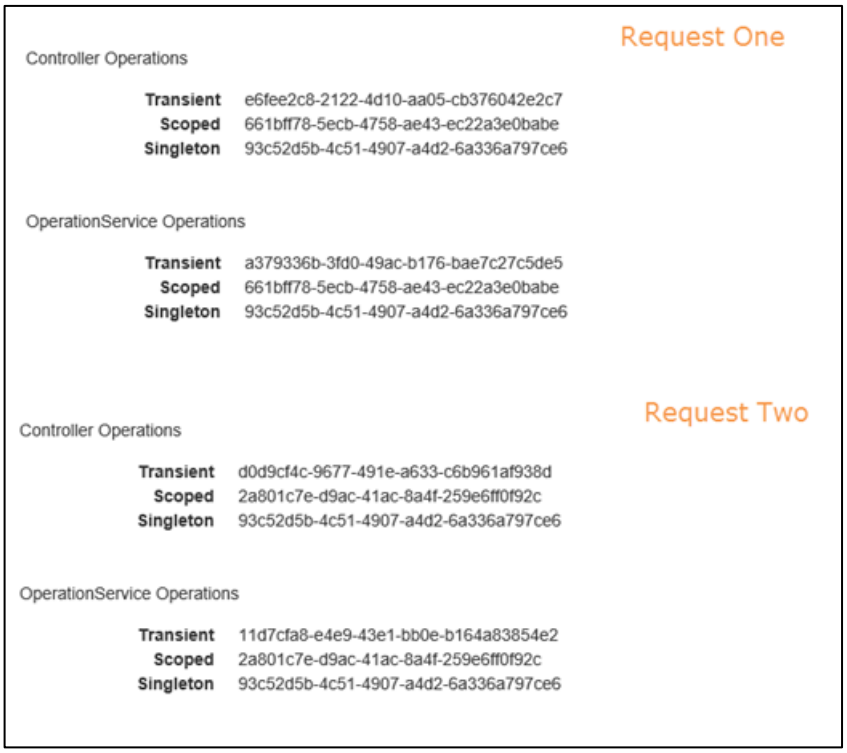
V obou těchto příkladech třída `ExampleClass` nemusí vytvářet vlastní instanci implementace `IDataRepository`. Díky tomu je třída mnohem jednodušší a lépe udržovatelná, protože odpovědnost za vytváření závislostí byla přesunuta do kontejneru IoC.

Další možné rozdělení ve WPF typů DI je podle životnosti objektů v kontejneru IoC:

Singleton: Objekt Singleton je jedinou instancí třídy, kterou sdílejí všechny části aplikace, které jsou na ní závislé. Kontejner IoC vytvoří jednu instanci objektu a vrátí stejnou instanci pokaždé, když si to závislý objekt vyžádá. Tím je zajištěno, že všechny části aplikace používají stejnou instanci objektu a že jakékoli změny stavu objektu se projeví ve všech částech aplikace.

Transient: Transient objekt je nová instance třídy, která je vytvořena pokaždé, když je požadována závislým objektem. Kontejner IoC vytvoří novou instanci objektu pro každý požadavek a instance nejsou sdíleny mezi objekty. To je užitečné při tvorbě nové instance objektu pro každou instanci závislého objektu nebo když je potřeba izolovat stav objektu od ostatních částí aplikace.

Scope: Scope definuje životnost objektu v kontejneru IoC. Scope určuje, kdy je objekt vytvořen, kdy je vyřazen a kdy je vhodný pro sběr odpadu. Scope je dále děleno na per-request a per-thread. Per-request scope vytvoří novou instanci objektu pro každý požadavek, zatímco per-thread scope vytvoří jednu instanci objektu, který je sdílen všemi částmi aplikace, které běží na stejném vláknu.



Obrázek 11 DI: Životnost objektů [24]

Na Obrázek 11 je příklad jednotlivých životností objektu a zobrazení ID jejich operace. Singleton je vždy stejný, protože vytvoří pouze jednu instanci pro celou aplikaci. Scope typu per-request vytvoří novou instanci vždy s dalším požadavkem, ale instance jsou sdíleny mezi objekty, stejný Scope je pro Controller Operation i pro OperationService Operations. A Transient vytváří novou instanci pro každý požadavek.

[9] [10] [25]

2.3 Commands

Commands, česky příkazy, jsou ve WPF způsob, jak implementovat běžné funkce opakovaně znovupoužitelným a flexibilním způsobem. WPF poskytuje podporu pro příkazy prostřednictvím interface (rozhraní) `System.Windows.Input.ICommand`. Rozhraní `ICommand` definuje dvě metody, `Execute` a `CanExecute` a událost `CanExecuteChanged`. Metoda `Execute` je volána při provádění příkazu a metoda `CanExecute` je volána, aby se zjistilo, zda lze příkaz provést v aktuálním stavu. Událost `CanExecuteChanged` je vyvolána, když se změní hodnota vrácená metodou `CanExecute`. [10] [26]

Příkazy, které trvají dlouho na zpracování, například výstup dat z databáze, by zastavily průběh celé aplikace a pro uživatele by UI působilo jako zaseknuté. Pro tyto příkazy je vhodné využít asynchronní zpracování.

```
public abstract class CommandBase : ICommand
{
    public event EventHandler? CanExecuteChanged;

    public virtual bool CanExecute(object? parameter)
    {
        return true;
    }

    public abstract void Execute(object? parameter);

    protected void OnCanExecutedChanged()
    {
        CanExecuteChanged?.Invoke(this, new EventArgs());
    }
}
```

Obrázek 12 Commands: Základní příkaz

První je na Obrázek 12 navrhnuta abstraktní třída `CommandBase`, která implementuje rozhraní `ICommand`. Funkce `CanExecute` je implementována a vrací vždy `true`, proto se třída `CommandBase` využívá pro synchronní příkazy. Zároveň je funkce virtuální, to znamená, že v odvozených třídách může být implementována znova.

Na Obrázek 13 je navržena abstraktní třída `AsyncCommandBase`, která implementuje `CommandBase`. Hlavní rozdíl mezi nimi je, že při volání funkce `Execute` se změní hodnota `IsExecuting` a tím i hodnota, kterou vrací funkce `CanExecute`. Zároveň je spuštěna asynchronně, díky čemuž UI aplikace nezamrzne, protože se příkaz začne provádět v jiném vláknu. V UI díky tomu můžeme provádět další činnosti, které nejsou přímo závislé na výsledku spuštěného příkazu. Zároveň lze v UI zobrazovat například točící se kolečko, které znamená, že příkaz běží, díky využití `CanExecute`. To je vhodné, jedná-li se o příkaz, který nezobrazuje žádný výstup na UI, například zaslání emailu po kliknutí na tlačítko odeslat může chvíli trvat, než se vše zpracuje a email se opravdu odešle. Nebo pokus o přihlášení uživatele, login a hash hesla se odešle do databáze a čeká se na odpověď, zda uživatel s tímto heslem existuje.

```
public abstract class AsyncCommandBase : CommandBase
{
    private bool _isExecuting;

    private bool IsExecuting
    {
        get
        {
            return _isExecuting;
        }
        set
        {
            _isExecuting = value;
            OnCanExecutedChanged();
        }
    }

    public override bool CanExecute(object? parameter)
    {
        return !IsExecuting && base.CanExecute(parameter);
    }

    public override async void Execute(object? parameter)
    {
        IsExecuting = true;

        try
        {
            await ExecuteAsync(parameter);
        }
        finally
        {
            IsExecuting = false;
        }
    }

    public abstract Task ExecuteAsync(object? parameter);
}
```

Obrázek 13 Commands: Asynchronní příkaz

2.4 Stores

K uchování hodnot, objektů nebo například právě přihlášeného uživatele se využívají tzv. stores, česky skladiště. Skladiště jsou nejčastěji instanciována jako Singleton (kapitola 2.222), tedy jsou společná pro celý běh aplikace.

Vhodné pro uchování i všech hodnot v aplikacích, ke kterým přistupuje a pracuje s nimi pouze jeden uživatel. Do skladišť jsou nahrány hodnoty pouze při spuštění aplikace z databáze a poté se pracuje pouze s nimi a k databázi se už nepřistupuje pro vypisování, pouze při zapsání nových hodnot.

Tento přístup ale není možný u aplikací, do kterých přistupuje současně více uživatelů a zobrazují, upravují a zapisují hodnoty. Proto je potřeba zachovat integritu dat. Integritou dat se myslí, kdy zobrazená data jsou totožná s daty, které jsou uložené v databází, a kdy během uložení dat nedojde k jejich neočekávaným změnám.

Jednoduchým a klasickým příkladem skladiště může být `MessageStore`, do kterého přistupuje celá aplikace a ukládá do něj a zobrazuje z něj zprávy ve všech pohledech, kde je to potřeba.

```
public class MessageStore
{
    private string _message;

    public string Message
    {
        get
        {
            return _message;
        }
        set
        {
            _message = value;
            OnMessageChanged();
        }
    }

    public bool HasMessage => Message.Length > 0;

    public event Action MessageChanged;

    private void OnMessageChanged()
    {
        MessageChanged?.Invoke();
    }
}
```

Obrázek 14 Stores: MessageStore

Příklad implementace `MessageStore` je výše na Obrázek 14. V aplikaci je `MessegeStore` instancializován pomocí DI jako Singleton a do ViewModelu se vkládá pomocí konstruktoru. Všechny ViewModely pak pracují se stejnou instancí a díky tomu, přes tuto třídu mohou ViewModely mezi sebou komunikovat a případně zobrazovat zprávy o událostech.

2.5 Services

Services, česky služby, jsou objekty nebo třídy, které poskytují funkcionalitu, kterou mohou požívat jiné objekty. Služby se často používají k zapouzdření složitého chování nebo funkcí v rámci aplikace, takže k nim lze přistupovat z jiných částí aplikace, aniž by bylo nutné odhalovat podrobnosti o tom, jak jsou implementovány.

Například služba, která zpracovává záznamy (logy) v aplikaci WPF. Zapouzdřením funkce vytváření záznamů v rámci služby je možné vystavit jednoduché a snadno použitelné rozhraní ostatním částem aplikace, přičemž detaily implementace zůstávají skryté. To usnadňuje údržbu a aktualizaci funkcí vytváření záznamů v budoucnu, aniž by to ovlivnilo části aplikace.

```
public class LoggingService
{

    public void Log(string message)
    {
        Console.WriteLine($"[LOG] {message}");
    }
}
```

Obrázek 15 Services: LoggingService

```
public class ExampleViewModel
{
    private readonly LoggingService _loggingService;

    public ExampleViewModel(LoggingService loggingService)
    {
        _loggingService = loggingService;
    }

    public void DoSomething()
    {
        // Log a message
        _loggingService.Log("Doing something...");

        // Do something else...
    }
}
```

Obrázek 16 Services: Příklad využití služby ve ViewModel

2.6 Navigation

Navigace v WPF se týká procesu přechodu z jednoho pohledu (view) do druhého v rámci aplikace. WPF poskytuje několik způsobů implementace navigace:

Frame Navigation: Ovládací prvek `Frame` ve WPF poskytuje navigační možnosti pro pohyb mezi různými pohledy (Views) v rámci aplikace. Cílové zobrazení lze určit pomocí identifikátoru URI nebo lze využít kód k programové navigaci do pohledu (view). Ovládací prvek `Frame` také poskytuje zásobník, který umožňuje přejít zpět na předchozí pohledy.

Page Navigation: WPF také poskytuje podporu pro navigaci založenou na stránce, kde je každé zobrazení reprezentováno objektem stránky. Ovládací prvek `Frame` lze použít k navigaci mezi stránkami a třída `Page` poskytuje vlastnost `NavigationService`, kterou lze použít k programové navigaci mezi stránkami.

NavigationWindow Navigation: Třída `NavigationWindow` ve WPF poskytuje jednoduchý způsob, jak vytvořit okno nejvyšší úrovně, které podporuje navigaci mezi zobrazeními. Třída `NavigationWindow` poskytuje metodu `Navigate`, kterou lze použít k navigaci do cílového zobrazení, a zásobník, který umožňuje přejít zpět do předchozích zobrazení.

Vlastní navigace: Kromě vestavěných možností navigace poskytovaných WPF lze také implementovat vlastní navigaci. Například implementovat vlastní navigační službu, která poskytuje možnost navigace mezi zobrazeními a která poskytuje další navigační funkce.

Navigace ve WPF pomocí vzoru MVVM zahrnuje navigaci mezi různými pohledy při zachování oddělení zájmů mezi pohledem (View) a ViewModelem. Proto je v této práci navrhnuta vlastní navigace. Navržená navigace využívá Stores, Action, Command, Services a DI.

Zjednodušeně aplikace vytvoří jednu instanci objektu skladiště (store) pro uložení současného pohledu, který se má zobrazovat v hlavním pohledu, který je pro celou aplikaci stejný. Dále pomocí příkazů (commands), které využívají služby (services) pro navigaci a akcí (actions), které upozorňují UI na změnu pohledu reagovat, je možné přecházet z jednoho pohledu do druhého. Skladiště, který ukládá současný pohled, může zároveň udržovat v zásobníku předchozí pohledy pro možný návrat zpět, pokud to aplikace vyžaduje.

Díky tomuto přístupu je možné zobrazovat i více pohledů zároveň, tzn. rozdělit si aplikaci na několik částí a zobrazovat v každé části jiný pohled, který je obstaráván svým

ViewModelem. To se můžu hodit pro panely s tlačítky nebo prvky, které se během celé aplikace nemění. Speciální navigací je modální navigace, která vytváří vizuálně okno v popředí před právě zobrazovaným pohledem, například pro registraci nebo přihlášení.

[11] [27]

2.7 Styles

Ve WPF se styles česku styly používají k nastavení konzistentního vzhledu a schování ovládacích prvků v uživatelském rozhraní. Styl je sbírka nastavení vlastností, které definují vizuální vzhled a chování ovládacího prvku.

```
<Window
    x:Class="HelloWordWPF.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:HelloWordWPF"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    Title="MainWindow"
    Width="800"
    Height="450"
    mc:Ignorable="d">
    <Window.Resources>
        <Style x:Key="BlackText" TargetType="TextBlock">
            <Setter Property="Background" Value="Black" />
            <Setter Property="Foreground" Value="White" />
            <Setter Property="FontSize" Value="20" />
            <Setter Property="FontWeight" Value="Bold" />
        </Style>
    </Window.Resources>

    <StackPanel VerticalAlignment="Center">
        <TextBlock
            HorizontalAlignment="Center"
            Style="{StaticResource BlackText}"
            Text="Hello Word in WPF with style!" />
        <TextBlock HorizontalAlignment="Center" Text="Hello Word in WPF without style!" />
    </StackPanel>
</Window>
```

Obrázek 17 Styles: XAML Hello World aplikace se styly

Na obrázku Obrázek 17 je ukázka již jednou použitého XAML kódu, kde je přidán `Window.Resources` se `Style` pro `TextBlock`. Tento vytvořený styl je použit pouze pro `TextBlock` s nastaveným Style s názvem `BlackText`. Kdyby styl neměl žádný `x:Key`, použil by se automaticky na všechny existující `TextBlock` v daném `Window`.

Vytvořený styl `BlackText` obsahuje elementy, které nastavují pozadí textu, barvu písma, velikost písma a tloušťku písma.

Obrázek 18 Styles: Vizualizace Hello World aplikace se styly

Ideální je tvořit styly v `ResourceDictionary` se všemi styly, které v aplikaci budou použity. Pak v `App.xaml` vytvořenou `ResourceDictionary` připojit. Díky tomu mají ke všem stylům přistup všechny pohledy. To se hodí, když by bylo potřeba například změnit barvu textu stylu `BlackText` na jinou, tak to stačí udělat v `ResourceDictionary`, místo toho, aby upravovaly všechny jednotlivé `Window.Resources` v pohledech.

Styly ve WPF nejsou užitečné pouze pro definování vizuálního vzhledu ovládacích prvků uživatelského rozhraní, ale také pro definování jejich chování a interakce.

Event triggers – Spouštěče událostí: Styly mohou definovat spouštěče událostí, které reagují na interakce uživatele s ovládacím prvkem. Například ovládacímu prvku `Button` lze přiřadit styl, který definuje spouštěč pro událost `MouseEnter`, která změní barvu pozadí tlačítka, když na něj uživatel najede myší.

Control templates – Šablony ovládacích prvků: Styly mohou definovat šablony ovládacích prvků, které zcela nahrazují vizuální vzhled a chování ovládacího prvku. Například ovládacímu prvku Posuvník lze přiřadit styl, který definuje vlastní šablonu ovládacího prvku, která změní vzhled posuvníku a přidá nové chování, jako je přichycení k diskrétním hodnotám.

Attach behaviors – Připojené chování: Styly mohou definovat připojené chování, které rozšiřuje chování ovládacího prvku. Například ovládacímu prvku `TextBox` lze přiřadit styl, který definuje připojené chování pro manipulaci s klávesou Enter, která přesune `focus` na další ovládací prvek v pořadí karet.

Animation – Animace: Styly mohou definovat animace, které přidávají zajímavou vizualizaci a zpětnou vazbu k uživatelským interakcím. Například ovládacímu prvku `Button` lze přiřadit styl, který definuje animaci pro událost `Click`, která tlačítku přidá krátký barevný záblesk, který indikuje, že na něj bylo klepnuto.

[10] [15] [28]

Použitím stylů k definování chování a interakce ve WPF lze vytvářet bohatá, interaktivní uživatelská rozhraní, která reagují na akce uživatele intuitivním a poutavým způsobem.

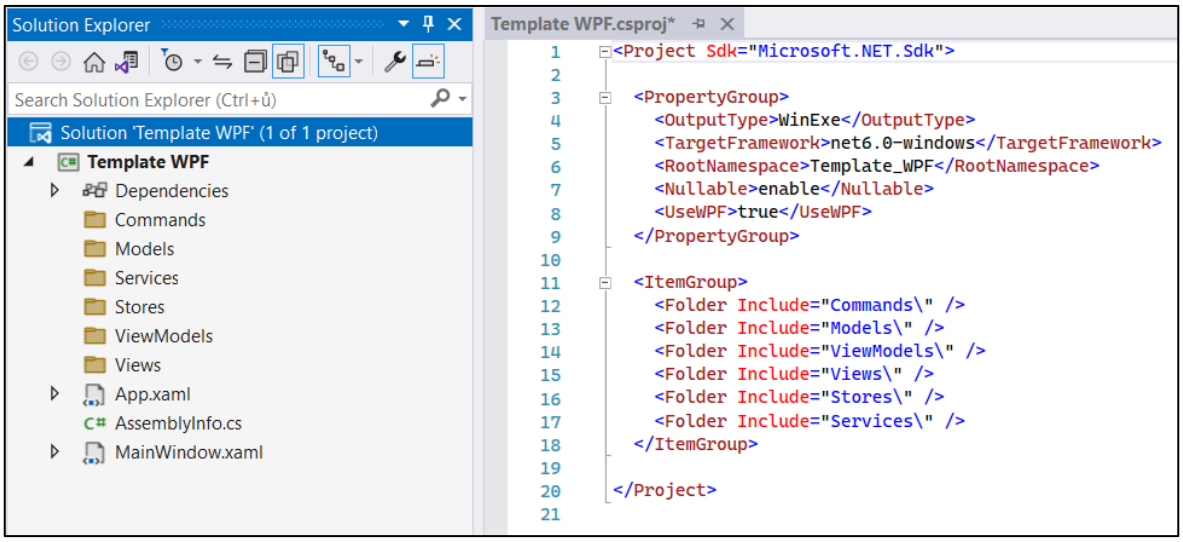
Funkční aplikaci lze vytvořit i úplně bez stylů pouze se základním předdefinovaným vzhledem. Aplikace by takto mohla fungovat dle požadavků zákazníka, ale zákazník vždy jako první vidí právě vzhled a to, jak s ním aplikace interaguje a první pohled může rozhodnout o tom, jestli zákazník o aplikaci zájem projeví nebo ne, přestože by aplikace měla všechny funkce dle jeho představ, proto je vzhled aplikace často velmi důležitý.

3 ŠABLONA WPF APLIKACE

Aplikace WPF pomocí vzoru MVVM obsahují vždy základní podobné vlastnosti. Všechny tyto vlastnosti byly popsány v kapitole 2. Šablona je všechny sjednotí a vytvoří základní aplikaci, která sice nebude dělat nic praktického, ale bude připravena k dalšímu vývoji a bude obsahovat všechny základní důležité věci jako je DataBinding 2.1, Dependency Injection 2.2, příkazy 2.3, skladiště 2.4, služby 2.5 a vlastní navigaci mezi pohledy 2.6.

3.1 Projekt ve Visual Studio

V prvním kroku vytvoříme projekt ve Visual Studiu, takže nejprve si vytvoříme nový projekt WPF Application s jazykem C#, pojmenujeme ho Template WPF a využijeme nejnovější Framework, který je v současnosti .NET 6.0. Tím jsme vytvořili prázdný projekt, do kterého jsou přidány složky pro rozdělení jednotlivých tříd pro lepší organizaci.



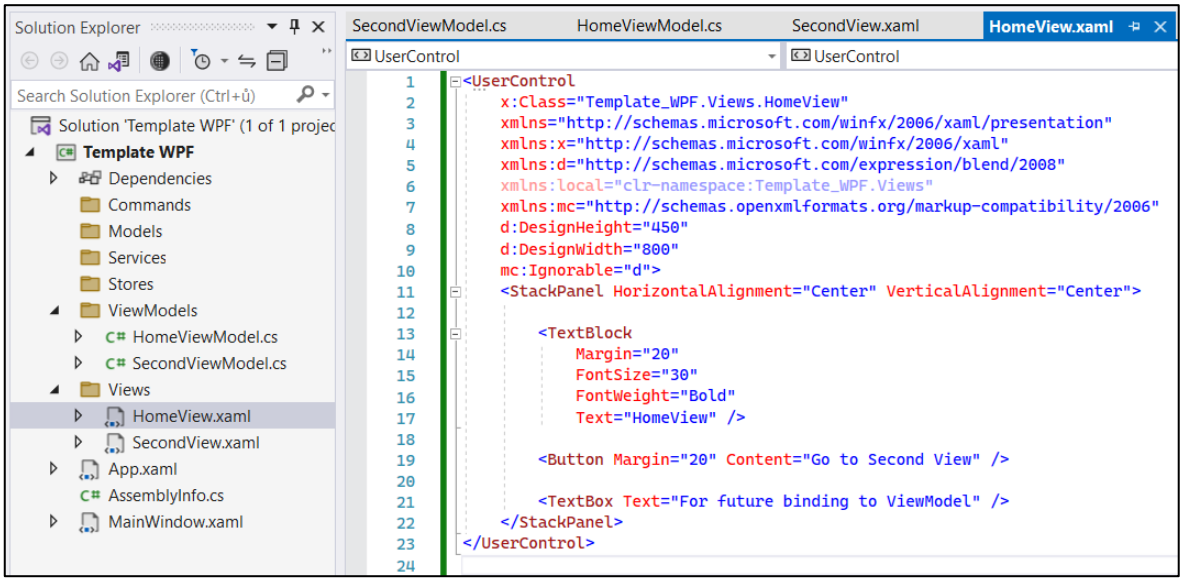
Obrázek 19 Šablona WPF: Vytvoření projektu

3.2 Views a ViewModels

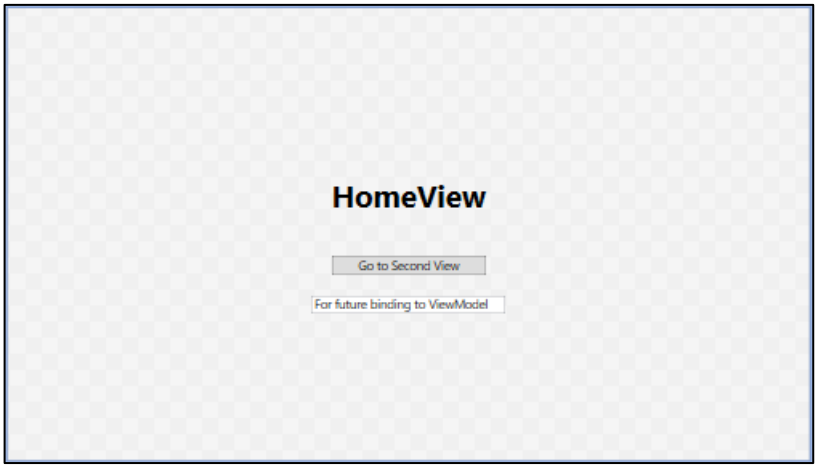
V druhém kroku vytvoříme, pro názornost navigace mezi pohledy, dva Views a k nim odpovídající ViewModels, kde každý bude obsahovat `TextBox`, díky kterému poznáme, ve kterém se právě nacházíme.

Views (pohledy) vytváříme jako `User Control`. `Grid` změníme na `StackPanel`, aby se přidané elementy „stackovaly“, přidávaly za sebou a nemuseli jsme navíc řešit jejich

pozicování vůči sobě. Jako elementy přidáme `TextBlock`, který zobrazuje informaci o tom, ve kterém pohledu se nacházíme. `Button` tlačítko, které zatím nemá žádnou funkci, ale později bude navigovat, přecházet z jednoho pohledu do druhého. `TextBox`, který později bude propojen s ViewModelem a bude zapsaný text ukládat do skladiště, s kterým bude například pracovat druhý ViewModel a zobrazovat ho. ViewModels jsou prozatím vytvořené pouze jako prázdné `public class`.



Obrázek 20 Šablona WPF: Přidání Views a ViewModels



Obrázek 21 Šablona WPF: Náhled HomeView

Při spuštění aplikace v současném stavu, aplikace nezobrazuje žádný vytvořený pohled (View), pouze prázdné okno. Je to způsobeno tím, že `App.xaml` používá `StartupUri` jako `MainWindow.xaml` a ten se od vytvoření projektu nezměnil a obsahuje pouze prázdné `Window`

s `Grid`. S tímto bude aplikace pracovat i nadále, celou dobu bude primárně zobrazovat `MainWindow.xaml` a ten bude zobrazovat právě vybraný View. Aby aplikace při spuštění alespoň něco zobrazovala, přidáme do `<view:HomeView />`, tím se bude zobrazovat vybraný pohled.

3.3 Stores, Commands a Services

Ve třetím kroku přidáme Stores skladiště pro navigaci a zprávy, Commands příkazy pro navigaci, Services služby pro navigaci a ViewModel, z kterého budou všechny ostatní ViewModely dědit. `ViewModelBase` dědí z `INotifyPropertyChanged` a obsahuje funkci `OnPropertyChanged` a událost `PropertyChanged`, na které chceme mít přístup ze všech ViewModelů, když chceme informovat View o změně dat ve ViewModelu.

Nejprve vytvoříme Stores skladiště `NavigationStore` pro uchování současného pohledu, který později `MainWindow` bude zobrazovat.

```
public class NavigationStore
{
    private ViewModelBase _currentViewModel;

    public ViewModelBase CurrentViewModel
    {
        get { return _currentViewModel; }
        set
        {
            _currentViewModel?.Dispose();
            _currentViewModel = value;
            OnCurrentViewModelChanged();
        }
    }

    public event Action CurrentViewModelChanged;

    private void OnCurrentViewModelChanged()
    {
        CurrentViewModelChanged?.Invoke();
    }
}
```

Obrázek 22 Šablona WPF: NavigationStore

`NavigationStore` udržuje hodnotu `ViewModelBase`, tedy o pohledu, který chceme právě zobrazovat. Při změně pohledu se nejprve zjistí, zda nějaký pohled obsahuje a ten se nejprve uvolní díky `Dispose()`, poté se zapíše nová hodnota a zavolá se funkce `OnCurrentViewModelChange`, která vyvolá událost `CurrentViewModelChanged`, aby se upozornili, části kódu, které danou událost poslouchají.

Services služby obsahují nově dvě třídy, `INavigationService`, což je interface s pouze jednou funkcí `Navigate`, kterou implementujeme v `NavigationService`. `NavigationService` využívá již vytvořeného skladiště `NavigationStore` a funkci `Func<TViewModel>`, což je typ delágata, který vrací proměnnou typu `ViewModelBase`.

```
public class NavigationService<TViewModel> : INavigationService where TViewModel : ViewModelBase
{
    private readonly NavigationStore _navigationStore;
    private readonly Func<TViewModel> _createViewModel;
    0 references
    public NavigationService(NavigationStore navigationStore, Func<TViewModel> createViewModel)
    {
        _navigationStore = navigationStore;
        _createViewModel = createViewModel;
    }
    3 references
    public void Navigate()
    {
        _navigationStore.CurrentViewModel = _createViewModel();
    }
}
```

Obrázek 23 Šablona WPF: NavigationService

Poté můžeme vytvořit Commands příkazy, první bude `CommandBase`, který dědí z `ICommand` a již byl popsán v kapitole 2.3. Druhý příkaz je pro navigaci `NavigateCommand`, dědí z vytvořeného `CommandBase`, využívá `INavigateService` a ve funkci `Execute` jen jednoduše nad `NavigationService` spustí funkci `Navigate()`.

```
public class NavigateCommand : CommandBase
{
    private readonly INavigationService _navigationService;
    0 references
    public NavigateCommand(INavigationService navigationService)
    {
        _navigationService = navigationService;
    }
    1 reference
    public override void Execute(object? parameter)
    {
        _navigationService.Navigate();
    }
}
```

Obrázek 24 Šablona WPF: NavigateCommand

3.4 Dependency Injection

V tento moment je v aplikaci funkční navigace, ale zatím není nikde aplikovaná a `MainWindow` s ní nepočítá, proto musíme změnit `MainWindow` a navíc ještě v `App.xaml.cs` zavést DI pro zavedení závislostí a inicializaci všech vytvořených tříd.

```
<Grid>
    <Grid.Resources>
        <DataTemplate DataType="{x:Type viewmodels:HomeViewModel}">
            <view:HomeView />
        </DataTemplate>

        <DataTemplate DataType="{x:Type viewmodels:SecondViewModel}">
            <view:SecondView />
        </DataTemplate>
    </Grid.Resources>
    <ContentControl Content="{Binding CurrentViewModel}" />
</Grid>
```

Obrázek 25 Šablona WPF: MainWindow po zavedení navigace

`MainWindow` teď zobrazuje `CurrentViewModel`, který jsme vytvořili ve skladišti `NavigationStore` a poprvé využíváme i DataBinding. Aby DataBinding fungovalo, potřebujeme i zdrojový objekt a ten zatím nikde nemáme. Proto vytvoříme `MainViewModel`, který bude obsahovat `NavigationStore` a bude poslouchat změny události `OnCurrentViewModelChanged`, aby když se změní pohled ve skladišti je `MainViewModel` zaregistroval a promítnul do `MainWindow`.

```
public  class MainViewModel : ViewModelBase
{
    private readonly NavigationStore _navigationStore;
    1 reference
    public ViewModelBase CurrentViewModel => _navigationStore.CurrentViewModel;
    0 references
    public MainViewModel(NavigationStore navigationStore)
    {
        _navigationStore = navigationStore;

        _navigationStore.CurrentViewModelChanged += OnCurrentViewModelChanged;
    }
    2 references
    private void OnCurrentViewModelChanged()
    {
        OnPropertyChanged(nameof(CurrentViewModel));
    }
    3 references
    public override void Dispose()
    {
        _navigationStore.CurrentViewModelChanged -= OnCurrentViewModelChanged;
        base.Dispose();
    }
}
```

Obrázek 26 Šablona WPF: MainViewModel

Nakonec zavedení DI do aplikace. DI se zavádí při spouštění aplikace, proto musíme přepsat to, jakým způsobem se aplikace spouští. Nejprve smažeme z `App.xaml` `StartupUri="MainWindow.xaml"`, protože spuštění `MainWindow` vytvoříme přímo v `App.xaml.cs`.

`App.xaml.cs` obsahuje třídu `App`, která spouští celou aplikaci. Vytvoříme konstruktor `App()`, přepíšeme funkci `OnStartup` a přidáme proměnou `IServiceProvider`, která se bude starat o DI. Abychom `IServiceProvider` mohli přidat, musíme do projektu nejprve nainstalovat balíček `Microsoft.Extensions.DependencyInjection`.

```
public partial class App : Application
{
    private readonly IServiceProvider _servicesProvider;

    public App()
    {
        IServiceCollection services = new ServiceCollection();


        services.AddSingleton<NavigationStore>();

        services.AddTransient<HomeViewModel>();
        services.AddTransient<SecondViewModel>();
        services.AddSingleton<MainViewModel>();

        services.AddSingleton<MainWindow>(s => new MainWindow()
        {
            DataContext = s.GetRequiredService<MainViewModel>()
        });

        services.AddSingleton<INavigationService>(CreateHomeNavigationService);

        _servicesProvider = services.BuildServiceProvider();
    }

    protected override void OnStartup(StartupEventArgs e)
    {

        INavigationService initialNavigationService = _servicesProvider.GetRequiredService<INavigationService>();
        initialNavigationService.Navigate();

        MainWindow = _servicesProvider.GetRequiredService<MainWindow>();
        MainWindow.Show();

        base.OnStartup(e);
    }

    private INavigationService CreateHomeNavigationService(IServiceProvider serviceProvider)
    {
        return new NavigationService<HomeViewModel>(
            serviceProvider.GetRequiredService<NavigationStore>(),
            () => serviceProvider.GetRequiredService<HomeViewModel>());
    }
}
```

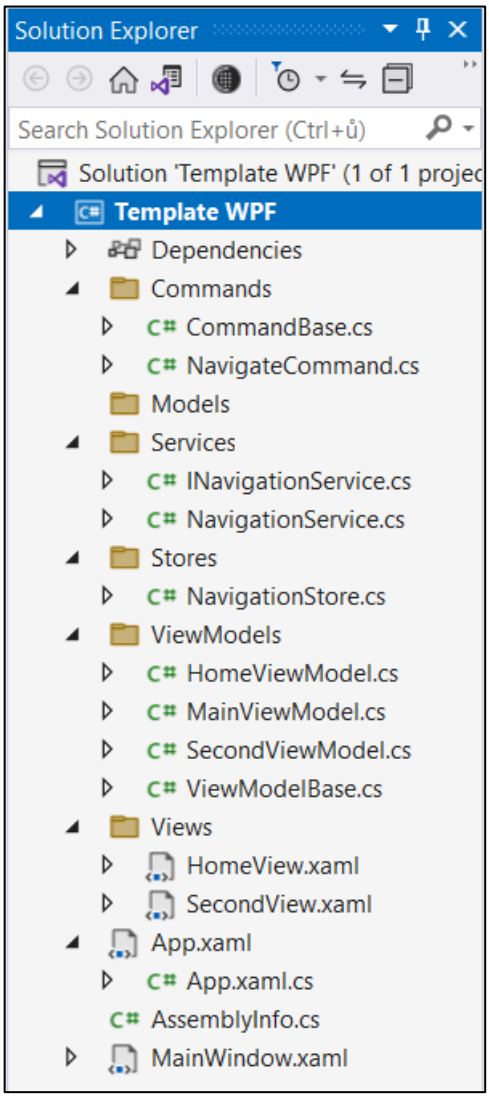
Obrázek 27 Šablona WPF: App.xaml.cs

Vytvoříme nový `ServiceCollection`, do kterého postupně přidáme všechny závislosti. Funkce `OnStartup` spustí a zobrazí okno aplikace, zároveň v ní proběhne první navigace, a to na `initialNavigationService`, který je vytvořen také pomocí DI.

Aplikace pak po spuštění zobrazí pohled `HomeView`, ale tlačítka a navigace v něm dál pořád nefungují, protože je potřeba ještě implementovat ViewModely, které se o to postarají.

3.5 MessageStore a implementace ViewModels

Projekt by v tento moment měl obsahovat tyto složky a soubory.



Obrázek 28 Šablona WPF: Projekt před implementací ViewModelů a zpráv

Aby mezi sebou pohledy mohly komunikovat a předávat si určitá data, přidáme do projektu další Store skladiště pro zprávy `MessageStore`. `MessageStore` je velice podobný `NavigationStore`, ale místo toho, aby uchovával hodnotu o `ViewModelBase`, uchovává `string`, tedy řetězec textu, který si budou mezi sebou pohledy předávat. Po vytvoření musí být přidán i do DI.

```
public class MessageStore
{
    private string _message;

    public string Message
    {
        get { return _message; }
        set
        {
            _message = value;
            OnMessageChanged();
        }
    }
    public event Action MessageChanged;

    private void OnMessageChanged()
    {
        MessageChanged?.Invoke();
    }
}
```

Obrázek 29 Šablona WPF: MessageStore

Implementace ViewModelů je obdobná v `HomeViewModel` i v `SecondViewModel`, protože mají velice podobné chování. Od obou požadujeme, aby tlačítko navigovalo do druhého pohledu a zprávu zároveň zobrazovalo a ukládalo do `MessageStore`. Díky DI jsou do nich v konstruktoru předány objekty navigace na požadovaný pohled a skladiště zpráv. Proto je ve ViewModelu tlačítko, které využívá již vytvořený `NavigateCommand` se službou, která naviguje do druhého modelu. ViewModel je přihlášen k události z `MessageStore`, aby byl upozorněn, pokud by se zpráva změnila. Zároveň pokud se z ViewModelu přechází do jiného ViewModelu, tak je zavolána funkce `Dispose`, která se stará primárně o to, aby byl ViewModel správně uvolněn z paměti a v tomto případě i přestal poslouchat události z `MessageStore`. Kdyby ViewModel nepřestal poslouchat `MessageStore` při jeho odchodu, nikdy by nebyl správně uvolněn a zůstával by v paměti.

Vytvořený příkaz a řetězec se nakonec díky DataBinding zobrazí ve View.

```
public class HomeViewModel : ViewModelBase
{
    private readonly MessageStore _messageStore;

    public ICommand NavigateSecondViewCommand { get; }

    public HomeViewModel(INavigationService navigateSecondViewService, MessageStore messageStore)
    {
        NavigateSecondViewCommand = new NavigateCommand(navigateSecondViewService);
        _messageStore = messageStore;
        _messageStore.MessageChanged += OnMessageChanged;
        Message = messageStore.Message;
    }
    private string _message;

    public string Message
    {
        get { return _message; }
        set
        {
            _message = value;
            _messageStore.Message = _message;
            OnPropertyChanged(nameof(Message));
        }
    }

    private void OnMessageChanged()
    {
        OnPropertyChanged(nameof(Message));
    }

    public override void Dispose()
    {
        _messageStore.MessageChanged -= OnMessageChanged;
        base.Dispose();
    }
}
```

Obrázek 30 Šablona WPF: ViewModel implementace

```
<UserControl
    x:Class="Template_WPF.Views.HomeView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:Template_WPF.Views"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    d:DesignHeight="450"
    d:DesignWidth="800"
    mc:Ignorable="d">
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

        <TextBlock
            Margin="20"
            FontSize="30"
            FontWeight="Bold"
            Text="HomeView" />

        <Button
            Margin="20"
            Command="{Binding NavigateSecondViewCommand}"
            Content="Go to Second View" />

        <TextBox Text="{Binding Message, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
    </StackPanel>
</UserControl>
```

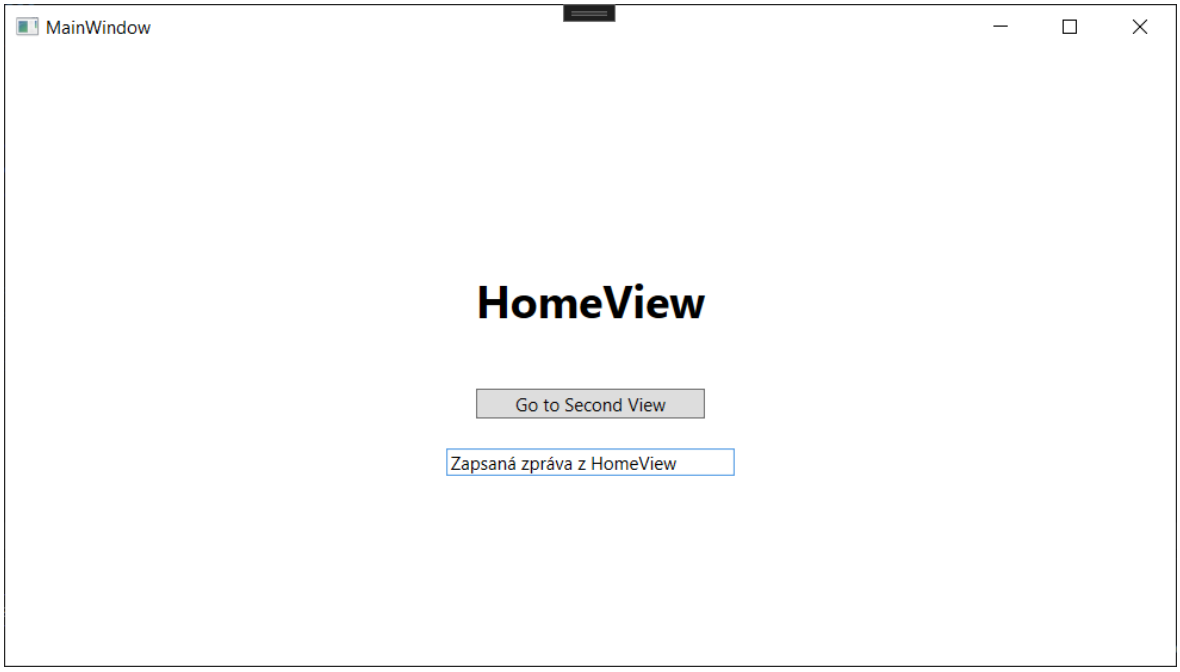
Obrázek 31 Šablona WPF: View implementace DataBinding

Aby vše fungovalo a všechny potřebné objekty dostávala aplikace z DI, je potřeba ještě upravit `App.xaml.cs`, kde vytvoříme funkce, které tvoří navigaci do našich ViewModelů a funkce, které tvoří samotné ViewModely. Pokud by některý z ViewModelů v budoucnu potřeboval další objekt, stačí jen upravit funkci, která vytváří ViewModel.

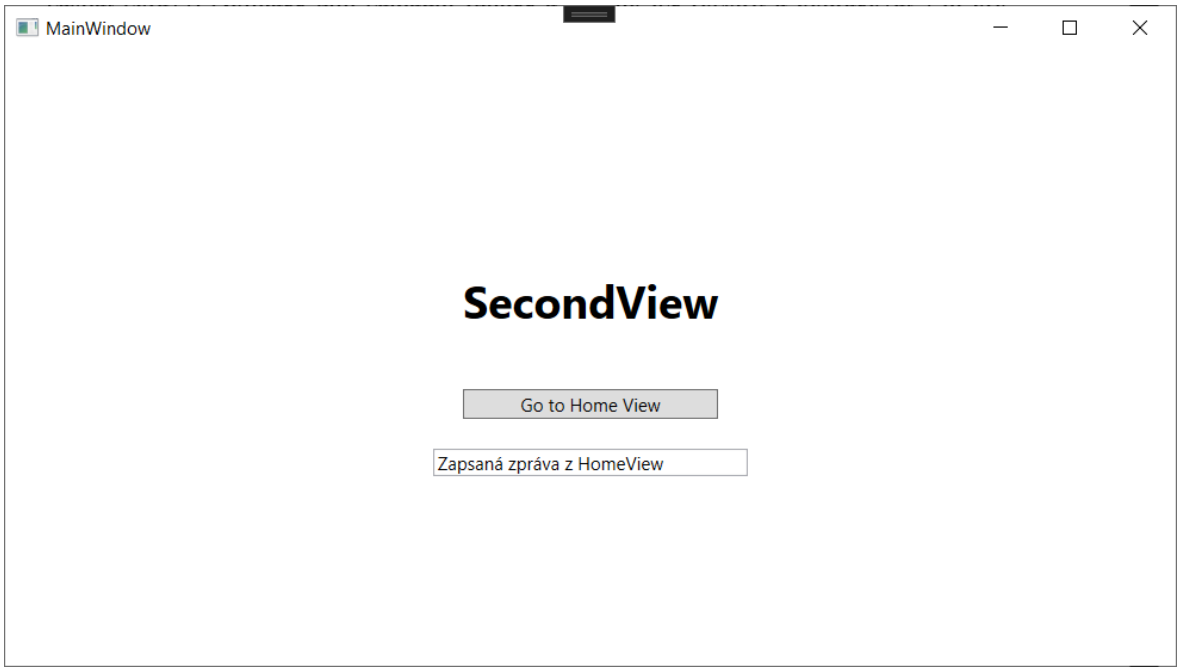
```
public partial class App : Application
{
    private readonly IServiceProvider _servicesProvider;

    public App()
    {
        IServiceCollection services = new ServiceCollection();

        services.AddSingleton<NavigationStore>();
        services.AddSingleton<MessageStore>();

        services.AddTransient<HomeViewModel>(CreateHomeViewModel);
        services.AddTransient<SecondViewModel>(CreateSecondViewModel);
        services.AddSingleton<MainViewModel>();

        services.AddSingleton<MainWindow>(s => new MainWindow()
        {
            DataContext = s.GetRequiredService<MainViewModel>()
        });

        services.AddSingleton<INavigationService>(CreateHomeNavigationService);

        _servicesProvider = services.BuildServiceProvider();
    }

    protected override void OnStartup(StartupEventArgs e)
    {

        INavigationService initialNavigationService = _servicesProvider.GetRequiredService<INavigationService>();
        initialNavigationService.Navigate();

        MainWindow = _servicesProvider.GetRequiredService<MainWindow>();
        MainWindow.Show();

        base.OnStartup(e);
    }

    private INavigationService CreateHomeNavigationService(IServiceProvider serviceProvider)
    {
        return new NavigationService<HomeViewModel>(
            serviceProvider.GetRequiredService<NavigationStore>(),
            () => serviceProvider.GetRequiredService<HomeViewModel>());
    }

    private INavigationService CreateSecondNavigationService(IServiceProvider serviceProvider)
    {
        return new NavigationService<SecondViewModel>(
            serviceProvider.GetRequiredService<NavigationStore>(),
            () => serviceProvider.GetRequiredService<SecondViewModel>());
    }

    private HomeViewModel CreateHomeViewModel(IServiceProvider serviceProvider)
    {
        return new HomeViewModel(
            CreateSecondNavigationService(serviceProvider),
            serviceProvider.GetRequiredService<MessageStore>());
    }

    private SecondViewModel CreateSecondViewModel(IServiceProvider serviceProvider)
    {
        return new SecondViewModel(
            CreateHomeNavigationService(serviceProvider),
            serviceProvider.GetRequiredService<MessageStore>());
    }
}
```

Obrázek 32 Šablona WPF: Finální App.xaml.cs

V tento moment je v aplikaci funkční navigace mezi dvěma pohledy, díky příkazům, službám a skladištím. Zároveň díky skladišti `MessageStore` si pohledy mezi sebou mohou zasílat zprávy. Aplikace umí základní funkce a snadno lze rozšířit a pokračovat v ní pro vytvoření reálných funkčností.



Obrázek 33 Šablona WPF: Pohled HomeView



Obrázek 34 Šablona WPF: Pohled SecondView

Po spuštění aplikace se zobrazí pohled HomeView, kde zapíšeme zprávu „Zapsaná zpráva z HomeView“ a poté pomocí tlačítka „Go to Second View“ přejdeme do druhého Pohledu SecondView. Oba pohledy mají vlastní ViewModel, který pracuje s navigací, aby se mezi nimi dalo přecházet a zároveň pracují i s `MessageStore`, pomocí kterého se zpráva zapsaná v jednom pohledu dostane do druhé.

3.6 Publikování a Git

Jakmile je aplikace vytvořená, je potřeba ji publikovat. To lze ve WPF velmi jednoduše. Celý projekt sestavíme v režimu Release, tím se vytvoří sada souborů, které lze publikovat. Publikujeme pomocí ClickOnce, kde se nastaví složka, kde se vytvoří soubory ke spuštění aplikace, způsob, jak se aplikace bude aktualizovat, jakou verzi právě publikujeme. Dále se nastavuje režim nasazení, kde lze vybrat mezi Framework-dependent a Self-contained. Framework-dependent předpokládá, že počítač, na kterém se aplikace bude spouštět, má nainstalované běhové prostředí (runtime) .NET s požadovanou verzí, kterou je aplikace vytvořená. Self-contained běhové prostředí přidá do aplikace, takže lze spustit i na počítačích, kde běhové prostření .NET není nainstalované, ale aplikace se kvůli tomu výrazně zvětší. Konkrétně v tomto případě aplikace publikovaná i s běhovým prostředím .NET má velikost 150,6 MB a bez běhového prostředí .NET má velikost 291 KB.

Publikovanou aplikaci můžeme dále sdílet a poběží i dalším uživatelům, kteří nemají Visual Studio.

[10] [29]

Ve Visual Studiu můžeme projekt i průběžně sdílet a zálohovat pomocí Git například na GitHub. GitHub je webová platforma, která vývojářům a organizacím poskytuje možnosti, jak nakládat s kódy aplikací, hlavními výhodami je správa verzí a spolupráce s ostatními. Tato aplikace pro Template WPF je nahrána na GitHub jako kód i jako publikovaná aplikace.

https://github.com/tsev26/Template-WPF

II. PRAKTICKÁ ČÁST

4 APLIKACE NA DOCHÁZKU

Docházkový systém nebo aplikace na docházku je nástroj, který organizaci dovoluje sledovat docházku zaměstnanců, jejich odpracované hodiny a absence. Výhodami pro organizaci může být organizace času svých zaměstnanců a plánování plánů a směn. Správa mezd díky docházkovému systému organizace může přesně vypočítat mzdy zaměstnanců a benefity na základě odpracovaných hodin. Docházkový systém také může poskytnout informace o výkonu zaměstnanců, díky informacím o nepřítomnosti a nedochvilnosti.

Existuje několik typů docházkový systémů od historických manuálních, kde zaměstnanci cvakali své karty, což ovšem bylo pomalé a velice složité na zpracování, až po současné, které můžou ověřovat uživatele pomocí biometriky, RFID čipů / karet nebo pomocí mobilních telefonů NFC.

Docházkové systémy se používají v různých organizacích, v kancelářích, ve školách a na univerzitách, ve výrobě a ve skladech, ve zdravotnictví, v obchodech a pohostinství. Všude, kde je nutné sledovat docházku zaměstnanců nebo studentů a spravovat pracovní plány ke zlepšení řízení pracovního výkonu.

4.1 Existující řešení

V současnosti existuje již spousta funkčních aplikací na správu docházky. Často jsou součástí velkého komplexní systému pro správu lidských zdrojů celé firmy, ať už se jedná o docházkový systém, plánování směn, evidence majetku, mzdy, nábor nových zaměstnanců. Nebo nabízejí přímo vlastní hardware s aplikací. Od toho se taky odvíjí jejich cena, která často není přímo daná a může se odvíjet podle počtu zaměstnanců, a právě podle toho, jak moc komplexní systém je zvolen a který hardware je vybrán. Platby jsou většinou na měsíční bázi, protože systémy využívají cloudové služby pro ukládání dat.

Příklady existujících docházkových systémů:

Docházkový systém Alveno

- https://www.alveno.cz/dochazkovy-system
- Webová aplikace
- Nabízí: Biometrické čtečky
- Dále nabízí: HR systém
- Využívá: Heureka, Flixbus, Zasilkovna, Sruvio

- Cena měsíce podle počtu zaměstnanců
- Webová aplikace vypadá dobře, má spoustu zajímavých funkcí jako plánování nepravidelných směn, nerovnoměrně rozvržená pracovní doba, nastavení výjimek, žádanky o dovolenou, schvalování, potvrzení GPS lokace

Docházkový systém Vema

- https://www.vema.cz/cs/dochazkovy-system
- Webová aplikace
- Nabízí: Displej + čtečka čipových karet, klíčenek a otisků prstů
- Dále nabízí: HR systémy, ekonomické systémy
- Využívá: především několik měst a nemocnic
- Cena na poptávku
- Webová aplikace vypadá dobře, podporuje schvalování a úpravy docházky

Docházkový systém ACS-line

- https://www.acsline.cz/cs/dochazkovy-system
- Webová aplikace
- Nabízí: Docházka MINI - ČIPY DALLAS, Docházka MINI - BEZKONTAKTNÍ ČIPY, Docházka MINI - OTISKY PRSTŮ, Docházka MINI - bezkontaktní karty MIFARE
- Dále nabízí: evidence návštěv, stravovací systém, evidence výroby, hotelový systém, řízení budov, personalistika
- Součástí docházkového systému jsou funkce pro přístupový systém, který je pro zamezení vstupu neoprávněným osobám
- Využívá: například Greiner packaging, Hirschmann Czech, ZAPA beton
- Cena na poptávku
- Webová aplikace vypadá uživatelsky přívětivě, nabízí různé model pracovní doby pro pracovníky, možnosti schvalování, evidenci návštěv, plánování absencí

Docházkový a přístupový systém AKTION CLOUD

- https://www.aktion.cz/produkty/dochazkovy-system.html
- Hardware + Online aplikace
- Nabízí: bezkontaktní snímač karet / čipů DOCHÁZKA, biometrický snímač otisků prstů DOCHÁZKA, bezkontaktní snímač karet / čipů OTEVŘENÍ, biometrický snímač otisků prstů OTEVŘENÍ a WEB, TABLET A MOBIL
- Dále nabízí: řízení přístupů, objednávání a výdej stravy, evidence a plánování návštěv, řízení zakázek, zápůjčky předmět
- Využívá: NOTINO, HOKAMI
- Cena je podle vybraného zařízení a podle počtu zaměstnanců
- Snímače vypadají velice profesionálně, nabízí žádosti a schvalování dovolených, vícesměnný provoz, exporty dat, pracovní cesty, přehled průchody na jednotlivých snímačích

Docházkový systém id partner

- https://www.idpartner.cz/dochazkovy-system/
- Hardware + Online aplikace
- Nabízí: Docházkový systém Profi a Docházkový systém Lite
- Dále nabízí: zabezpečení škol – čipový přístupový systém, přístupový systém pro bytové domy, automatické systémy pro otevírání závor, bran, garáží, návštěvní systém, pochůzkový systém, stravovací a objednávkový systém, výrobní logstika – sledování zakázek, domovní telefony
- Využívá: přes 2700 spokojených klientů (bez zmínky konkrétního)
- Cena na poptávku
- Docházka Lite obsahuje snímač, karty nebo klíčenky a systém umožňuje všechny funkce docházky

Docházkové hodiny vistech

- https://www.dochazkove-hodiny.cz/
- Hardware + aplikace
- Nabízí: Docházkové hodiny SEIKO Z120, Docházkový systém Time BOX X1, Docházkový systém Time BOX X4, Docházkový systém Time BOX X4 BIO, Docházkový systém Time BOX X4 WiFi, Docházkový systém Time BOX X4 BIO WiFi
- Využívá: bez zmíněné reference
- Cena pouze za snímač s aplikací bez dalších měsíčních poplatků
- Aplikace na snímačích vypadá snadno použitelná, má základní nastavení

Docházkový systém Fingera

- https://www.fingera.com/cs/dochazkovy-system-fingera/
- Hardware + aplikace + webová aplikace
- Nabízí: Tvářový terminál, FingeraGo, Web terminál, Přístupový terminál, Docházkový terminál do interiéru, Přístupový terminál II, Docházkový terminál do exteriéru, Přístupový turniket, Závora
- Dále nabízí: přístupový systém, stravovací systém
- Využívá: Košík.cz, Ariete, SKANSKA, SenergoS
- Cena docházkové systému měsíčně podle počtu uživatelů
- Aplikace i hardware vypadají dobře, nabízí i mobilní aplikaci

Všechny zmíněné docházkové systémy mají své výhody a nevýhody. Některé vyžadují zakoupení jejich hardwaru, které pak většinou slouží jako displej i jako čtečka karet a čipů nebo využívá biometriku jako otisk prstů. Většina z nich jsou založeny na měsíční platbě, a to podle počtu uživatelů. Pořizovací cena a měsíční platby patří mezi nevýhody. Na druhou stranu nabízí plně funkční řešení, které již bylo otestováno a určitě několikrát aktualizováno, tak aby perfektně fungovalo.

V této práci bude navržena aplikace na docházku, která poběží na počítači, takže k ní nebude třeba žádný dodatečný hardware ke spuštění a běhu aplikace. Jediný další hardware by byl nutný, pokud by se uživatelé autorizovali jinak, než pomocí klávesnice (hesla, PINu), například pomocí RFID čipu nebo telefonu. Pokud bude přístup k docházkové systému z více míst je nutné, aby byl připojen k síti nebo sdílenému uložišti, kde se data budou ukládat do databázového souboru SQLite.

Výhodou této aplikace oproti zmíněným docházkovým aplikacím je její bezplatná dostupnost, kde pro její provoz není potřeba koupě žádného speciálního hardwaru, stačí nějaký „starší“ počítač. Pro její funkčnost není ani potřeba, aby byla připojena k internetu, data se ukládají do souboru, který může být uložen na lokální síti, při nutnosti přístupu z více míst. Pro firmy by tato aplikace mohla být zajímavá jako první docházkový systém po přechodu z ručního počítání docházky. Pokud by pro ně byla nedostatečná, tak by se aplikace mohla rozšířit nebo by mohli přejít na jedno ze zmíněných placených řešeních.

5 PLÁNOVÁNÍ A CÍLE

Plánování je zásadní při vývoji softwaru, protože pomáhá zajistit, aby byl projekt dokončen včas, v rámci rozpočtu a v požadované kvalitě. Bez řádného plánování mohou projekty vývoje softwaru snadno sejít z cesty, což má za následek zpoždění, překročení nákladů a nižší kvalitu výsledků. Cíle plánování vývoje softwaru:

Definování rozsahu projektu: Plánování pomáhá definovat rozsah projektu, který zahrnuje vlastnosti a funkčnost softwaru, jakož i časovou osu a rozpočet projektu. To pomáhá zajistit, aby každý zapojený do projektu chápal, co se od něj očekává a co projekt přinese.

Identifikace požadavků: Plánování také pomáhá identifikovat požadavky softwaru, včetně potřeb uživatelů, technických požadavků a obchodních cílů. To pomáhá zajistit, že software splňuje potřeby zamýšlených uživatelů a zainteresovaných stran.

Alokace zdrojů: Plánování pomáhá alokovat potřebné zdroje pro projekt, včetně personálu, hardwaru a softwaru. To pomáhá zajistit, že projekt má potřebné zdroje k dosažení svých cílů.

Vytvoření plánu: Plánování pomáhá vytvořit plán projektu, který obsahuje milníky projektu, výstupy a termíny. To pomáhá zajistit, že projekt zůstane na správné cestě a že bude dosaženo pokroku směrem k cílům projektu.

Identifikace rizik: Plánování pomáhá identifikovat potenciální rizika pro projekt, jako jsou technické problémy, rozpočtová omezení a změny požadavků. To umožňuje projektovému týmu vyvinout strategie ke zmírnění těchto rizik a minimalizaci jejich dopadu na projekt.

Celkově je plánování vývoje softwaru zásadní pro zajištění úspěšnosti projektů vývoje softwaru. Definováním rozsahu projektu, identifikací požadavků, alokací zdrojů, vytvořením plánu a identifikací rizik mohou projektovým týmům zajistit, aby software vyhovoval potřebám jeho uživatelů, zainteresovaných stran a organizace jako celku. [30]

5.1 Požadavky

Definování požadavků je důležité při vývoji softwaru, rozdělují se na funkční požadavky a nefunkční požadavky.

Funkční požadavky popisují specifické vlastnosti a funkce, které musí softwarový systém plnit, aby vyhovoval potřebám svých uživatelů. Tyto požadavky definují, co by měl systém dělat a jak by se měl chovat. Příklady funkčních požadavků můžou být požadavky na uživatelské rozhraní, požadavky na ukládání a získávání dat, požadavky na validaci dat, požadavky na zpracování, požadavky na reportování.

Nefunkční požadavky naopak popisují kvalitativní atributy softwarového systému. Tyto požadavky specifikují, jak dobře by měl systém fungovat. Zaměřují se na vlastnosti, jako je použitelnost, spolehlivost, výkon, zabezpečení a udržovatelnost. Příklady nefunkčních požadavků můžou být požadavky na použitelnost, jako je snadnost použití a pochopení aplikace, požadavky na výkon, jako je doba odezvy, požadavky na zabezpečení, požadavky na dostupnost, jako je doba provozuschopnosti a zotavení po havárii, požadavky na škálovatelnost, jako je schopnost zvládnout zvýšený provoz nebo objem dat nebo požadavky na údržbu, jako je snadná údržba a rozšiřitelnost.

Zatímco funkční požadavky definují, co by měl softwarový systém dělat, nefunkční požadavky definují, jak dobře by to měl dělat. [31]

5.1.1 Funkční požadavky

Funkční požadavky definují, co by měl softwarový systém dělat, aby vyhovoval potřebám svých uživatelů. Pro vytvářenou docházkovou aplikaci:

- Ověření/autentizace uživatele: Systém by měl vyžadovat, aby se uživatelé pro přístup k údajům o docházce autentizovali pomocí metod, jako jsou hesla, PINy nebo jiné autentizační metody.
- Správa uživatelů: Systém by měl oprávněným uživatelům umožnit vytvářet účty, spravovat profily a členit je do skupin, podle kterých by jim byla přiřazená práva.
- Sledování času: Systém by měl být schopen sledovat a zaznamenávat čas, který každý uživatel stráví v práci, pomocí aktivit, které si uživatelé v systému zadávají.

- Správa aktivit: Systém by měl oprávněným uživatelům umožnit vytvářet a spravovat aktivity, které mají různé vlastnosti jako započítat do odpracované doby, maximální strávená doba aktivity během dne, možnost zapsat aktivitu jen určitou skupinou.
- Správa nepřítomnosti: Systém by měl schopen sledovat a spravovat nepřítomnosti zaměstnanců pomocí zadaných aktivit jako jsou pracovní neschopnosti, dovolené.
- Aktualizace v reálném čase: Systém by měl být schopen poskytovat právě odpracovanou dobu a součty dalších aktivit v reálném čase.
- Reporting: Systém by měl generovat reporty na základě uživatelského vstupu. Nejlépe pro integraci s jinými systémy, jako je mzdový software.
- Zadávání a ověřování dat: Systém by neměl dovolit zadávat chybné údaje, měl by ověřovat jejich správnost a okamžitě upozornit uživatele v UI.

5.1.2 Nefunkční požadavky

Nefunkční požadavky definují, jak se by měl softwarový systém chovat, aby vyhovoval potřebám svých uživatelů.

- Spolehlivost: Software by měl být spolehlivý a dostupný za všech okolností, s minimálními odezvami nebo přerušením přístupu uživatelů. Software by také měl být schopen zpracovat a zotavit se z chyb nebo selhání bez ztráty dat nebo poškození dat.
- Škálovatelnost: Software by měl být schopen zvládnout rostoucí počet uživatelů, dat nebo požadavků, jak organizace roste.
- Zabezpečení: Software by měl být bezpečný a měl by chránit uživatelská data před neoprávněným přístupem, manipulací nebo krádeží.
- Výkon: Software by měl být schopen dobře fungovat při vysokém uživatelském zatížení a velkém objemu dat s minimálním zpožděním nebo zpomalením. Měl by být optimalizován tak, aby se rychle načítal a včas reagoval na vstupy uživatelů.
- Použitelnost: Software by měl být snadno použitelný a intuitivní, s čistým a organizovaným uživatelským rozhraním. Měl by být navržen s ohledem na

uživatelskou zkušenost a měl by být přístupný uživatelům s různou úrovní technických znalostí.

- Údržba: Software by měl být udržovatelný a snadno aktualizovatelný nebo upravovatelný podle potřeby. S jasnými standardy kódování a osvědčenými postupy.

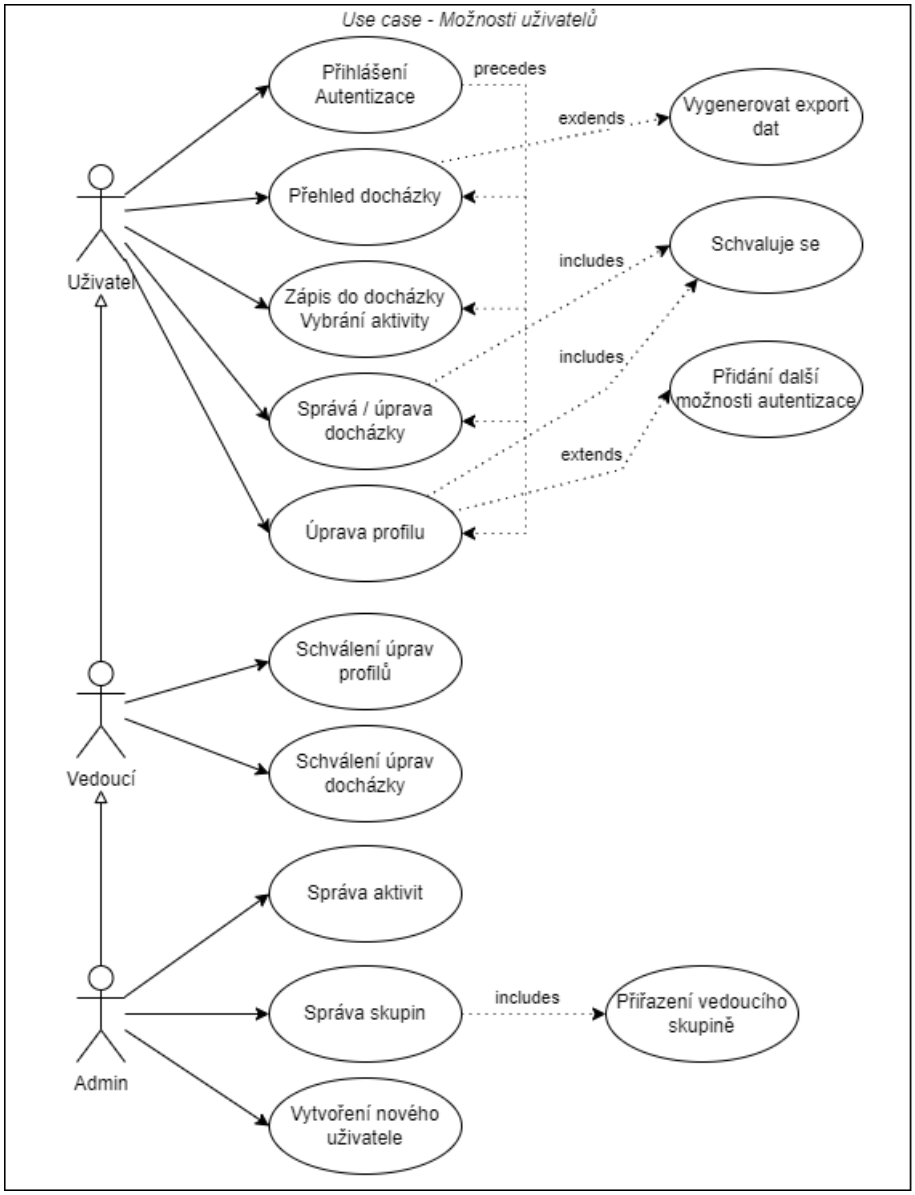
To jsou nefunkční požadavky, které může mít docházková aplikace.

5.2 Scénáře využití

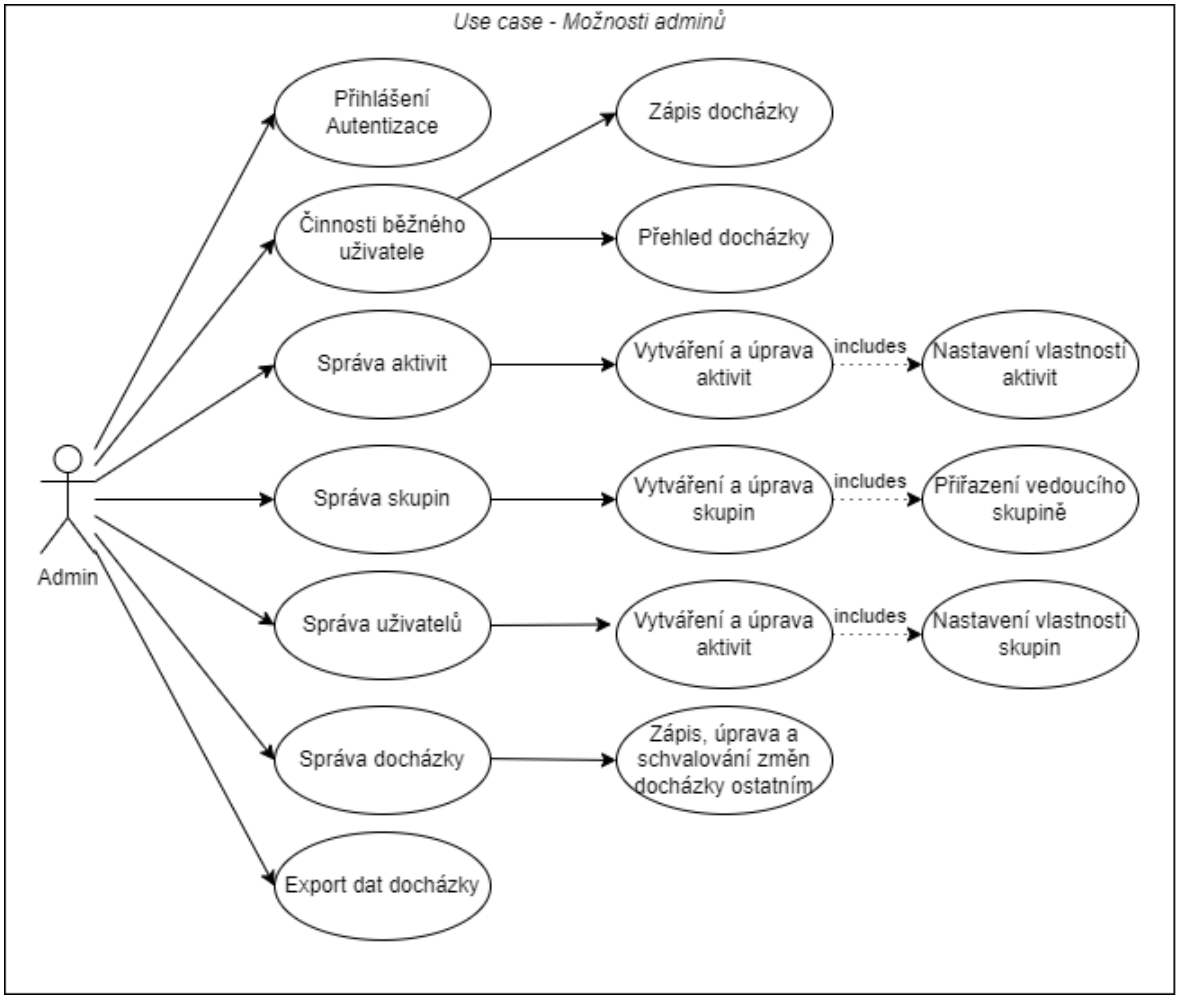
Scénáře využití jsou vytvořeny jako typ diagramu případu užití. Případ užití je popis specifického chování nebo interakce mezi systémem a jeho uživateli. Je to způsob, jak zachytit funkční požadavky systému, a to popisem jakým s ním mohou uživatelé komunikovat.

5.2.1 Možnosti uživatelů

Jednotlivé možnosti uživatelů jsou znázorněny na Obrázek 35. Uživatelé jsou rozdělení do tří kategorií. Nejnižší a základní kategorie je bežný uživatel, aby mohl provádět funkce a akci, tak se vždy musí nejprve přihlásit / autentizovat. Jakmile je přihlášen, má možnost si prohlédnout svoji docházku, případně z ní vygenerovat export, zápsat si příchod výběrem některé z nabízených aktivit, možnost upravit si předešlou docházku a svůj profil. Po úpravě docházky a profilu musí přijít ještě schválení. Úprava profilu zahrnuje i přidávání dalších klíčů, díky kterým se uživatel může přihlásit. Další kategorií je vedoucí, který má všechny možnosti jako bežný uživatel a navíc může schvalovat úpravy docházky a profilu uživatelů, kteří jsou v jeho skupině, kde je nastaven jako vedoucí. Zároveň mu aplikace poskytuje přehled docházky a možnosti úprav docházky i pro uživatele ve skupině. Poslední a nejvyšší kategorie je admin, který má všechny možnosti jako uživatel i jako vedoucí, ale úpravat a schvalovat může všechny, místo jen skupiny, tak jako tomu je u vedoucího. Zároveň admin spravuje nastavení docházky jako je vytváření a nastavování aktivit, což může být například práce a nastavení, jak se počítá a pro které skupiny je aktivita dostupná. Nastavuje skupiny, které může vytvářet, přiřazovat do nich vedoucí a uživatele a nastavovat skupině defaultní vlastnosti. Může upravovat a jako jediný vytvářet nové uživatele. Detailní zobrazení možností admina je na Obrázek 36.



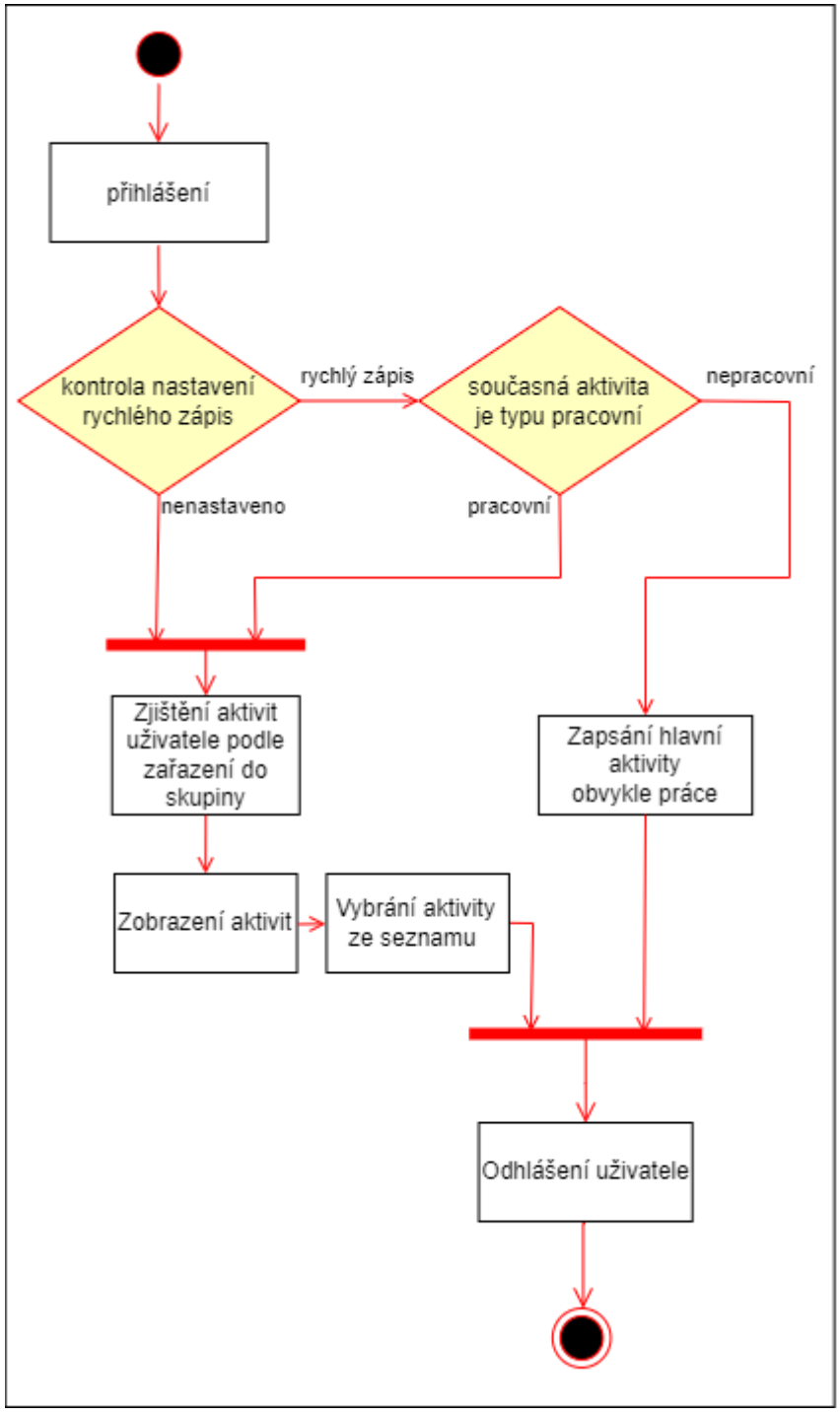
Obrázek 35 Případ užití: Možnosti uživatelů



Obrázek 36 Případ užití: Možnosti adminů

5.2.2 Zapsání aktivity do docházky uživatelem

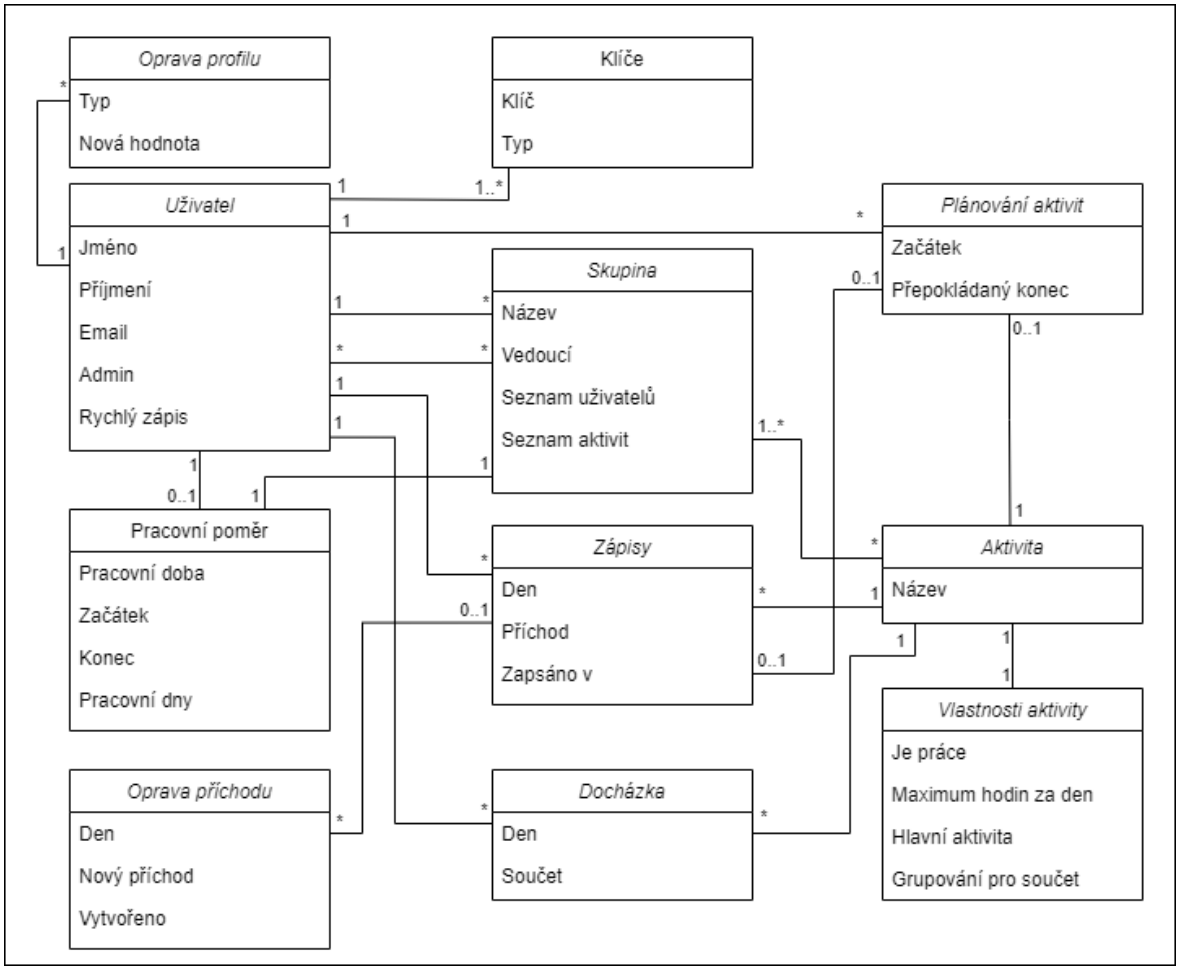
Uživatel se autentizuje / přihlásí. Jakmile je přihlášen, podle zařazení do skupiny, aplikace zobrazí aktivity, mezi kterými může vybírat. Zároveň ještě před zobrazením aktivity je u uživatele provedena kontrola, zda má nastaven rychlý zápis. V případě, že rychlý zápis má aktivovaný, tak se nejprve zkontroluje současná aktivita a pokud je aktivita nepracovní, tak se automaticky přepíše na hlavní aktivitu, která bude ve většině případů práce. V případě, že rychlý zápis aktivovaný není nebo současná aktivita je pracovní, tak se zobrazí seznam aktivit, ze kterých uživatel může vybrat. Po výběru aktivity nebo po rychlém zápisu aktivity je uživatel vždy odhlášen, aby byla docházka připravena k přihlášení a zápisu docházky dalším uživatelem. Pro znázornění je použit diagram aktivit zobrazen na Obrázek 37.



Obrázek 37 Diagram aktivit: Zapsání aktivity uživatelem

5.3 Doménový model

Doménový model je formou diagramu tříd a ilustruje strukturu systému tím, že ukazuje třídy, jejich atributy a vztahy mezi nimi. Základní třídy jsou Uživatel, Zápisy a Aktivita. Třída Zápisy obsahuje informace o jednotlivých příchodech uživatelích a aktivitě, kterou vybrali, to tvoří hlavní kostru aplikace. Uživatel může mít několik zápisů, ale zápis má vždy právě jednoho uživatele, proto je mezi nimi vztah uživatel 1-* zápisy. Uživatel může mít více klíčů, od toho je třída Klíče, ale vždy alespoň jeden, aby měl možnost se přihlásit. Třída Skupina má vždy jednoho vedoucího, který je uživatel, ale jeden uživatel může být vedoucí více skupin, zároveň třída Skupina má seznam uživatelů, kde může být libovolný počet uživatelů. Podobně je tomu se seznamem aktivit povolených pro skupinu. Třída aktivita má vždy vlastnosti, podle kterých se pak počítá. Počítání součtů aktivit za jednotlivé dny je ve třídě Docházka, kde vždy při zápisu do tabulky zápis přepočítávají i součty ve třídě Docházka pro daný den.



Obrázek 38 Plánovaný doménový model

5.4 Možnosti autentizace

Prvním krokem uživatele pro práci s aplikací je vždy se přihlásit, tedy autentizovat. Bez toho nemá oprávnění, tedy autorizace, cokoliv provádět. Přihlášení musí být jednoduché, rychlé a zároveň bezpečné. Způsoby autentizace jsou závislé na hardwaru, kterým se zadává nebo čte. Klávesnicí lze zapsat PIN nebo login a heslo. Pro další typy autentizace je potřeba dodatečný hardware jako čtečka RFID čipů, čtečka NFC, čtečka otisku prstu, rozpoznání obličeje nebo oční duhovky.

5.4.1 PIN

PIN je typu uživatelské znalosti. Jedná se o několikamístný počet znaků, obvykle čtyřmístné až šestimístné číslo. Může být i delší a skládat se i z písmen. V aplikaci docházky může být PIN zadán bez přihlašovacího jména pro rychlou autentizaci. Problém nastává, když PIN zadá uživatel, kterému nepatří. To se může stát překlikem uživatele nebo schválně zadání cizího PINu.

5.4.2 Login a heslo

Login (přihlašovací jméno) a heslo je typu uživatelské znalosti. Jedná se o nejčastější přihlašování na internetu. Uživatel zadá své přihlašovací jméno, které může být veřejně známé a k tomu své heslo, které zná pouze on. Aby bylo heslo bezpečné, mělo by mít alespoň osm až dvanáct znaků, mělo by být složité, to znamená, že by mělo obsahovat velká a malá písmena, číslice a speciální znaky, zároveň by mělo být pro každou aplikaci unikátní, kvůli bezpečnosti ostatních aplikací, kdyby z jedné databáze hesel unikla. Pro aplikaci docházky je tato varianta velmi bezpečná, ale zároveň i velice pomalá. V jeden moment může přijít do práce několik zaměstnanců zároveň a kdyby se všichni měli ověřovat přihlašovacím jménem a bezpečným heslem tvořili by se fronty na zapsání docházky.

5.4.3 RFID čip nebo karta

RFID čip nebo karta je typu uživatelského vlastnictví. Uživatel pouze přiloží čip nebo kartu ke čtečce a unikátní číslo, které vysílá je přečteno. Pokud v systému některý z uživatelů toto číslo má zapsané jako jeden z klíčů, je autentizován. Tento přístup je velice rychlý a dá se považovat i za bezpečný, pokud se jedná o pouhé autentizování do docházky. RFID čipy lze snadno replikovat, a proto by se neměly využívat pro autentizaci do systémů, kde je bezpečnost velmi důležitá.

5.4.4 NFC mobilní telefon

NFC je nástupce RFID a využívají ji i všechny současné moderní telefony například k platbám jako náhrada bankovní karty. Funguje stejně jako RFID čipy, uživatel přiloží telefon se zapnutým NFC a často i se zapnutou aplikací, která je vytvořena přímo pro danou čtečku. Aplikace může být nutná, protože jinak se NFC v telefonech snaží najít platební portál a navázat s ním komunikaci, aplikace zapříčí, že místo pokusů o navázání komunikace s platebním portálem vysílá ID tedy unikátní číslo telefonu. Tento přístup je velice rychlý i bezpečný. Nutností je mít telefon s NFC a čtečku, která NFC umí číst.

5.4.5 Biometrické metody

Biometrické metody jsou typu uživatelské charakteristiky. Vychází z přesvědčení, že některé biologické charakteristiky jsou pro každého člověka jedinečné a neměnitelné. Základní využívané biometrické metody jsou otisk prstů, rozpoznání obličeje nebo rozpoznání oční duhovky.

[32] [33]

6 REALIZACE APLIKACE

6.1 Životní cyklus

Plánování: V této fázi byly určeny cíle, požadavky a rozsah aplikace. Definice vlastností a funkcí aplikace a identifikace cílů.

Design: V této fázi je navržena architektura aplikace, uživatelské rozhraní a systémové požadavky. Vzor MVVM je použit k vytvoření jasného oddělení logiky aplikace a uživatelského rozhraní.

Vývoj: V této fázi se aplikace vyvíjí pomocí technologií WPF a .NET. Proces vývoje zahrnuje vytvoření funkčnosti aplikace, její testování a opravu případných chyb nebo problémů.

Testování: V této fázi bude aplikace testována, aby se zajistilo, že splňuje požadavky a funguje tak, jak bylo zamýšleno. Mohou být prováděny různé typy testování, včetně funkčního testování, testování výkonu a testování použitelnosti.

Nasazení: V této fázi bude aplikace nasazena do produkčního prostředí. To může zahrnovat instalaci aplikace na uživatelská zařízení.

Údržba: V této fázi bude aplikace udržována a aktualizována, aby se opravily případné chyby nebo problémy, které se vyskytnou, a také aby se přidaly nové vlastnosti nebo funkce. Aby aplikace zůstala spolehlivá a bezpečná, byla by prováděna pravidelná údržba.

6.2 Design, vývoj a testování

Vývoj aplikace vychází z vytvořené šablony v kapitole 3. Šablona je rozšířena o modální navigaci, což jsou okna otevřená nad současným oknem. Často se jedná například o vytvoření, zápis nebo úpravu položky z listu, například vytvoření nového uživatele. Dále je rozšířena o asynchronní příkazy `AsyncCommandBase`, které slouží pro příkazy, které se mají spustit asynchronně.

6.2.1 Rozdělení projektu

Projekt je rozdělen do tří podprojektů Domain, EF a WPF. Projekt Domain obsahuje modely tedy třídy, se kterými pracuje celá aplikace, projekt EF obsahuje vytvoření spojení s databází a přístupy do databáze jako vybrání dat, vložení dat, úprava dat a smazání dat. Projekt WPF

pak obsahuje vytvořenou šablonu, kde modely jsou přesunuty do projektu Domain. V projektu WPF je následně vytvořena celá aplikace.

Rozdělení slouží pro případné rozšíření, například pro webové rozhraní, které by mohlo využívat projekt Domain a EF k získání již naprogramovaných modelů a přístupu a práci s databází.

6.2.2 Kroky vývoje

Celá aplikace byla vytvořena v 44 krocích. Každý krok přidával nové možnosti, opravoval stávající funkce nebo opravoval existující chyby.

6.2.2.1 Krok 1-3

Nejprve byl vytvořen projekt, do kterého by implementována šablona z kroku 3 rozšířená o modální navigaci a asynchronní příkazy. Následně do Domain byly vytvořeny modely:

- DomainObject: ID
- Activity: Aktivita – název, zkratka a vlastnosti aktivity
- ActivityProperty: Vlastnosti aktivity – určuje u aktivit, zda počítat, je plán, má pauzu, podle kterého názvu slučovat do exportu
- AttendanceRecord: Záznam v docházce – datum a čas vstupu pro konkrétní aktivitu a uživatele
- AttendaceRecordDetail: Rozšíření záznamu v docházce – předpokládaný návrat a popis
- AttendaceTotal: Součet aktivity pro uživatele a konkrétní den – součet trvání aktivity pro uživatele a den
- Group: Skupina – název, vedoucí, uživatelé (podřízení) a závazek
- Key: Klíč uživatele k autentizaci do docházky – hodnota klíče
- Obligation: Závazek – informace o závazku skupiny nebo uživatele a dostupné aktivity
- User: Uživatel – jméno, příjmení, email, zda se jedná o admina, skupina, klíče, závazek

Všechny modely dědí ze třídy `DomainObject`, která obsahuje pouze ID, díky tomu při práci s jakýmkoliv modelem, lze předpokládat existenci ID a jeho využití.

Následně byly do projektu přidány všechny předpokládané View, které bude aplikace využívat:

- Aktivity: pro zobrazení aktivit `ActivitiesView`, `ActivityPropertyView` a pro přidání a úpravu `ActivityUpsertView`
- Skupiny: `GroupsView` pro zobrazení skupin, `GroupsRequestsProfileUpdateView` a `GroupsRequestsFixAttendanceRecordView` pro schvalování úprav profilu a docházky podřízených uživatelů. `GroupUpsertView` pro úpravu a přidávání skupin a `GroupAddUser` pro přidávání uživatelů do skupin
- Uživatel: `UsersView` zobrazení všech podřízených, `UserDailyOverviewView` zobrazuje jednotlivé příchody a součty aktivit pro vybraný den, `UserFixAttendanceRecordView` úprava zápisů do docházky, `UsersHistoryView` zobrazuje dny v měsíci a celkově odpracovanou dobu, `UserKeyUpsertView` pro úpravu nebo přidání klíče k uživateli, `UserProfileUpdateView` pro úpravu profilu uživatele, `UserProfileView` pro zobrazení profilu uživatele, `UserSelectActivityView` pro výběr možných aktivit, `UserSelectActivitySpecialView` po výběru aktivity, která je například plán, se zobrazí toto View sloužící k zapsání dodatečných informací jako začátek, konec a popis, `UserUpserView` pro úpravu a vytváření nových uživatelů
- Základní: `HomeView` zobrazuje pole pro zadání klíče, `NavigationBarView` horní lišta s logem a s hodinami, zobrazující tlačítka podle toho, který uživatel je přihlášen a jaké má práva

6.2.2.2 Krok 4-8

V tomto kroku byly vytvořeny ViewModely `HomeViewModel`, `NavagationBarViewModel`, `UserMenuViewModel`, `UserDailyOverviewViewModel` a `UserSelectActivityViewModel`.

Aplikace pro testování vytváří uživatele a všechny hodnoty, s kterými aplikace pracuje v `MainViewModel` v konstruktoru, který je spuštěn vždy jen jednou při spuštění aplikace. Vytvoří se uživatel admin s klíčem admin a implementuje se třída `HomeViewModel`, která při zapsání klíče, který má nejméně 3 znaky, kontroluje, zda uživatel s tímto klíčem existuje, pokud ano, provede se navigace do nového View – `UserMenuView`, který obsahuje

`UserSelectActivityView` a `UserDailyOverviewViewModel`, pro zobrazení aktivit k výběru a současně denních statistik právě probíhajícího dne. Uživatel, který se úspěšně přihlásil je zapsán do třídy `CurrentUser`, což je třída typu Store, která uchovává v průběhu aplikace informaci o tom, která uživatel je právě přihlášen.

Zároveň je implementován `NavigationBarViewModel`, který zobrazuje logo, datum a probíhající čas a taky tlačítka pro přihlášeného uživatele. Většina tlačítek v tomto stavu je zatím nefunkčních, pouze připravených k budoucí implementaci.

Aplikace při spuštění vytváří i aktivity, se kterými pak pracuje `UserSelectActivityViewModel` a zobrazuje je jako jednotlivá tlačítka pro vybrání a zapsání uživatelem. Navíc každá aktivita má svoji zkratku, která je zvýrazněna a pokud je zadána na klávesnici, tak je uživatelem vybrána a zapsána. Zápis probíhá pomocí funkce v `CurrentUser`, který obsahuje List všech zápisů do docházky (`AttendanceRecord`). Zároveň `CurrentUser` poskytuje funkce na výpočet celkové doby v práci pro den, celkové odpracované doby a celkové doby pauzy. Zároveň každý zápis přepočítává pro daného uživatele a den součty docházky (`AttendanceTotal`). Díky tomu je součet docházky vždy všech zápisů do docházky, kromě právě probíhajícího (protože není dopředu jasné, kdy bude končit, takže se připočítává průběžně), vždy aktuální a nemusí se například přepočítává na konci dne automat nebo dávkou.

Po těchto krocích je uživatel se schopen svým klíčem přihlásit do docházky a vybrat si aktivitu, která se mu pak počítá, takže vidí průběžné časy doby strávené různými aktivitami.

6.2.2.3 Krok 9-11

V těchto krocích byl nejprve vytvořen `UserKeyViewModel` a `UserKeyUpsertViewModel` k odpovídajícím Views, sloužící pro uživatele a zobrazení všech jeho klíčů, úpravu a přidání nových klíčů k uživateli. `UserKeyUpsetView` je první použití modální navigace a modálního pohledu. Při úpravě nebo vytváření nového klíče se zobrazí okno nad současným oknem. Současné okno ztmavne a v nově otevřeném okně má uživatel možnost vytvořit nebo upravit klíč nebo modální okno zavřít a vrátit se na přehled klíčů.

Následně byl vytvořen a implementován `UserProfileViewModel`, který zobrazuje profil právě přihlášeného uživatele a umožňuje mu si svůj profil upravit. Profil obsahuje jméno, příjmení, email a závazek, který obsahuje informace o pracovním závazku jako příchod a odchod z práce nebo dny, kdy uživatel pracuje. Po úpravě uživatele profilu je vytvořena pouze žádost o úpravu profilu. V aplikační logice to znamená, že je vytvořen nový uživatel,

který má nastavenou hodnotu `ToApprove` na `true` (ke schválení) a odkazuje `UserId` na `ID` originálního uživatele. Uživatel všechny své žádosti o úpravu profilu v `UserProfileView` vidí, má možnost je smazat nebo upravit. Jakmile bude žádost o úpravu profilu schválena, tak zmizí i žádost.

6.2.2.4 Krok 12-13

Pro úpravu a vytváření skupin byl vytvořen `GroupsViewModel`, který zobrazuje existující skupiny, jejich uživatele, jejích vedoucí a závazek. Při spuštění aplikace jsou nově vytvořeny i dvě skupiny Vedení a Základní. Skupiny jsou uchovávány v `GroupStore`. Přidávání uživatelů bylo zamýšleno nejprve pomocí modálního okna `GroupAddUserView`, ale nakonec je vše řešeno v `GroupsView`.

`GroupView` tedy zobrazuje všechny existující skupiny, podle vybrané skupiny zobrazuje list uživatelů v dané skupině. Pro vybranou skupinu dále umožňuje zobrazit její Nastavení, což je vedoucí a závazek. Tento závazek pak mají všichni uživatelé skupiny, pokud u nich nebyl nastaven jiný úpravou profilu. Díky tomu se nemusí nastavovat závazek každému vytvořenému uživateli zvlášť a stačí jej přidat do odpovídající skupiny. Do skupiny lze přidávat další uživatele, zobrazí se seznam uživatelů, kteří ve skupině nejsou a je možné je přidat. Podobně je tomu při změně vedoucího, zobrazí se seznam uživatelů, kteří nejsou právě vedoucím vybrané skupiny a je možno je nastavit jako vedoucí. Další možností je vytvořit a smazat skupinu. Při tvorbě skupiny je využito modální okno `GroupUpsertView`, kde se zadává pouze název a uživatel (měl by to být vždy admin, protože nikdo jiný by neměl mít právo na to, se do nastavení skupin dostat), který skupinu vytvořil je nastaven i jako vedoucím skupiny a závazek je nastaven automaticky a po vytvoření by se pomocí nastavení měl změnit.

Dále budou mít skupiny i možnost vybrat aktivity, které jsou pro danou skupinu povolené a pouze ty budou uživatelé s danou skupinou mít možnost vybrat, bude implementováno v pozdějším vývoji.

6.2.2.5 Krok 14-15

Implementace `ActivityViewModel`, který slouží k vytváření a úpravě aktivit. `ActivityView` zobrazuje seznam existujících aktivit a po vybrání její detail a vlastnosti. Detail obsahuje název a zkratku, vlastnosti jsou z modelu `ActivityProperty` a určují, zda se jedná o aktivitu, která se má počítat, zda se jedná o přestávku, zda se jedná o plán, jak často je

vyžadována pauza, jak dlouho je nutná vyžadovaná pauza, podle jakého názvu seskupovat do exportu. `ActivityUpsertView` je další z modálních oken, který slouží k vytvoření nové aktivity a jejích vlastností. Aktivity obdobně jako skupiny mají vlastní Store – `AcitivityStore`, kde jsou všechny uloženy a funkce v ní slouží pro práci s nimi.

Následně byl namodelován `UsersView`, který obsahuje seznam podřízených uživatelů, možnost vytvořit nového uživatele a úpravu profilu uživatele (bez schválení oproti úpravě samotným uživatelem), úprava a přidávání klíčů podřízeným uživatelům a náhled jejich docházky. Pohled je zatím nefunkční, pouze připraven k napárování `UsersViewModel`.

6.2.2.6 Krok 16-17

Vytvoření a implementace `UserHistoryViewModel` sloužící k vybrání měsíce a roku a zobrazení jednotlivých dnů v měsíci a celkové odpracované doby pro přihlášeného uživatele. Po vybrání konkrétního dne je zobrazena denní statistika z `UserDailyOverviewView`, to znamená, že `UserHistoryViewModel` obsahuje `UserDailyOverviewViewModel`, ve kterém podle vybraného dne mění datum, pro které má zobrazit denní sumy aktivit, denní příhody a denní součty.

6.2.2.7 Krok 18

V tomto kroku jsou přepracovány aktivity a je vytvořen nový model `ActivityGlobalSetting`, který obsahuje délku, po které je nutná pauza, délka nutné pauzy, dále informace o hlavní pracovní aktivitě a hlavní nepracovní aktivitě, a nakonec délku celodenní aktivity a délku půldenní aktivity. Hlavní pracovní a nepracovní aktivity a délka celodenní a půldenní aktivity budou využity v dalších krocích při implementaci aktivit, které jsou plány. Díky tomu, z modelu `ActivityProperty` vlastnosti aktivity jsou odebrány tyto parametry a je přidán jeden nový a to `MaxInDay`, který určuje maximální délku aktivity za den.

Kvůli tomu je přepracován `ActivityStore`, který nově obsahuje informace o `ActivityGlobalSetting` a následně je přepracován i `ActivityView`, který zobrazuje globálního nastavení a umožňuje ho měnit.

Délka, po které je nutná pauza, slouží k vynucení pauzy po určité době určité aktivity. Například může existovat aktivita práce, která má vlastnost `HasPause` (má nutnou pauzu), pokud její trvání přesáhne délku, po které je nutná pauza, délka práce se zkrátí o délku nutné

pauzy a zapíše se aktivita pauza. Často je délka, po které je nutná pauza 6 hodin a délka pauzy 30 minut. Ze zákona Přestávka v práci a bezpečnostní přestávka (§ 88 a 89):

„*Zaměstnavatel je povinen poskytnout zaměstnanci nejdéle po 6 hodinách nepřetržité práce přestávku v práci na jídlo a oddech v trvání nejméně 30 minut.*“ [34]

Takže i kdyby si uživatel pauzu nevybral zapsáním některé z aktivit, které mají vlastnost `IsPause` (je pauza), tak se mu pauza zapíše automaticky, protože se předpokládá, že si ji vybral i během své pracovní aktivity, aniž by si ji zapsal do docházkové aplikace.

6.2.2.8 Krok 19

Vytvoření a implementace `UserViewModel` k již vytvořenému pohledu `UserView`. Obsahuje seznam všech podřízených uživatelů. Admin může vytvářet nové uživatele pomocí modálního okna `UserUpsertView`. Po vybrání jednoho z podřízených je možné zobrazit jejich klíče, které lze upravovat nebo přidávat, zobrazit jejich profil, který lze upravit (pouze admin může upravovat skupinu) a docházku obdobně jako tomu je v `UserHistoryView`.

6.2.2.9 Krok 20-21

V těchto krocích je upraven výběr aktivit, pokud se jedná o aktivitu, která je typu plán. Pokud se aktivita typu plán v `UserSelectActivityView` vybere, tak je otevřen nově vytvořený modální pohled `UserSelectActivitySpecialView`, ve kterém uživatel zapisuje, kdy plánovaná aktivita začíná a kdy končí a případně její popis. Zároveň se rozlišují aktivity typu plán na dvě kategorie, kde v jedné se zadává i čas, například služební cesta, a v druhé bez času se navíc rozlišuje, zda se jedná o celodenní aktivitu nebo o půldenní aktivitu a podle toho se zadává počáteční a koncové datum nebo jen datum dne a zda se jedná a půlden na začátku nebo na konci, například dovolená.

Zápis plánu probíhá stejně jako běžné aktivity pouze s budoucím datem, tedy datem začátku plánu a do `AttendanceRecordDetail` je uložen předpokládaný začátek, konec a popis. Každý `AttendanceRecord` (zápis do docházky) nově obsahuje volitelně třídu `AttendanceRecordDetail` (detail zápisu do docházky). Navíc kromě zápisu samotného plánu je zapsána i další aktivita, která má datum konce plánu, a jedná se o aktivitu z globálního nastavení aktivit – hlavní nepracovní aktivitu (domov). Ve výsledku se tedy zapíše aktivita typu plán s datem začátku plánu a hlavní nepracovní aktivita začínající koncem plánu. Díky tomu není potřeba žádný automat, který by pravidelně kontroloval

výskyt plánu, který není zapsaný do docházky a snažil se ho zapsat, protože všechny plány jsou zapsány a začnou probíhat se začátkem naplánování.

Tímto způsobem zápisu plánů by mohl vzniknout problém, když by plán začínal například 13:00 a uživatel byl v práci až do 13:30 a na plán by odcházel později, než bylo naplánováno. Nebo obdobně, když by se z plánu vrátil dříve, než skončil. Proto třída `UserMenuViewModel`, která je vytvořena vždy po úspěšném přihlášení uživatele, kontroluje, zda u přihlášené uživatele je současná probíhající aktivita typu plán, pokud ne, nic se neděje a uživatel může vybrat normálně aktivitu, pokud ano, je zobrazen nový modální pohled `UserHasCurrentlyPlanView`. Pohled informuje uživatele o probíhajícím plánu, od kdy do kdy trvá a nabízí mu možnosti:

- Zapsat až od teď – díky tomu, kdyby uživatel na plán odcházel z práce například po půl hodině, než byl její plánovaný začátek, tak by zvolil tuto možnost a začátek plánu se upraví na současný čas
- Zrušit plán – uživatel se může rozhodnout, že plán by neměl vůbec probíhat, tím se v docházkových záznamech úplně smaže, jak začátek plánu, tak i vytvořený konec (domov) plánu a v docházce mu probíhá aktivita, kterou měl zapsanou před zapsáním plánu
- Ukončit už teď – naopak, kdyby se uživatel z plánu vrátil dříve, vybere tuto možnost a plán se ukončí na současný čas, zapsaný konec hlavní nepracovní aktivitou se smaže a do docházky se zapíše hlavní pracovní aktivita (práce). Tím se předpokládá, že uživatel se z plánu vrátil dříve a má možnost vybrat novou aktivitu. Kdyby se automaticky nezapsala hlavní pracovní aktivita a uživatel plán pouze ukončil dříve a žádnou aktivitu nevybral, tak by plán stále probíhal, protože aktivity a plány se vždy ukončují zapsání nové aktivity.

6.2.2.10 Krok 22-23

V těchto krocích je pro uživatele přidána možnost vytvořit žádost o úpravu docházky. V pohledu `UserDailyOverviewView` je nyní možnost vybrat jednotlivé příchody a upravit je nebo odebrat. Nebo přímo přidat nový zápis. Při úpravě a zápisu se zobrazí nově vytvořený modální pohled `UserFixAttendanceRecordView`, který umožňuje vybrat aktivitu, datum a čas, buď pro vytvoření nového zápisu nebo při úpravě existujícího. Podobně jako při úpravě profilu je pouze vytvořena žádost o úpravu a aplikace je navigována do nového pohledu `UserFixesAttendanceRecordView`. To obsahuje seznam všech žádostí o úpravu docházky,

o jaký typ se jedná (úprava, přidání, odebrání) a zda je žádost schválena, zamítnuta nebo čeká na vyřízení. Zároveň pohled zobrazuje list všech zápisů docházky, které lze upravovat, odebírat nebo lze přidat nový zápis, stejně jako z `UserDailyOverviewView`.

Zároveň je v tomto kroku vytvořen nový Store `AttendanceRecordStore`, který obsahuje listy jednotlivých příchodů, součtů aktivit a žádostí o opravu docházky. Všechny funkcionality spojené se zápisem docházky jsou předělány do `AttendanceRecordStore`.

6.2.2.11 Krok 24-25

Pro možnost schvalování žádostí oprav docházky a oprav profilů uživatelů je vytvořen pohled `UsersRequestsView`, který obsahuje seznam všech žádostí o změnu zápisu docházky a seznam všech žádostí o změnu profilu. Po vybraní žádosti jsou zobrazeny předešlé hodnoty a nové hodnoty a možnost schválit nebo zamítnout.

Po schválení změny zápisu docházky je docházka a součty aktivit pro daného uživatele v den, pro který byly změny provedeny, přepočteny.

Každý vedoucí vidí pouze žádosti svých podřízených (stejně tomu je v `UsersView`).

6.2.2.12 Krok 25-28

V těchto krocích je nejprve přidaná funkcionalita pro rozpoznání vedoucího po přihlášení. Pokud se jedná o vedoucího některé ze skupin, zobrazují se mu navíc tlačítka Podřízení (`UsersView`) a Žádosti (`UsersRequestsView`). Uživatelé, kteří vedoucí nejsou, tlačítka nevidí. Admin vidí vše a v Podřízených a Žádostech vidí všechny nezávisle na tom, zda je vedoucí nebo ne.

Do skupin byla přidaná plánovaná možnost přidávat a odebírat aktivity. Uživatelé zařazení v dané skupině pak vidí a mají možnost vybrat pouze aktivity, které jsou skupině povoleny. To může být užitečné například ve skupině brigádnici nebo noví, kde zadávání dovolené nebo homeoffice by nemělo smysl a bylo pro ně spíše zakázané.

Protože dostupné aktivity jsou pouze na základě skupiny, jsou předělány ze závazku přímo do skupiny.

6.2.2.13 Krok 29-30

V tomto kroku je upravena funkce na výpočet součtů aktivit za den pro uživatele. Nově zahrnuje při výpočtu i povinnou pauzu a automaticky ji přidává. Zároveň pro každou aktivitu kontroluje, zda nepřesáhla své povolené maximum za den.

6.2.2.14 Krok 31-32

Již od implementace šablony aplikace obsahuje `MessageStore` pro předávání zpráv a informování o změnách. V těchto krocích je zahrnut do všech příkazů a změn, kde je potřeba informovat uživatele o tom, co se stalo. `MessageStore` je upraven tak, že zpráva je zobrazena vždy 5 sekund a poté zmizí. Například při zápisu aktivity je docházka přesměrována na `HomeView` (uživatel zapsal aktivitu a je odhlášen) a zobrazí se informace / zpráva o tom, který uživatel si zapsal jakou aktivitu.

Zároveň byla přidána pro všechny uživatele možnost nastavit si „Rychlý zápis hlavní aktivity“. Hlavní pracovní aktivita je nastavena v globálním nastavení aktivit, často zvolena jako práce. Pokud uživatel má nastaven rychlý zápis, tak při přihlášení je zkontrolována jeho současná aktivita, pokud se jedná o aktivitu, která se nepočítá nebo o pauzu, která není plán, tak se do docházky automaticky zapíše hlavní pracovní aktivita a uživatel se ani nedostane do pohledu s výběrem aktivit.

Navíc byl přidán pohled `UserPlanView`, který rozděluje zobrazené aktivity podle toho, jestli jsou plán. Ty, které plán nejsou, se zobrazují v `UserSelectActivityView`. Naopak aktivity typu plán se zobrazují v `UserPlanView`. Kromě možnosti vybrat aktivitu typu plán zobrazuje i již vytvořené budoucí plány, které je možné smazat. Zároveň se provádí kontrola, zda přihlášený uživatel má ze skupiny dostupné aktivity typu plán a přístup do pohledu s plány.

6.2.2.15 Krok 33-37

Do těchto kroků aplikace pracovala pouze s daty vytvořenými v `MainViewModel` při zapnutí aplikace a změny dat se ukládaly do patřičných skladišť (stores). Po vypnutí a znovu zapnutí aplikace docházka ztratila všechny vytvořená a upravená data uživatelem a znova se jen vytvořila předchystaná data z `MainViewModel`. Proto je nutné přidání databáze, kde by se data uchovávala. Pro jednoduchost instalace a použití bylo zvoleno SQLite.

Při spuštění aplikace je nejprve zkontrolováno, zda existuje soubor appsetting.json ve složce, odkud se aplikace spouští. Pokud neexistuje, soubor se vytvoří a vytvoří se v něm i prázdná cesta k databázi. Následně se zkontroluje cesta k databázi, pokud neexistuje, uživatel je vyzván, aby vybral složku, kde bude databáze vytvořena nebo soubor s již existující databází. Díky tomu lze snadno připojit více aplikací k jedné databázi a pracovat se stejnými daty. Nakonec je zkontrolováno, zda databáze obsahuje pro určité modely data, a pokud ne, tak se vytvoří nová – uživatelé včetně admina, skupiny, aktivity, globální nastavení aktivit, závazky. Poté je aplikace připravena a spustí se. Při druhém spuštění se zkontroluje existence

appsetting.json, z něj se zjistí cesta k databázi, ověří se existence databáze, ověří se existence dat a aplikace se spustí.

Všechny přístupy do databáze jsou v projektu EF. Projekt WPF využívá svých skladišť (stores) a pomocí EF do nich nahrává data nebo zapisuje do databáze nová.

Postupně se předělaly všechny funkce, tak aby využívaly EF. Zároveň je zde využito asynchronismu, to znamená, že všechny funkce v EF jsou asynchronní, aby nebránili běhu UI. Jakmile jsou data načtena, skladiště (store), který data načítalo informuje pomocí eventu všechny části, které tomuto eventu naslouchají. S příchozí informací o změně se UI informuje o nových datech a zobrazí je.

6.2.2.16 Krok 38

V tomto kroku byla přidána chybějící funkcionalita do `UsersView`, tak aby vedoucí mohl nahlédnout do jednotlivých dnů a součtů docházky svým podřízeným. Muselo kvůli tomu být předělán pohled `UserHistoryView` a vzniknout nový pohled `UserHistoryOverviewView`, aby nevznikal zbytečně duplicitní kód a `UserHistoryView` se dal využít na více místech.

6.2.2.17 Krok 39

Aby existovala informace o tom, kteří uživatelé pracují, mají přestávku nebo ještě nepřišli do práce, vznikl nový modální pohled `UsersCurrentActivityView` dostupný všem uživatelům z `HomeView` tlačítkem Současná aktivita uživatelů. Pohled zobrazuje všechny uživatele a jejich současnou aktivitu a informaci, kdy tato aktivita začala. Zároveň obsahuje možnost vyhledávání podle jména nebo příjmení nebo kombinace.

Do teď mohl zjistit současnou aktivitu uživatelů pouze nadřízený jen u svých podřízených, a to přes Podřízení a jednotlivě po uživatelích zjišťovat jakou mají dnešní docházku.

6.2.2.18 Krok 40-42

Poslední funkcionalita je přidání možnosti exportovat součty aktivit podle zvolých názvů slučování zadaných ve vlastnostech aktivit. Generovat export může každý uživatel, vedoucí navíc může generovat export i pro všechny své podřízené. Export je ve formátu csv. Vygenerovaný soubor obsahuje jméno uživatele, datum a aktivity podle názvu slučování. Pro generované uživatele jsou využity data z `AttendaceTotal`, kde jsou součty aktivit pro dny spočítány, jednotlivé aktivity převedeny na jejich sloučené názvy a případně sečteny.

6.2.2.19 Krok 42 a další

V posledních krocích jsou už pouze opravy nalezených chyb.

6.2.3 Shrnutí vývoje

Během vývoje byla neustále opravována a vylepšována vizuální část aplikace. Komponenty a styly, které se využívaly opakovaně, jsou `Common.xaml`. Ten obsahuje styly tlačítek, rámečků, seznamů a dalších využitých komponent.

Ve vývoji je taky zahrnuto několik převodníků (Converters), které často slouží pro zobrazení více hodnot nebo pro převod booleovské hodnoty na vizuální – například pro skrývání.

Celý vývoj ve zmíněných krocích je zaznamenán na github:

https://github.com/tsev26/AttendanceApp

Vygenerovaná aplikace lze přímo z github stáhnout, a to ve dvou verzích:

- v1.0.1win-x64 – Self-contained, win-x64 (obsahuje i nutný framework ke spuštění vygenerovaný pro win-x64)
- v1.0.1portable – Framework-dependant, Portable (předpokládá, že framework je na počítači nainstalovaný nebo odkáže na jeho instalaci)

6.3 Představení

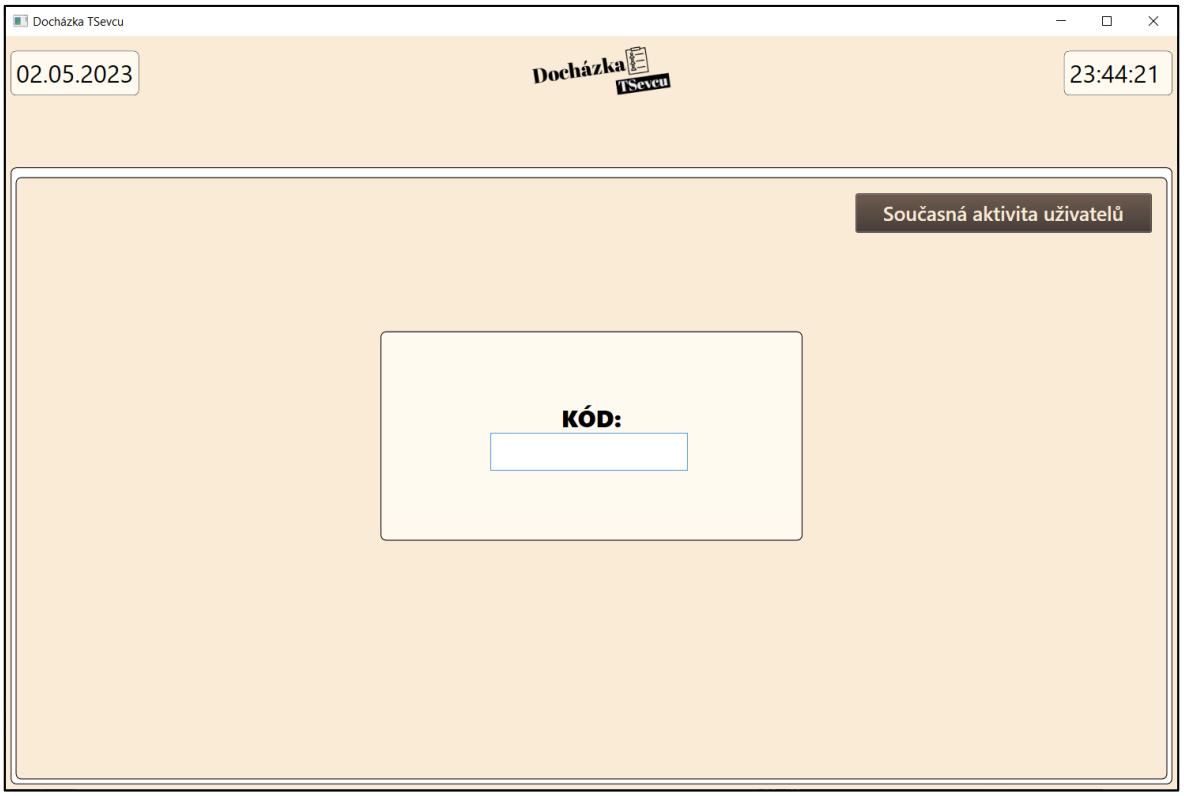
Celá aplikace je naprogramována, aby byla responzivní. Po přihlášení je aplikace rozdělená ve všech pohledech do tří částí. Horní panel, který obsahuje datum, logo, čas, navigační menu s tlačítky a jméno se současnou aktivitou přihlášeného uživatele. A hlavní část na levou a pravou stranu, které jsou odděleny rámečky, které se mění podle velikost a roztáhnutí okna aplikace.

6.3.1 První spuštění

Při prvním spuštění je nutné aplikaci určit, kde bude nebo již je uložená databáze. Aplikace při spuštění kontroluje existenci souboru appsettings.json, který když neexistuje, tak ho aplikace vytvoří. V appsettings.json se nachází cesta k databázi. Aplikace zkontroluje, zda cesta k souboru databázi existuje. Pokud neexistuje uživatel je nucen vybrat složku, kde se nová databáze vytvoří, nebo soubor s již existující databází. Díky tomu se může snadno připojit více aplikací ke stejné databázi. Proto je ideální vytvořit první databázi na sdíleném uložišti, aby k ní mělo přístup více počítačů, pokud se aplikace využívá z více míst zároveň. Soubor appsettings.json se vytváří a kontroluje v místě, kde se aplikace nachází, proto je ideální mít aplikaci ve složce a pouze si k ní vytvořit zástupce.

Po prvním spuštění je tedy vytvořen soubor appsettings.json a databáze. Následně je provedena kontrola v databázi na výskyt aktivit, globálního nastavení aktivit, uživatelů a skupin. Pokud se v databázi žádná data nevyskytují, tak jsou vytvořena testovací data. Aplikace vytvoří 5 aktivit – práce, pauza, domov, služební cesta a dovolená, kde služební cesta a dovolená jsou aktivity typu plán. Globální nastavení aktivit je nastaveno tak, že hlavní pracovní aktivita je práce, hlavní nepracovní aktivita je domov, hlavní aktivita pro pauzu je pauza, nutná přestávka je po 6 hodinách a délka nutné přestávky je 30 minut. Skupiny jsou vytvořeny dvě, a to vedení a základní. Nakonec se vytvoří 5 uživatelů, kde hlavní z nich je uživatel admin, s klíčem pro přihlášení AAA, který si po přihlášení může změnit. Kontrola na výskyt dat pro jednotlivé modely je prováděna vždy při spuštění, protože pokud by aplikace neobsahovala například žádné uživatele typu admin, tak by neexistoval uživatel, který by mohl měnit všechno nastavení.

Po těchto krocích a kontrolách je aplikace spuštěna a zobrazí domovský pohled. Aplikace vždy v horní liště zobrazuje současné datum a čas. Domovský pohled očekává klíč od uživatele pro přihlášení, navíc nabízí zobrazení současných aktivit všech uživatelů.



Obrázek 39 Docházková aplikace: Domovský pohled

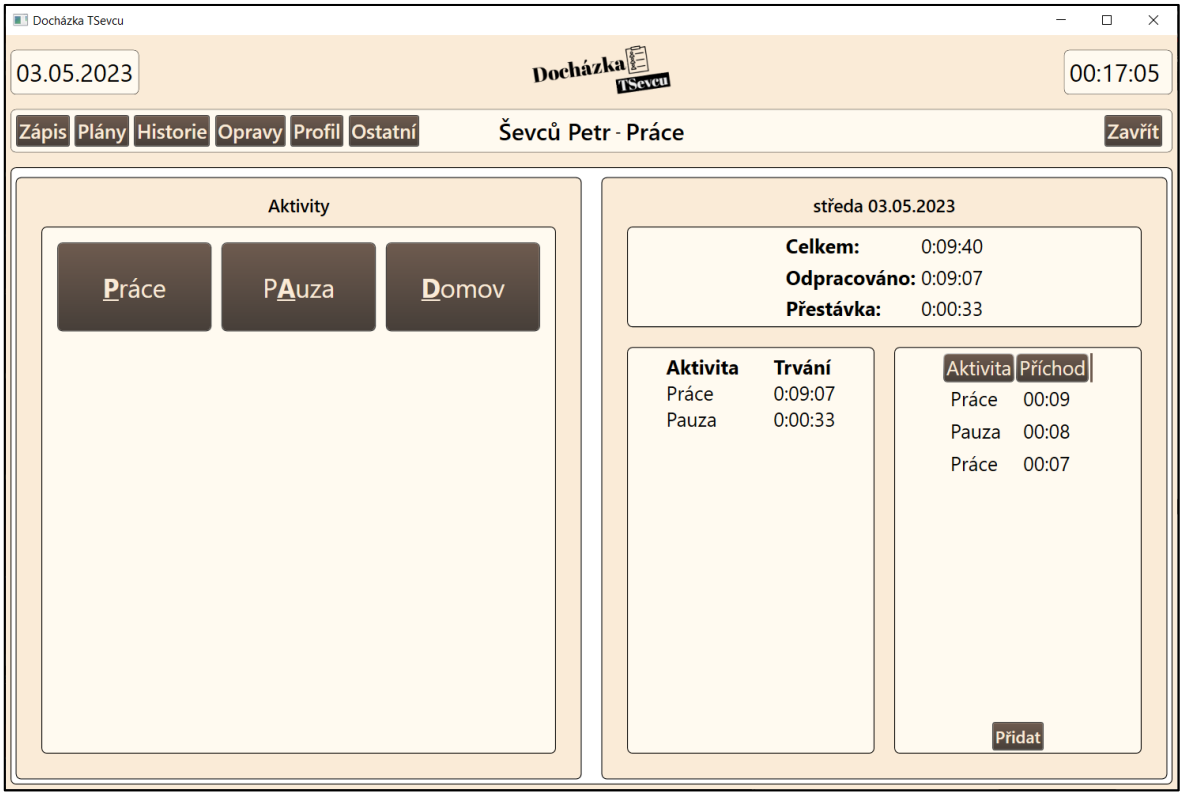
6.3.2 Pohled uživatele

Základní uživatel není admin a není ani vedoucím žádné ze skupin. Má proto možnosti, které se týkají pouze jeho docházky, a to zápis aktivit, zápis a správa plánů, zobrazení historie, vytvoření oprav zápisů aktivit do docházky, zobrazení a oprava vlastního uživatelského profilu, správa klíčů, nastavení rychlého zápisu a možnost exportu vlastní docházky.

6.3.2.1 Zápis aktivit

Hlavní funkce docházky je zápis aktivity. Pohled zápisu aktivit se zobrazuje vždy po přihlášení. Obsahuje seznam aktivit, které má uživatel dostupné díky zařazení ve skupině a přehled současného dne. Ten zobrazuje celkově strávenou dobu aktivit, které se počítají, celkově odpracovanou dobu a dobu strávenou přestávkou. Dále zobrazuje součty jednotlivých aktivit a samostatné zápisy aktivit do docházky s časem zápisu. Aplikace po prvním spuštění obsahuje aktivity práce, pauza a domov. Uživatel si může vybrat jednu z nabízených aktivit, a to buď vybráním anebo pomocí klávesnice a zapsání zvýrazněného písmena v názvu aktivity. Po výběru je uživateli aktivita zapsána do docházky a aplikace se přesměruje zpět do domovského pohledu, kde zobrazí informace o zápisu.

Kromě výběru aktivity má uživatel v tomto okně možnost upravit nebo smazat existující zápis v docházce nebo přidat nový. Tím se vytvoří oprava docházky, která musí být následně schválena vedoucím nebo adminem, aby se v docházce promítla.



Obrázek 40 Docházková aplikace: Zápis aktivit

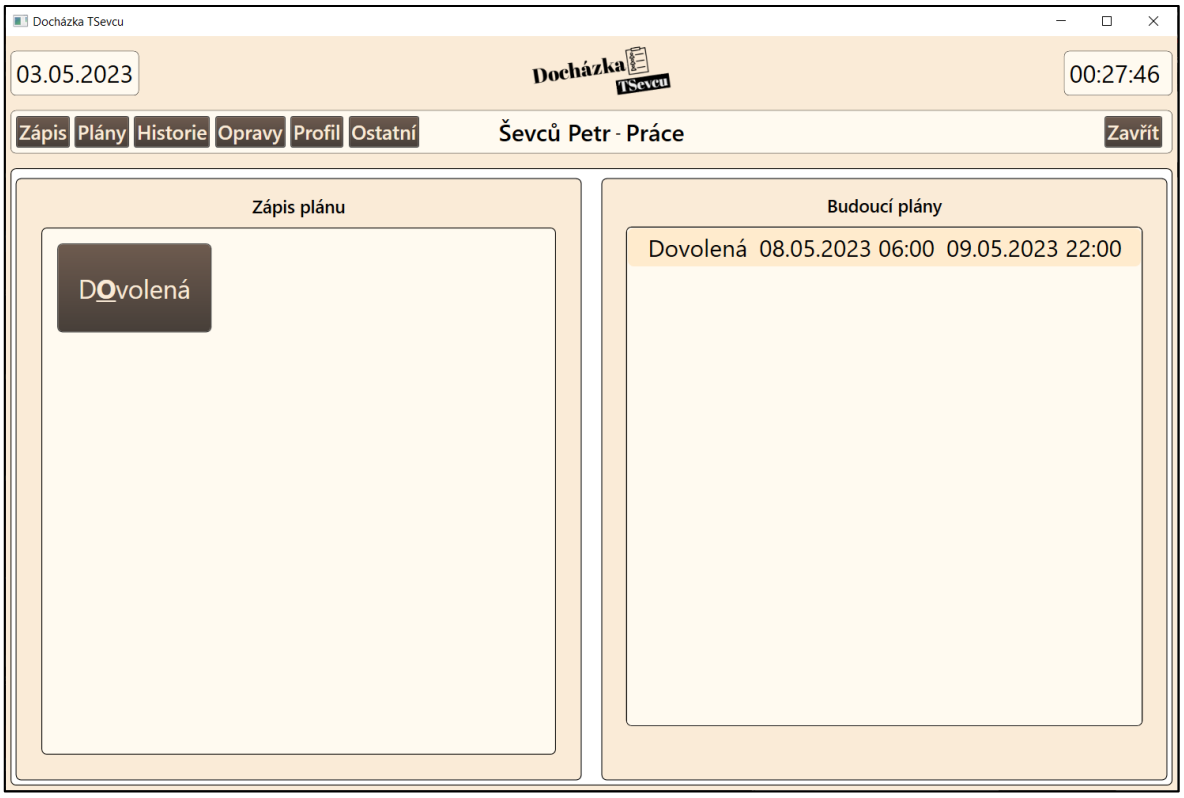
6.3.2.2 Plány

Zápis plánů je obdoba zápisu aktivit, protože plány jsou aktivity s vlastností plán. Ne všichni uživatelé musí mít vůbec přístup do plánů. Je to ovlivněno nastavením aktivit a plánů pro skupiny, ve které se uživatel nachází. Pokud skupina nemá žádný plán přiřazen, uživateli ani do plánů přístup nemá a tlačítko je pro něj skryto.

Pohled plánů obsahuje seznam možných plánů a seznam budoucích, již dříve vytvořených, plánů. Výběr plánů funguje stejně jako aktivit, uživatel plán vybere nebo zadá zkratku, která je zvýrazněná v jeho názvu. Na rozdíl od aktivit, musí následně zadat začátek a konec plánu. Plány jsou dvojího typu:

- Zadává se i čas, například služební cesta.
- Zadává se pouze datum, a navíc je možné vybrat, zda se jedná o celodenní nebo půldenní plán, například dovolená.

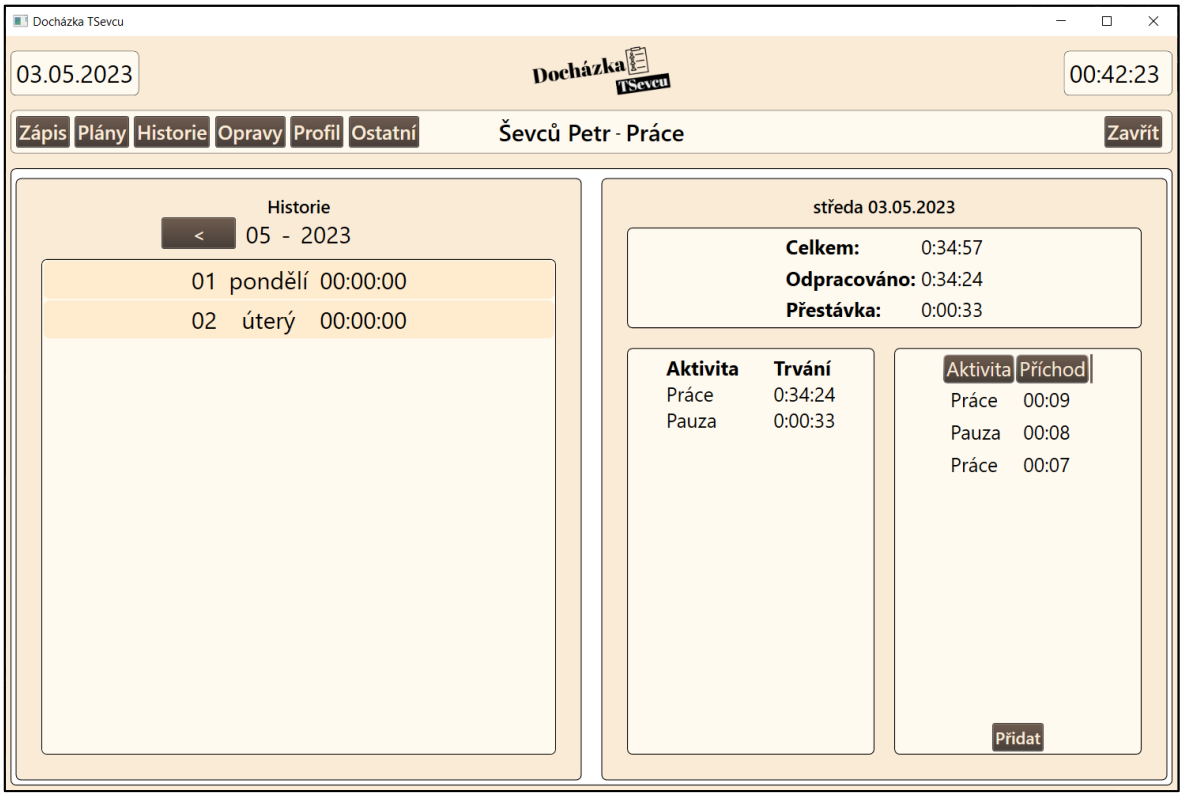
Pokud je zadané datum, případně čas, stejné jako současné, tak se aplikace přesměruje zpět do domovského pohledu a informuje o vytvoření a zapsání plánu. Pokud se jedná o plán, který bude probíhat v budoucnosti, tak se pouze vytvoří nový plán, který se zobrazí v seznamu budoucích plánů. Budoucí plány lze vybrat a smazat.



Obrázek 41 Docházková aplikace: Plány

6.3.2.3 Historie

Pohled historie zobrazuje jednotlivé dny v měsíci a přehled vybraného dne. Uživatel může přecházet mezi měsíci pomocí šipek a vybírat jednotlivé dny pro zobrazení přehledu vybraného dne. Seznam dnů v měsíci zobrazují den v měsíci, den v týdnu a trvání odpracované doby. Pokud žádný den není vybraný, zobrazuje se současný den. Stejně jako v pohledu zápisu aktivit, lze i zde upravovat a mazat existující zápisy ve vybraný den nebo přidávat nové.



Obrázek 42 Docházková aplikace: Historie

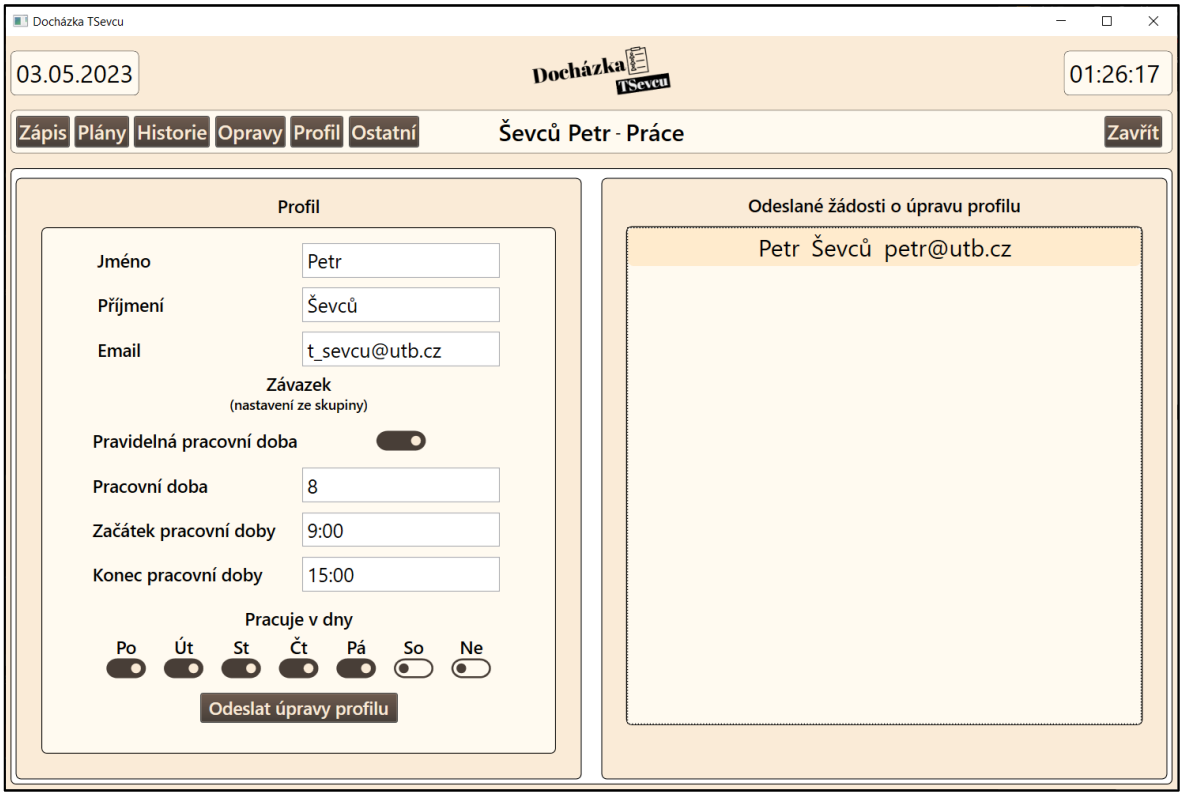
6.3.2.4 Opravy

Pohled opravy zobrazuje seznam všech žádostí o úpravu docházky a seznam všech zápisů do docházky. Žádosti jsou typu úprava, přidání a odebrání a může být ve stavu čeká, schváleno nebo zamítnuto. Ze seznamu zápisů do docházky lze vytvářen další opravy.

Obrázek 43 Docházková aplikace: Opravy

6.3.2.5 Profil

Pohled profil zobrazuje profil uživatele, jeho závazek a odeslané žádosti o úpravu profilu. Každý uživatel má možnost si upravit vlastní profil. Úpravy nejsou přímo promítnuty, ale jsou pouze vytvořeny žádosti o úpravu profilu, kterou musí schválit vedoucí nebo admin, stejně jako opravy docházky.



Obrázek 44 Docházková aplikace: Profil

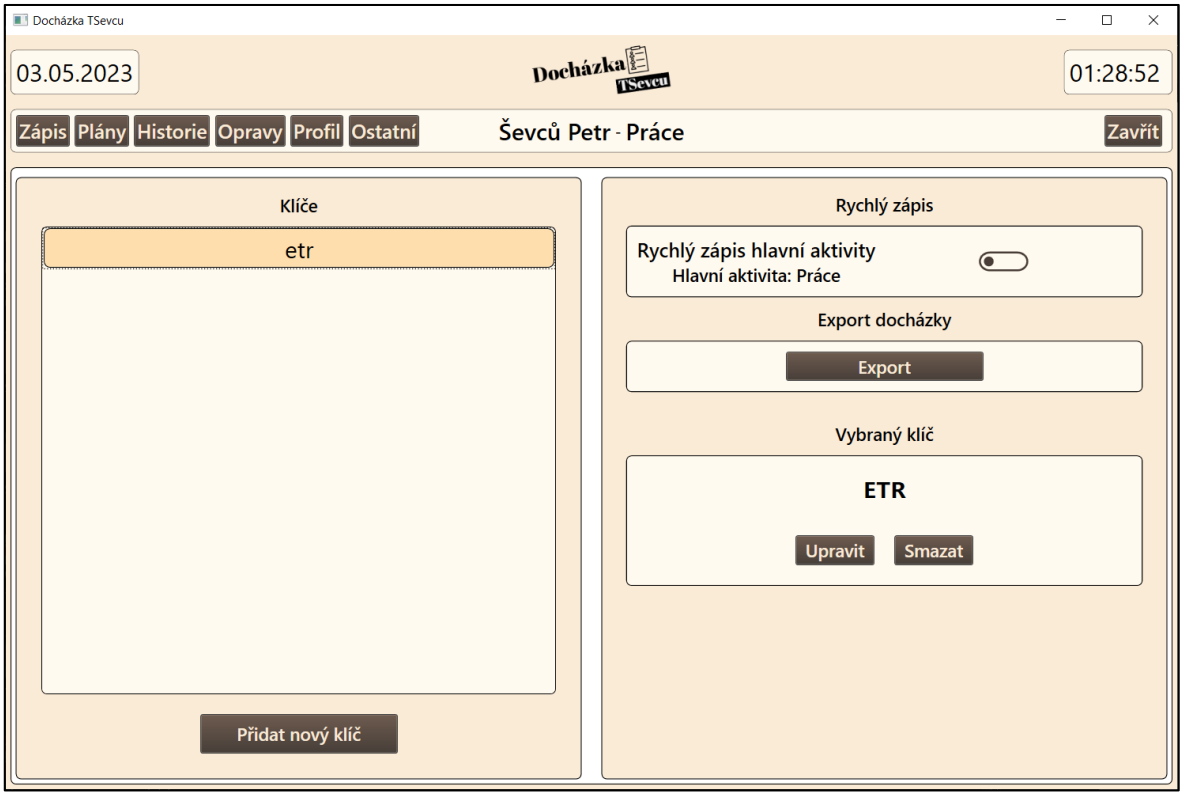
6.3.2.6 Ostatní

Tlačítko Ostatní vede na pohled se správou klíčů uživatele, možností nastavit rychlý zápis hlavní pracovní aktivity a umožňuje export docházky.

Uživatel může mít více klíčů. Například při použití zařízení čtečky pro RFID klíče nebo NFC klíče, může mít uživatel klíč pro každé zařízení jiné.

Rychlý zápis hlavní aktivity slouží pro rychlé zapsání aktivity, která je v globálním nastavení aktivit nastavená jako hlavní pracovní aktivita. Pokud ji má uživatel zapnutou, tak se po přihlášení zkontroluje jeho současná aktivita, pokud se jedná o aktivitu, která má vlastnost nepočítat nebo je typu pauza, uživateli se automaticky zapíše hlavní pracovní aktivita a aplikace se přesměruje zpět na domovský pohled, kde informuje o zápisu aktivity.

Při výběru export, aplikace otevře modální okno, kde uživatel vybere měsíc, který chce exportovat a následně vybere složku, kam chce data exportovat. Exportuje se csv soubor s jménem uživatele, datem a názvy jednotlivých aktivit, pro které je spočtena suma hodin v jednotlivé dny.



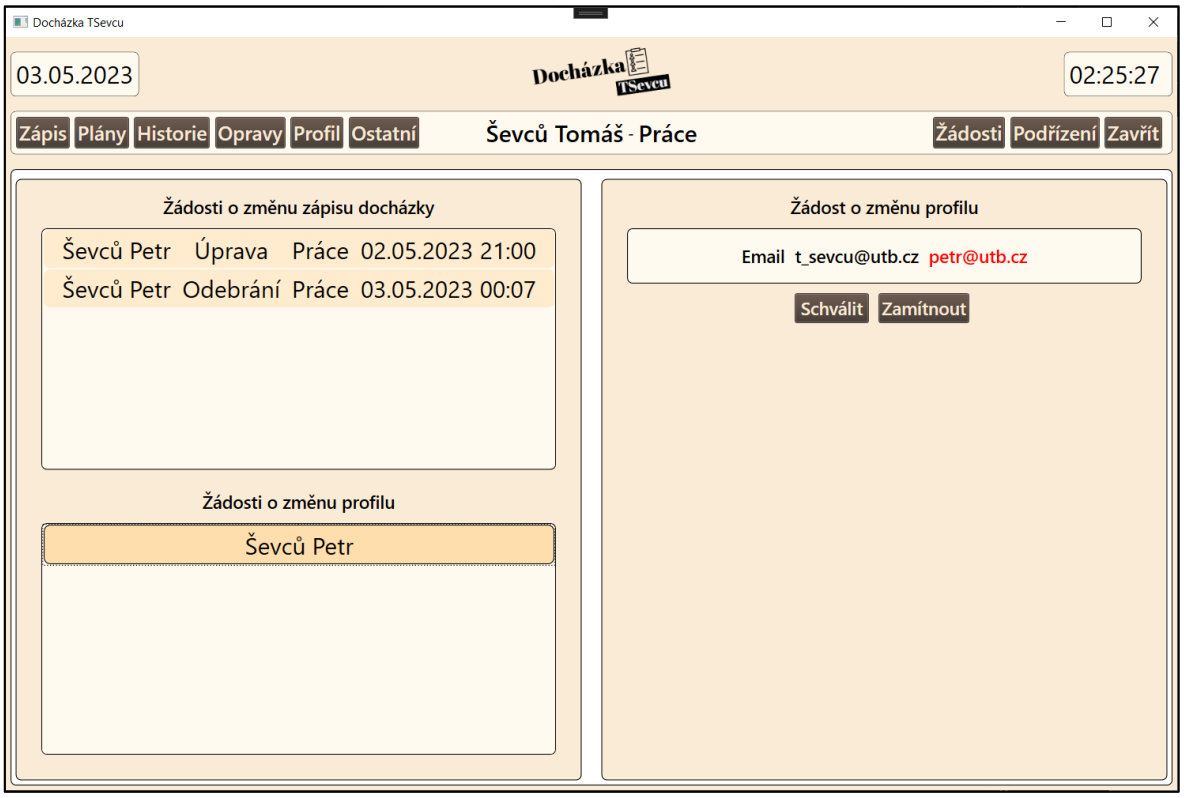
Obrázek 45 Docházková aplikace: Ostatní

6.3.3 Pohled vedoucího

Vedoucí má všechny možnosti jako běžný uživatel. Navíc může schvalovat žádosti o změnu zápisu docházky a žádosti o změnu profilu. A má přehled o všech svých podřízených uživatelích, u kterých může upravovat jejich profil, klíče, opravovat docházku a exportovat jejich docházku.

6.3.3.1 Žádosti

V pohledu žádosti vidí vedoucí všechny nevyřízené žádosti rozdělené na seznam žádostí o změnu zápisu do docházky a seznam žádostí o změnu profilu. Po vybrání některé z žádostí se zobrazí, co se mění a upravuje. Červeně nová hodnota, černě předešlá hodnota. U každé žádosti má možnost ji schválit nebo zamítnout.

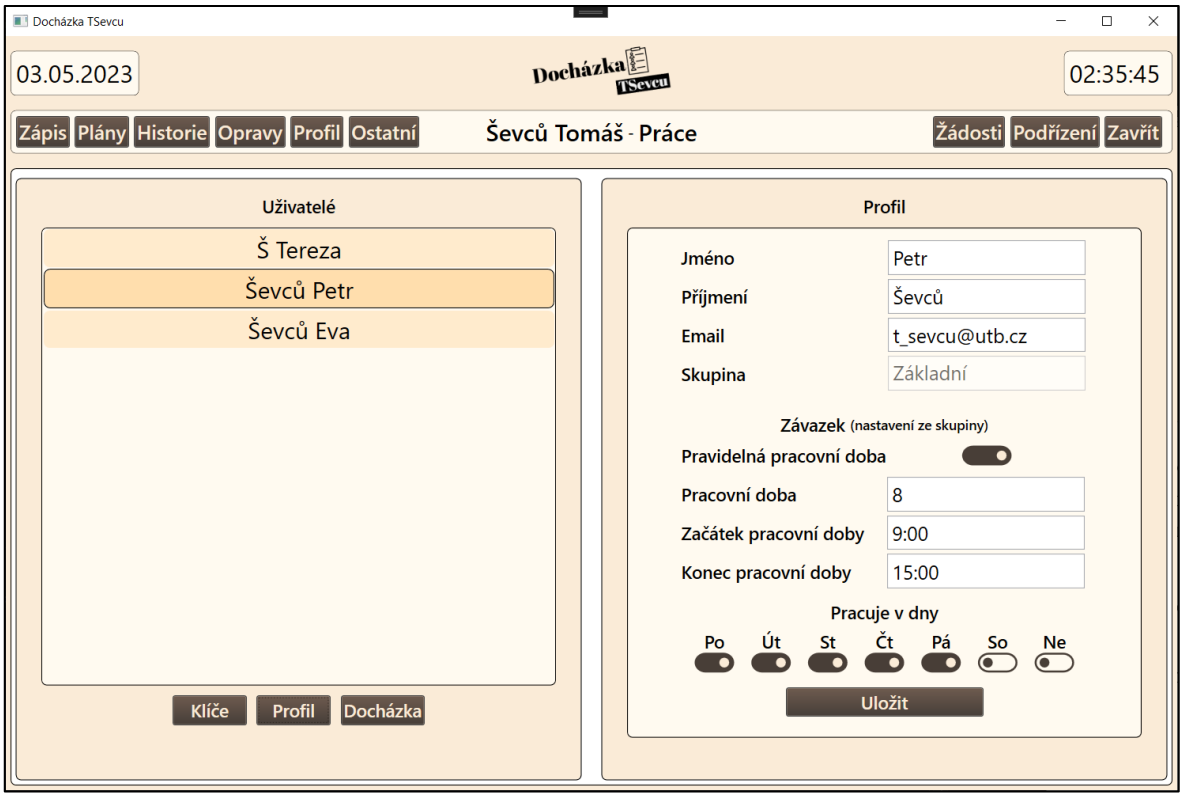


Obrázek 46 Docházková aplikace: Žádosti

6.3.3.2 Podřízení

Pohled podřízení zobrazuje všechny uživatele, kteří jsou ve skupinách, kde je právě přihlášený uživatel nastaven jako vedoucí.

U každého uživatele lze zobrazit a upravit jeho profil, zobrazit klíče, upravovat je a přidat nové a zobrazit docházku, podobně jako v pohledu historie.



Obrázek 47 Docházková aplikace: Podřízení

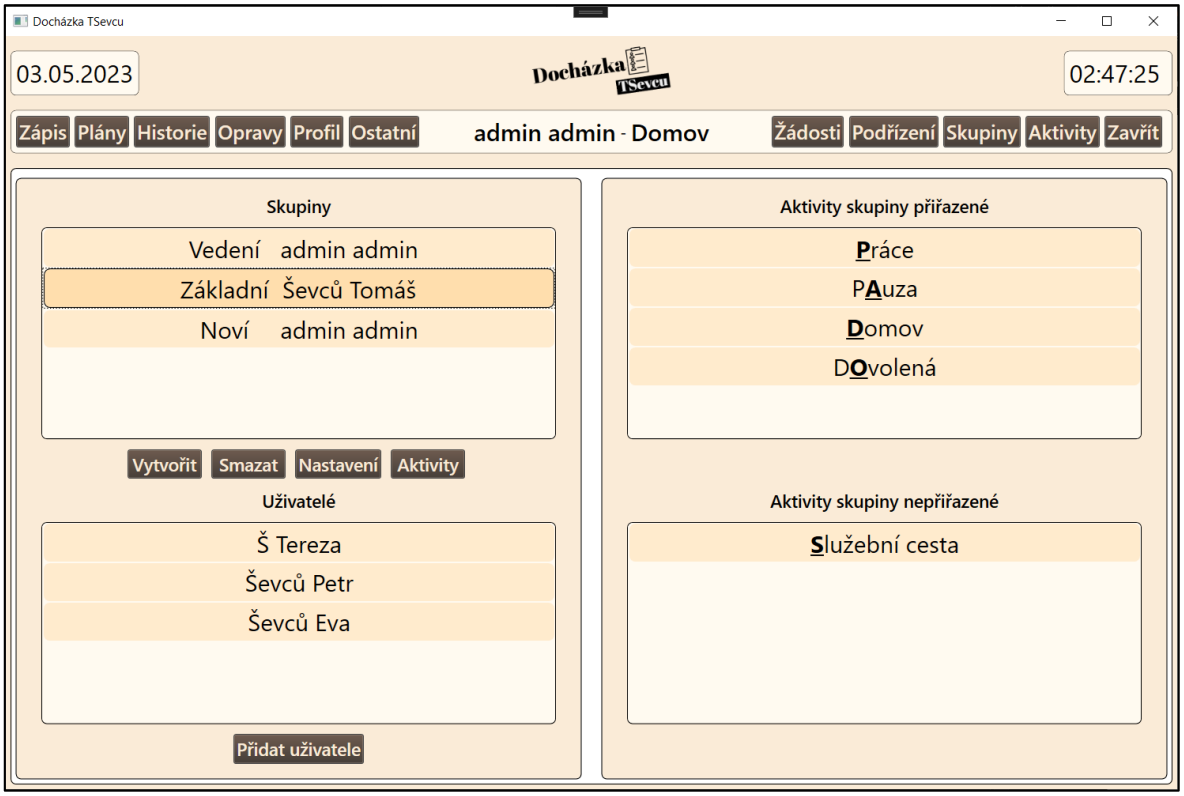
6.3.4 Pohled admina

Admin má všechny možnosti jako běžný uživatel, zároveň je brán jako vedoucí všech uživatelů, takže vidí a má správu nad všemi uživateli. Další jeho možnosti jsou vytváření nových uživatelů v pohledu podřízení, správa skupin a správa aktivit.

6.3.4.1 Skupiny

Pohled skupiny zobrazuje seznam všech skupin. Po vybrání skupiny se zobrazí v seznamu uživatelé, kteří do skupiny patří. Skupině lze nastavovat vedoucí, závazek a aktivity. Uživatelé ve skupině pak mají defaultně nastavený závazek, které má skupina a mohou vybírat pouze aktivity, které jsou skupině přiřazeny.

Skupiny lze vytvářet nové a existující mazat. Vytvoření nové skupiny otevře modální okno, kde se zadává pouze název. Vytvořená skupina má nastaveného vedoucího vždy admina, který ji vytvářel. Nastavení, přiřazení aktivit a změna vedoucího se nastavuje po vytvoření. Při smazání existující skupiny, ve které jsou uživatelé, se všichni uživatelé přiřadí do nové skupiny Nezařazení (pokud skupina již existuje, tak se nevytváří a zařadí se přímo do stávající).



Obrázek 48 Docházková aplikace: Skupiny

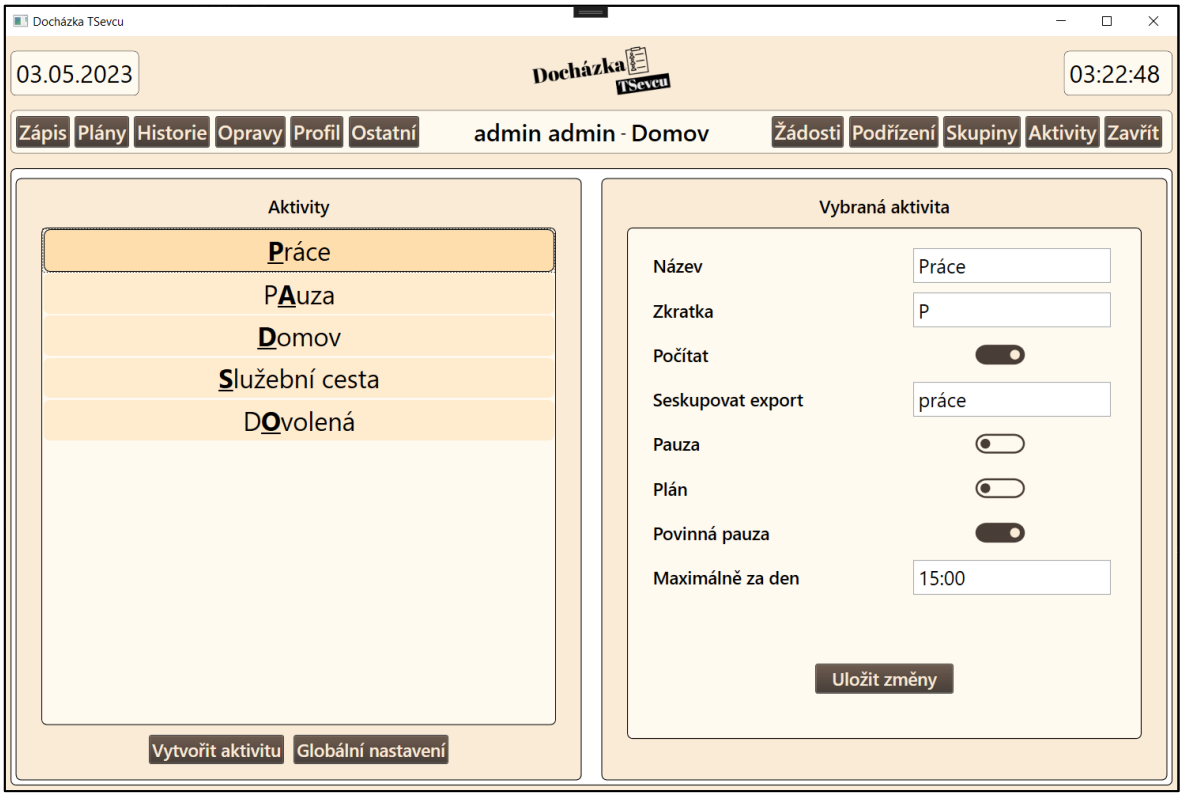
6.3.4.2 Aktivity

Pohled aktivity zobrazuje seznam aktivit, po výběru jedné z aktivit zobrazuje její vlastnosti aktivity. Navíc obsahuje zobrazení pro nastavení globálních aktivit. U aktivit se nastavuje:

- Název
- Zkratka (pro zapsání z klávesnice)
- Počítat – zda se má aktivita počítat. Aktivity, které se počítají mají v přehledu sumu, například domov se nepočítá, ale ostatní aktivity ano
- Seskupovat export – název podle, kterého se aktivity seskupují do exportu, například můžu mít aktivity práce, sklad, účtárna a hlavní budova, kde každý zaměstnanec si

zapisuje na svém pracovišti aktivitu práce, ale pokud by se přesunul na jiné pracoviště, například z hlavní budovy do skladu, narazil by si sklad, aby byla informace, že není na své pozici, ale pro export by se všechny tyto aktivity seskupovaly pod názvem práce

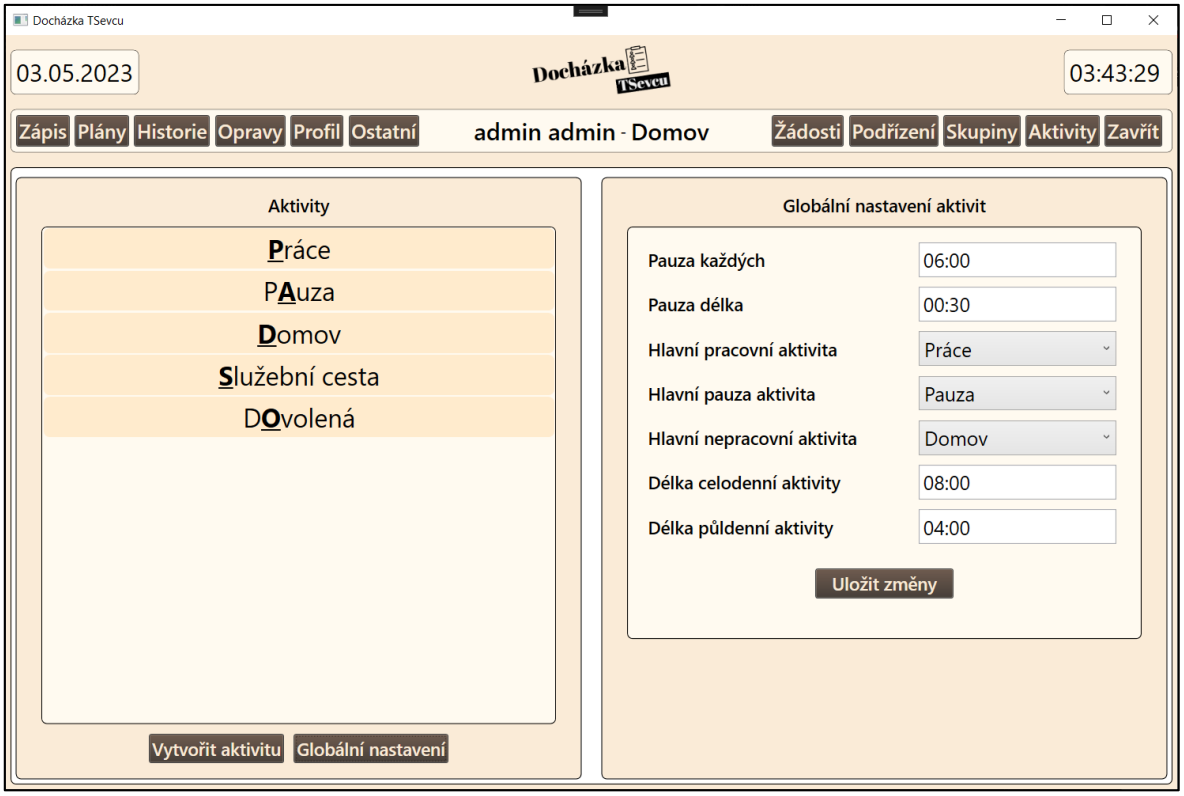
- Pauza – určuje, zda se jedná o pracovní aktivitu nebo nepracovní aktivitu, tedy pauzu
- Plán – rozděluje aktivity na plány a na běžné aktivity, kde u plánů se zapisuje začátek a konec
- Povinná pauza – u aktivit, které mají nastavenou povinnou pauzu, se kontroluje, zda během dne mají požadovanou přestávku za určitou odpracovanou dobu z globálního nastavení aktivit
- Maximálně za den – pokud by se suma trvání aktivity za den překročila hodnotu zadanou v tomto poli, tak se nastaví tato hodnota
- Zadává se čas – nastavuje se pouze u plánů a rozděluje plány na ty, kde se zadává čas plánu, například služební cesta a plány, kde se nezadává čas, ale určuje se zda, se jedná o celodenní nebo půldenní plán, například dovolená



Obrázek 49 Docházková aplikace: Aktivity

V globálním nastavení aktivit se nastavuje:

- Pauza každých – určuje po jak dlouhé době u aktivit, které mají vlastnost Povinná pauza, je vyžadována pauza
- Pauza délka – určuje, jak je dlouhá povinná pauza
- Hlavní pracovní aktivita – slouží pro rychlý zápis a zapisuje se při předčasném ukončení plánu
- Hlavní pauza aktivita – zapisuje se, pokud nebyla vybrána pauza aktivitami během dne a byl překročen časový úsek Pauza každých
- Hlavní nepracovní aktivita – zapisuje se po skončení plánu
- Délka celodenní aktivity – u plánů, které nemají vlastnost Zadává se čas, určuje délku při nastavení celodenní
- Délka půldenní aktivity – u plánů, které nemají vlastnost Zadává se čas, určuje délku při nastavení půldenní



Obrázek 50 Docházková aplikace: Globální nastavení aktivit

7 ZHODNOCENÍ A MOŽNOSTI ROZVOJE

Vytvoření jakékoli nové aplikace je často náročný proces vývoje plný nečekaných zvratů. Navzdory pečlivému plánování a detailním požadavkům se může konečný produkt lišit od původních plánů nepředvídatelnými způsoby. Proto je klíčové zhodnotit výslednou aplikaci a prozkoumat možnosti rozvoje, aby bylo zajištěno, že výsledek splňuje zamýšlené cíle a požadavky.

7.1 Výsledný diagram tříd

Výsledný diagram tříd s mírnými rozdíly odpovídá plánovanému doménovému modelu Obrázek 38 Plánovaný . Rozdíly mezi nimi jsou zapříčiněny optimalizací a změnami při vývoji.

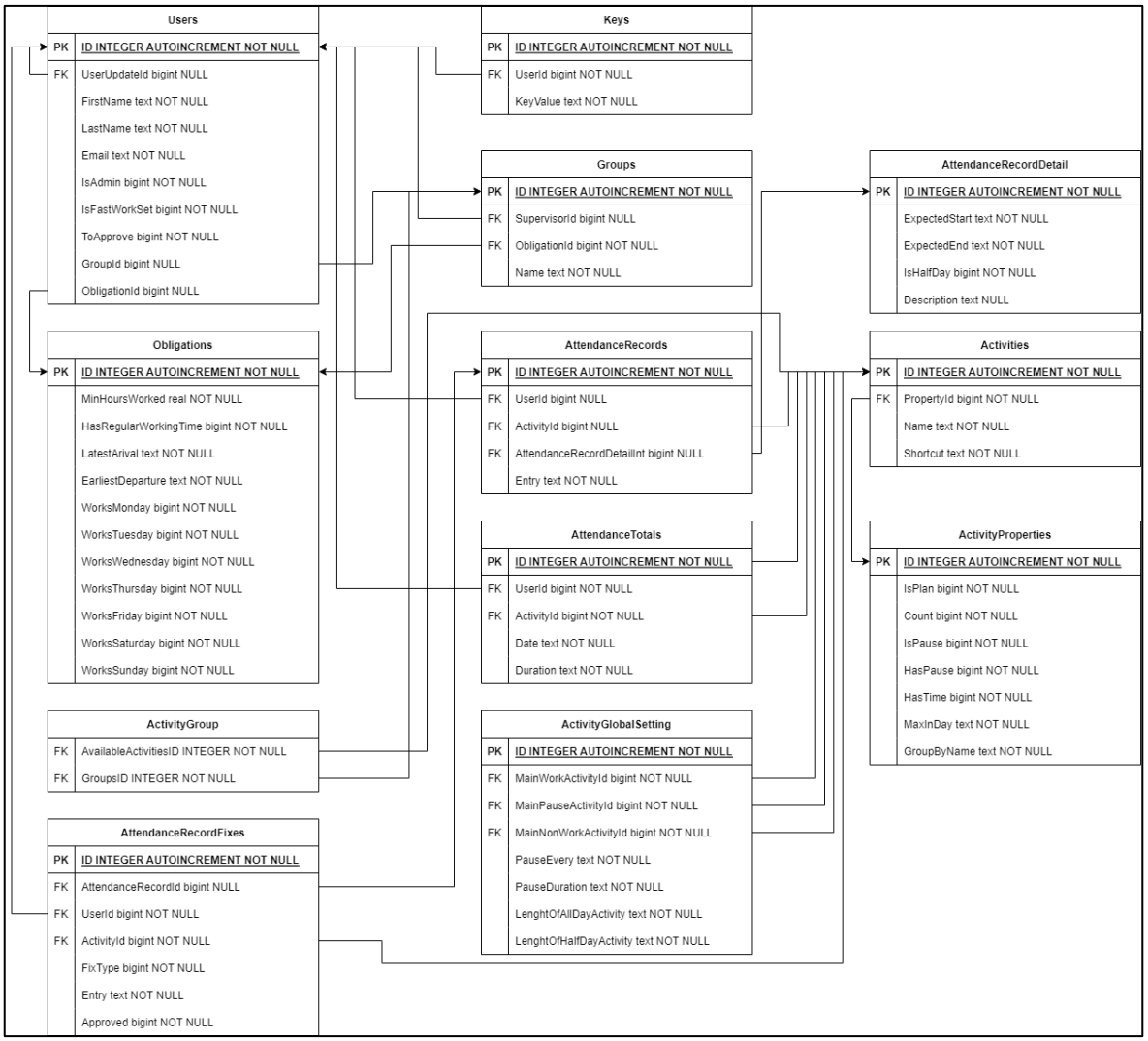
Plánovaná třída `Oprava profilu` je ve výsledné aplikaci přímo zahrnuta ve třídě `User` (Uživatel), kdy při změně profilu je vytvořen nový uživatel s atributem `ToApprove` a odkazem `UserUpdateId` na originálního uživatele, žádající o změnu profilu.

Podobně je tomu u plánované třídy `Plánování aktivit`. Ve výsledném diagramu tříd je nahrazena `AttendanceRecordDetail`, která pouze rozšiřuje zápis docházky `AttendanceRecords`. Díky tomu není potřeba pravidelně kontrolovat plánované aktivity na výskyt plánu, který by měl začátek s konec mezi současným časem a následně zapisovat do zápisů docházky. Ve výsledné docházce se plán zapíše přímo do zápisů docházky s budoucím nastaveným datem začátku.

Dalším rozdílem mezi plánovanými a výslednými diagramy tříd je třída `ActivityGlobalSetting`, která obsahuje vlastnosti související s nastavením aktivit a trvání a pauz.

Kromě uvedených rozdílů existují odchylky v atributech obsažených v jednotlivých třídách. Tyto rozdíly mohou zahrnovat přidání nových atributů nebo odstranění stávajících za účelem vylepšení funkčnosti systému nebo řešení specifické funkcionality.

Přes tyto rozdíly zůstává výsledný diagram tříd v souladu s plánovaným diagramem tříd. Základní struktura a vztahy mezi třídami zůstaly nedotčeny, pouze s drobnými úpravami tříd a jejich atributů.



Obrázek 51 Výsledný diagram tříd

7.2 Splnění požadavků

Splnění plánovaných požadavků je klíčové pro zajištění, aby koncový produkt vyhovoval potřebám koncových uživatelů.

První požadavkem bylo ověření/autentizace uživatele. Docházková aplikace vyžaduje, aby uživatelé ověřili svou identitu prostřednictví přihlašovacího procesu. Tento proces ověřování je kvůli jednoduchosti a rychlosti přihlášení autentizace tvořen pouze klíčem uživatele. Po úspěšném zadání klíče, kde klíč může být i hodnota z přečteného čipu, karty nebo telefonu, je uživatel přihlášen do své docházky.

Druhým požadavkem byla správa uživatelů. Docházková aplikace poskytuje funkce správy uživatelů a umožňuje správcům vytvářet a spravovat uživatelské účty. Navíc jsou uživatelé

rozčleněni do skupin, kde každá skupina má svého vedoucího, který může spravovat účty svých podřízených. Uživatelé mohou také aktualizovat své profily a spravovat záznamy o docházce.

Důležitým požadavkem bylo také sledování času. Aplikace docházky zobrazuje v horní liště současné datum a čas a ten se pak i zapisuje při vybraní aktivity. Pro jednotlivé aktivity jsou počítány sumy trvání pro každý den a každého uživatele, podle kterých lze následně zjistit, jak dlouhou dobu uživatel v práci strávil.

Jedním z hlavních požadavků byla správa aktivit. Uživatel typu admin může v aplikaci docházky vytvářet a spravovat aktivity. U jednotlivých aktivit lze nastavovat jejich vlastnosti, podle kterých se pak započítávají do celkového času odpracované doby. Aktivity a jejich vlastnosti obsahují i informace podle kterých se aktivity exportují nebo kolik se maximálně aktivity za den započítá.

Dalším významným požadavkem plánované aplikace byl managment absencí a plánů. Docházková aplikace poskytuje uživatelům vytvářet plány, což jsou speciální aktivity, u kterých se zadává začátek a konec. Díky tomu si mohou uživatelé naplánovat aktivity do budoucna.

Plánovaným požadavkem byla také funkce reportování. Docházková aplikace generuje exporty typu csv a obsahuje data pro uživatele o délce aktivit pro vybraný měsíc.

Nakonec plánovaná aplikace vyžadovala funkci zadávání dat a ověřování. Docházková aplikace poskytuje uživatelům možnost zadávat a ověřovat docházkové údaje, čímž zajišťuje, že údaje zadávané do systému jsou přesné a aktuální.

Vytvořená docházková aplikace úspěšně splňuje plánované požadavky. Aplikace poskytuje ověřování/autentizaci uživatelů, správu uživatelů, sledování času, správu aktivit, správu nepřítomností, aktualizace v reálném čase, reportování a funkce zadávání a ověřování dat.

7.3 Rozvoj a omezení

Vývoj vytvořené docházkové aplikace byl zaměřen na poskytnutí řešení správy docházky uživatelům. Stále však existují oblasti, kde může být nezbytný další vývoj. Kromě toho má aplikace omezení, která je třeba vzít v úvahu.

Jednou z oblastí, kde lze uvažovat o dalším rozvoji, je závazek, který se zadává pro skupinu nebo pro uživatele. V současné verzi aplikace je tento údaj pouze informativní a není součástí výpočtu údajů o docházce. S dalších vývojem by však tyto informace mohly být použity ke zvýšení přesnosti údajů o docházce a poskytování podrobnějších zpráv o docházce uživateli.

Další oblasti, kde by mohl být nutný další vývoj, by mohly být odhaleny při nasazení docházkové aplikace do reálného provozu. Tím by se identifikovaly nedostatky, se kterými se při vývoji původně počítalo. Tyto nedostatky by mohly být odstraněny dalším vývojem, zlepšením celkové funkčnosti a použitelnosti aplikace.

Vytvořená docházková aplikace má svá omezení. Jedno z omezení souvisí s technologií použitou pro vývoj aplikace. Aplikace je vyvinuta pomocí WPF, což znamená, že ji lze spustit pouze na operačních systémech Windows. Toto omezení by mohlo potenciálně omezit počet uživatelů, kteří mohou přistupovat k aplikaci, protože ne všichni uživatelé mohou používat operační systém Windows.

7.4 Využití

Vytvořená docházková aplikace poskytuje spolehlivé a funkční řešení pro správu docházkových dat a může tak být ideální volbou pro mnoho firem. Aplikace je všestranná a lze ji využít v různých nastaveních, takže je vhodná pro podniky všech velikostí.

Jedním z možných případů použití aplikace je mít u vchodu do budovy počítač s dotykovým displejem a připojenou klávesnicí. Tento počítač by mohl mít také připojenou čtečku čipů, karet nebo telefonů, která zaměstnancům usnadní přihlášení. Docházková aplikace na tomto počítači umožní zaměstnancům evidovat docházku při vstupu do budovy. To by zaměstnancům poskytlo efektivní a pohodlný způsob sledování jejich docházky.

Aplikace je navržená tak, aby byla uživatelsky přívětivá a snadno ovladatelná, aby byla přístupná všem zaměstnancům. Personalista firmy by mohl být administrátorem aplikace, tak aby mohl vytvářet a spravovat uživatele, aplikaci by měl nainstalovanou i na svém počítači. Vedoucí skupin by také získali přístup k aplikaci ze svých počítačů, což by jim umožnilo spravovat své podřízené a schvalovat opravy docházky a úpravy profilů.

Vytvořená docházková aplikace je k dispozici ke stažení a používání zdarma, takže je ideální volbou pro malé firmy, které nemusí mít finanční prostředky na nákup docházkové aplikace

nabízené společnostmi. Mnoho zaměstnanců v malých firmách stále podává svou docházku na konci měsíce na papíře, kde jsou ručně evidovány odpracované hodiny za každý den. Vytvořená docházková aplikace může pomoci tento proces zefektivnit a poskytnout přesné a spolehlivé údaje o docházce.

Aplikace je však použitelná i pro větší firmy, které mohou mít více zaměstnanců a více vstupů do budovy. I když tyto firmy mají často větší finanční možnosti a mohou mít již docházkovou aplikaci zahrnutou ve svém HR softwaru.

Vytvořená aplikace poskytuje všestranné a uživatelky přívětivé řešení pro správu docházkových dat. Jeho bezplatná dostupnost a snadné použití z ní činí ideálno volbu pro malé společnosti, zatímco její funkčnost a spolehlivost je vhodná i pro větší společnosti. Se svými mnoha funkcemi a možnostmi je vytvořená docházková aplikace vynikající volbou pro všechny podniky, které chtějí zefektivnit své procesy správy docházky.

ZÁVĚR

Vývoj docházkové aplikace vyžaduje použití robustních frameworků a vzorů, které zajistí efektivitu a spolehlivost, jednou z možností je využití WPF se vzorem MVVM. Teoretická část této práce představuje relevantní technologie a koncepty jako .NET, WPF, EF a vzor MVVM, dále se zabývá nejdůležitějšími částmi WPF a MVVM, pomocí kterých je vytvořena aplikační šablona WPF, která obsahuje navigaci, příkazy, služby a skladiště. Praktická část se zaměřuje na vývoj docházkové aplikace, pokrytí jejích požadavků, proces vývoje, prezentace a hodnocení naprogramované aplikace. Prostřednictvím této práce čtenář porozumí WPF frameworku, vzoru MVVM a vývoji docházkové aplikace, kterou lze v budoucnu reálně využít jako docházkový systém ve firmě. Celkově vývoj docházkových aplikací výrazně proměnil sledování docházky z manuálního na automatizované systémy, které jsou mnohem jednodušší a efektivnější. Od vytvořené docházkové aplikace se očekává, že nabídne přesné, spolehlivé a efektivní řešení pro sledování docházky.

V rámci diplomové práce byly všechny body zadání splněny. V kapitole 4.1 se nachází rešerše existujících řešení v oblasti aplikací na docházku. V kapitole 1 jsou popsány technologie použité v rámci vývoje aplikace, včetně frameworku WPF a architektury MVVM. Kapitola 2 se zaměřuje na ukázky postupů a kódu pro klíčové části WPF a MVVM, jako jsou DataBinding, Dependency Injection, příkazy, skladiště, služby, navigace a styly. Aplikační šablona WPF je následně vytvořena v kapitole 3 pomocí výše zmíněných technologií. Kapitola 5 představuje návrhy jednotlivých částí aplikace, pracovní postupy a plány, které zahrnují navržené scénáře využití, diagram aktivit a doménový model docházkové aplikace. V kapitole 6.2 je popsán postupný vývoj a programování v jednotlivých krocích aplikace. Kapitola 6.3 prezentuje vytvořenou aplikaci a její funkce, které jsou představeny podle typu přihlášeného uživatele. Vyhodnocení vytvořené aplikace a její reálné použití jsou poté popsány v kapitole 7.

SEZNAM POUŽITÉ LITERATURY

[1] Microsoft Learn: What is .NET?. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

[2] Microsoft Learn: Overview of .NET Framework. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview

[3] Microsoft Learn: What is ASP.NET?. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet

[4] Microsoft Learn: What is .NET? Introduction and overview. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/introduction

[5] Microsoft: Game development with .NET. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/games

[6] TROELSEN, Andrew a Philip JAPIKSE. Pro C# 7: With .NET and .NET Core. 8th Edition. Apress, 2017. ISBN 978-1484230176.

[7] Microsoft Learn: Desktop Guide (WPF .NET). *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview

[8] Microsoft Learn: XAML overview (WPF .NET). *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml

[9] YUEN, Sheridan. Mastering Windows Presentation Foundation: Build responsive UIs for desktop applications with WPF. 2nd Edition. Birmingham: Packt Publishing, 2020. ISBN 978-1838643416.

[10] MACDONALD, Matthew. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5. 4th Edition. New York: Apress, 2012. ISBN 978-1430243656.

[11] CHOWDHURY, Kunal. Windows Presentation Foundation Development Cookbook: 100 recipes to build rich desktop client applications on Windows. 2nd Edition. Birmingham: Packt Publishing, 2018. ISBN 978-1788399807.

[12] Microsoft Learn: What is Visual Studio?. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide

[13] STEPHENS, Rod. WPF Programmer's Reference: Windows Presentation Foundation with C# 2010 and .NET 4. Indianapolis: Wrox, 2010. ISBN 978-0470477229.

[14] Microsoft Learn: Learn C#. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/csharp

[15] SELLS, Chris a Ian GRIFFITHS. Programming WPF. 2nd Edition. New York: O'Reilly Media, 2007. ISBN 978-0596510374

[16] GAROFALO, Raffaele. Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern. California: Microsoft Press, 2011. ISBN 978-0735650923.

[17] Microsoft Learn: Patterns - WPF Apps With The Model-View-ViewModel Design Pattern. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern

[18] *Microsoft Learn: Model-View-ViewModel (MVVM)* [online]. In: . [cit. 2023-02-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

[19] Microsoft Learn: Entity Framework overview. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview

[20] Microsoft Learn: Entity Framework Core. *Microsoft* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/ef/core/

[21] SMITH, Jon P. Entity Framework Core in Action. 2nd Edition. Shelter Island: Manning Publications, 2018. ISBN 978-1617294563.

[22] S.O.L.I.D. Principles of Object-Oriented Programming in C#. *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://www.educative.io/blog/solid-principles-oop-c-sharp

[23] Microsoft Learn: Data binding overview (WPF .NET). *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data

[24] AddTransient, AddScoped and AddSingleton Services Differences. In: *Stack Overflow* [online]. [cit. 2023-05-05]. Dostupné z: https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences

[25] Microsoft Learn: .NET dependency injection. *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

[26] Microsoft Learn: Commanding Overview. *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/commanding-overview

[27] Microsoft Learn: Navigation Overview. *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/app-development/navigation-overview

[28] Microsoft Learn: Styles and templates (WPF .NET). *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/styles-templates-overview

[29] Microsoft Learn: Deploy a WPF Application. *Educative* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/app-development/deploying-a-wpf-application-wpf

[30] Project planning: What is it and 5 steps to create a plan. *TechTarget* [online]. [cit. 2023-05-04]. Dostupné z: https://www.techtarget.com/searchcio/definition/project-planning

[31] Functional vs Non Functional Requirements. *GeeksforGeeks* [online]. [cit. 2023-05-04]. Dostupné z: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/

[32] ŠTRÁFELDA, Jan. Autentizace. *Štráfelda* [online]. [cit. 2023-05-04]. Dostupné z: https://www.strafelda.cz/autentizace

[33] Autentizace jménem a heslem. *ITnetwork* [online]. [cit. 2023-05-04]. Dostupné z: https://www.itnetwork.cz/programovani/nezarazene/autentizace-jmenem-a-heslem-ukladani-hesel/

[34] Příručka pro personální agendu a odměňování zaměstnanců: VIII.4 Přestávka v práci a bezpečnostní přestávka. *Ministerstvo práce a sociálních věcí* [online]. [cit. 2023-05-04]. Dostupné z: https://ppropo.mpsv.cz/VIII4Prestavkavpraciabezpecnostn

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

WPF	Windows Presentation Foundation
XAML	eXtensible Application Markup Language
IDE	Integrated Development Environment – Vývojové prostředí
MVVM	Model-View-ViewModel
MVC	Model-View-Controller
EF	Entity Framework
ORM	Object-Relational Mapping – objektově relační mapování
SQL	Structured Query Language
LINQ	Language Integrated Query
IoC	Inversion Of Control
UI	User Interface – uživatelské rozhraní

SEZNAM OBRÁZKŮ