Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Desktopové aplikace v .NET Core

Bakalářská práce

Autor: Marek Růžička
Studijní program: Aplikovaná Informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové Duben 2024

Prohlášení:

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

V Hradci Králové dne 22.4.2024

……

Marek Růžička

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

Abstrakt

Tato bakalářská práce se věnuje prozkoumání frameworku .NET MAUI. Jejím hlavním cílem je vytvořit funkční vzorovou aplikaci, která bude použitelná na desktopovém i mobilním zařízení. Rámec této práce se zabývá problematikou použití jednotlivých komponent a odhalením jejich nedostatků během vývoje. První část je teoretická a hovoří o tvorbě desktopových aplikaci .NET a .NET MAUI. Další část práce definuje očekávání od vzorové aplikace, její požadavky a koncept uživatelského rozhraní. Poslední část se věnuje implementaci jednotlivých komponent se zdrojovými kódy k lepšímu porozumění. Také sleduje rozdíly v ovládacích prvcích na různých operačních systémech. Práce pomáhá lépe porozumět .NET MAUI.

Klíčová slova:

.NET, MAUI, multiplatformní, Framework, aplikace, komponenta

Abstract

This bachelor thesis is dedicated to the exploration of the .NET MAUI framework. The main goal is to create a functional sample application that will be usable on both desktop and mobile devices. This thesis deals with the issues of using each component and revealing their shortcomings during development. The first part is theoretical and talks about the development of .NET and .NET MAUI desktop applications. The next part of the thesis defines the expectations from the sample application, requirements, and the concept of the user interface. The last part discusses the implementation of each component with source codes to better understand it. It also traces the differences in controls on different operating systems. The thesis helps to understand .NET MAUI better.

Title: Desktop Applications in .NET Core

Key words:

.NET, MAUI, multiplatform, Framework, application, component

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

Obsah

1 Úvod

Bakalářská práce se zaměřuje na desktopový a mobilní vývoj v prostředí .NET.
V současné době se neustále vyvíjejí desktopová i mobilní zařízení poměrně rychle, a to s sebou přináší i potřebu rychlejšího vývoje aplikací pro obě platformy. Vývoj pro více zařízení na různých platformách umožní přístup více uživatelů a zároveň snížit náklady při vývoji těchto aplikací. Právě prostředí .NET nabízí možnost napsat jeden kód a ten spustit na různých platformách.

Hlavním významem .NETu je moderní přístup k vývoji aplikací. Mojí motivací pro tvorbu v .NET je i fakt, že celou střední školu mě provázel programovací jazyk C#, který se při tvorbě programů využívá.

Tato práce se zaměří na možnosti tvorby desktopových aplikací ve starším .NET Framework a postupně se přesune k novějšímu .NET Core až k současné technologii .NET MAUI. Zároveň porovná nativní a multiplatformní vývoj.

Závěrečná část bude věnována vývoji aplikace ve zmíněném .NET MAUI.

2 Možnosti tvorby desktopových aplikací

V dnešní době je využíván k práci počítač, na kterém jsou nainstalovány desktopové aplikace. Ty ale dávno nejsou jedinou možností, jak obsluhovat mnoho konkrétních věcí, ale přesto se stále využívají. Je to z důvodu využití hardwaru a všech jeho funkcí. Kromě desktopových aplikací je možné se častěji setkat s webovými, které nabízí přístup z jakéhokoliv zařízení, nebo mobilní pro pohodlné ovládání odkudkoliv.

Desktopové aplikace jsou vyvíjeny za pomoci vyšších programovacích jazyků, mezi které se řadí například: C/C++, Java, Python, C#. Tyto aplikace jsou většinou programovány pro konkrétní operační systém, tím většinou bývá Windows, protože je nejrozšířenější. To sebou může přinášet obtíže, v případě, že by se aplikace měla rozšířit na další systémy, pak je nutností zaplatit další vývoj pro tyto systémy.

Nevýhodou desktopových aplikací je problematické zálohování jejich obsahu. To znamená, pokud uživatel pravidelně neukládá svoji práci, může při pádu aplikace přijít o veškerou svoji práci, kterou vykonával. Zároveň je zde komplikovanější práce v týmu, protože většinou tyto aplikace fungují v režimu off-line. Toto se dá eliminovat použitím cloudového řešení, jako je Microsoft 365 [1].

Navzdory velkému přívalu webových a mobilních aplikací si stále desktopové aplikace drží své místo na trhu, a to hlavně z pohledu efektivity, výkonu pro výpočetně náročné aplikace a lepší interakci s uživatelem. Jednodušší přístup k hardwarovým prvkům zařízení, mezi které patří Bluetooth, kamera, čtečka karet atd. Zároveň technologie pro tvorbu tohoto druhu aplikací jsou již dlouho vyvíjené, poměrně vyspělé a hlavně otestované. Stabilní aplikace je ta nejlepší aplikace.

2.1 .NET

Technologie .NET někdy taky dotnet z anglického výrazu pro tečku dot je bezplatná technologie pro tvorbu aplikací. Umožňuje tvořit aplikace cloudové, pro Windows nebo na různé další platformy. Je vytvořená společností Microsoft.

Microsoft tuto technologii stále vyvíjí a každý rok v listopadu vydává novou verzi .NET. Tyto verze se dělí na dva druhy. Za pomocí čísla, které se nachází za názvem se dá jednoduše určit o jakou verzi se jedná.

- Lichá čísla jsou nazývána jako „Long-Term“ (LTS) a mají podporu po dobu 3 let.
- Sudá čísla jsou označována jako „Standard-Term“ a jejich podpora bývá po dobu 18 měsíců, což je 1,5 roku.

Podpora této technologie není pouze na operačním systému Windows, ale i na ostatních, jako je Android, Apple a MacOS. Aby byla podpora úplná je zapotřebí myslet i na různé architektury procesorů x64, x86 a AMR64, které jsou taktéž podporovány [2].

2.2 C#

C# je programovací jazyk, který mnoho lidí zná, a hlavně si jej spojují s platformou .NET, pro kterou byl i vytvořen dvěma významnými autory Andersem Hejlsberg a Scottem Wiltamuth [3]. Microsoft potřeboval pro svojí novou platformu .NET jednoduchý, typově bezpečný, moderní, a objektově orientovaný programovací jazyk. Jeho výhodou je jednoduchá syntaxe, která je podobná jazykům z rodiny C a C++, a proto přechod na něj je velice jednoduchý. C# měl podobné cíle jako Java, při vývoji byl upřen pohled na její chyby a snaha vytvořit lepší programovací jazyk. S příchodem .NET 8, byla vydána nejnovější verze tohoto programovacího jazyka, která nese označením C# 12. Ta sebou přinesla řadu vylepšení jako primární konstruktory, výrazy, vložená pole, ref readonly paramery [4].

ConsoleApp1 — ConsoleApp1.Program

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApp1
8   {
        Počet odkazů: 0
9       internal class Program
10      {
            Počet odkazů: 0
11          static void Main(string[] args)
12          {
13              Console.WriteLine("Hello world!\n");
14              int a = 1;
15              int b = 2;
16              int c = a + b;
17              Console.WriteLine($"Součet čísel {a} + {b} = {c}\n");
18              string s = "Ahoj světe";
19              for (int i = 0; i < c; i++)
20              {
21                  Console.WriteLine(s);
22              }
23              Console.ReadKey();
24          }
25      }
26  }
27
```

Obr. 1: Ukázkový kód v konzolové aplikaci v jazyce C#
Zdroj: Autor

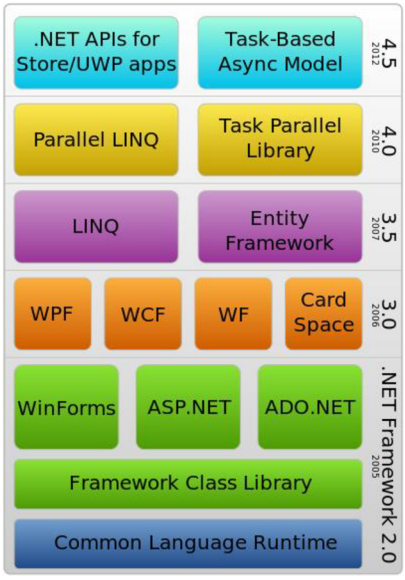
2.3 .NET Framework

.NET Framework je softwarový framework (softwarová struktura/knihovna) sloužící pro běh, správu a vývoj aplikací pro operační systém Windows. Jednou z důležitých komponent .NET Frameworku je Common Language Runtime (CLR), který spravuje kompilaci kódu a spouštění programů.

Dále nabízí unikátní vlastnost pro vývojáře, a tou je interoperabilita. Ta umožňuje v rámci jedné aplikace využít více programovacích jazyků na různé části. To dává větší flexibilitu vývojářů.

Klíčovými částmi .NET Frameworku z hlediska tvorby uživatelského rozhraní jsou Windows Form a Windows Presentation Foundation (WPF). Obě tyto možnosti umožnují tvorbu desktopových aplikací a liší se nabídkou nástrojů. Kromě tvorby desktopových aplikací nabízí i webové, a to pomocí technologie ASP.NET umožňující vytváření dynamických webových stránek.

.NET Framework nevyužívají pouze vývojáři, ale i běžní uživatelé na Windows. Ti však nepotřebují mít detailní znalosti o jeho fungování, ale jen to, že ho některé aplikace vyžadují pro svůj běh. Již od Windows 8 je .NET Framework součástí operačního systému. Na daném operačním systému se může objevovat více verzí .NET Frameworku současně. Zároveň by se neměly ostatní verze odinstalovat [5].



Obr. 2: Architektura .NET Framework
Zdroj: [6]

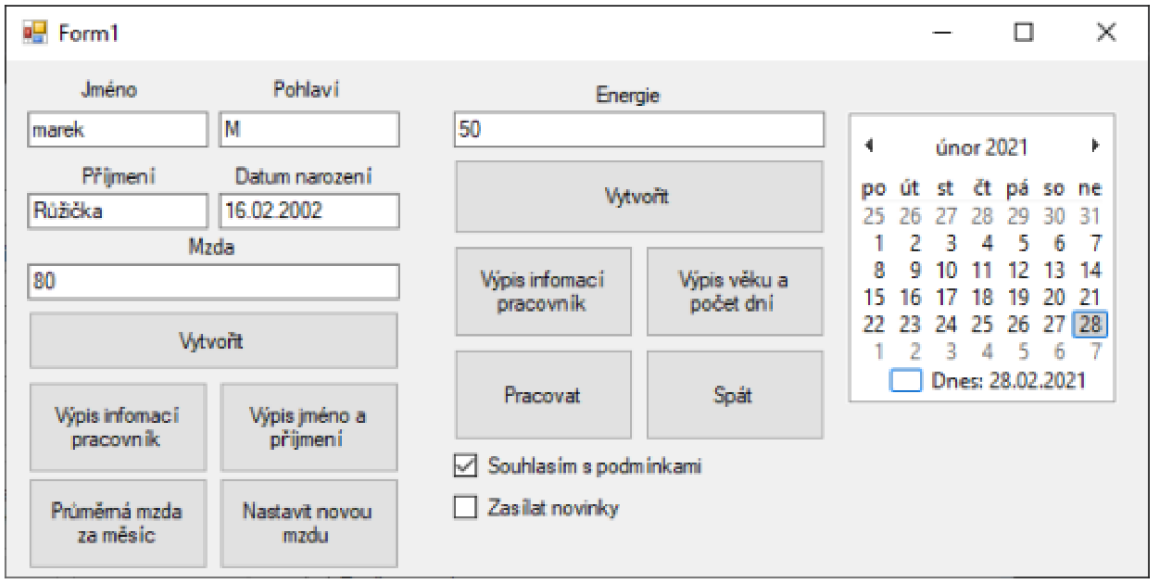
2.3.1 CLR

Common Language Runtime je klíčovou součástí .NET. Jedná se v podstatě o virtuální stroj podobně jako má Java svůj JVM (Java Virtual Machine), jehož úkolem je spouštět a spravovat napsaný kód v jazycích .NET [7]. Nabízí lepší přenositelnost, zabezpečení a jednoduchost vývoje aplikací. Každý jazyk v .NET kompiluje zdrojový kód do MSIL (Microsoft Intermediate Language). MSIL je strojově nezávislý kód, který obsahuje instrukce programu. Mimo jiné má funkci JIT – „just-in-time“, která překládá kompilovaný kód do strojového jazyka přímo za běhu programu. Dále poskytuje tyto služby: typová bezpečnost, správa paměti, zachycování výjimek a správu vláken [8].

2.3.2 Windows Forms

Windows Forms, někdy zkráceně WinForms, je uživatelské rozhraní sloužící k tvorbě desktopových aplikací pro operační systém Windows. Samotná tvorba těchto aplikací je velice intuitivní, a to i díky vizuálnímu návrháři, který je dodáván ve vývojovém prostředí Visual Studio od Microsoftu. V něm je možné za pomoci metody „drag and drop“ (táhni a pusť) jednoduše umisťovat jednotlivé komponenty, jako jsou textová pole, tlačítka apod. Následně lze s těmito komponentami pracovat a přiřazovat jim jednotlivé funkcionality. Nejběžnější funkcionalitou je reakce na stisknutí – například tlačítka, kdy se po stisku vykoná daný úkol [9].

Windows Forms přišly s .NET Framework 1.0 už v roce 2002, a jsou i nadále voleny pro tvorbu desktopových aplikací navzdory novějším technologiím jako je WPF, UWP nebo MAUI. Je to způsobeno jejich jednoduchostí a práci s daty.



Obr. 3: Ukázka Windows Forms aplikace
Zdroj: Autor

2.3.3 WPF

Windows Presentation Foundation je stejně jako Windows Forms technologie pro tvorbu desktopových aplikací na Windows a je součástí .NET Frameworku. Hlavním prvkem WPF je designový jazyk XAML. Ten pomáhá oddělit vizuální prezentaci od logiky aplikace, a tím přispívá k modularitě a údržbě kódu. Zároveň přináší bohatší grafické možnosti s podporou grafiky, animací a stylů. Výhodou při využívání jazyka XAML je tak zvaný data binding, který zajišťuje propojení mezi logikou aplikace a uživatelským rozhraním, tím se zjednodušuje manipulace s daty [10].

2.3.4 XAML

Zkratka XAML znamená eXtensible Application Markup Language a jedná se o deklarativní značkovací jazyk, který jak už název napovídá vychází z jazyka XML. Mimo jiné je i název XML obsažen v hlavičce souboru tohoto jazyka. XAML je určen pro tvorbu uživatelského rozhraní v technologii .NET. Jednotlivé prvky uživatelského rozhraní se definují za pomoci elementů. Tyto prvky lze definovat i v jazyce C#. U těchto elementů se ještě následně přiřazují vlastnosti dle příslušných atributů. Dále lze přidat předpona „x:“. Ta může nastavit jedinečný klíč „x:Key“ nebo název objektu „x:Name“. S tímto objektem jde následně pracovat v jazyce C#. Výhodou jazyka je oddělení grafického rozhraní pro lepší správu designu od aplikační logiky aplikace.

Při tvorbě designu aplikací je potřeba dbát na velká a malá písmena, protože je jazyk takzvaně „case-sensitive“, to znamená, že rozlišuje velká a malá písmena [11].

Níže uvedený zdrojový kód je ukázkou syntaxe jazyka. Všechny komponenty jsou vertikálně pod sebou, kdy první je obrázek, druhý je popisek a na závěr tlačítko, které je obsloužené v jazyce C# definovanou metodou v atributu události Clicked.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1_showCode7Net.MainPage">

    <VerticalStackLayout Padding="30,0" VerticalOptions="Center">
        <Image  Source="dotnet_bot.png"
                SemanticProperties.Description="Bot"
                HeightRequest="200"
                HorizontalOptions="Center" />
        <Label  Text="Zdravím vás"
                SemanticProperties.HeadingLevel="Level1"
                FontSize="32"
                HorizontalOptions="Center" />
        <Button x:Name="CounterBtn"
                Text="Klikni na mě"
                SemanticProperties.Hint="Po stiku se zapíše kliknutí"
                Clicked="OnCounterClicked"
                TextColor="White"
                BackgroundColor="{StaticResource Primary}"/>
    </VerticalStackLayout>
</ContentPage>
```

Zdrojový kód 1: Ukázka syntaxe zdrojového kódu XAML
Zdroj: Autor

2.3.5 Data binding

Data binding, česky datová vazba, je vazba dat mezi uživatelským rozhraním a daty, která jsou potřeba zobrazit. Umožňuje tak vývojářům propojit data z různých zdrojů, jako například z XAML souboru, kde je grafické rozhraní, a tím jednoduše aktualizovat UI. Velice často se využívá u textových polí, seznamů, popisů a dalších prvků. Data binding má dva základní směry toku a ty jsou: OneWay a TwoWay.

- **OneWay** – neboli jednosměrný tok dat znamená aktualizaci vlastností u cílového prvku. Zdrojový prvek zůstane nezměněný v původním stavu.
- **TwoWay** – je obousměrný tok dat, kdy vlastnosti prvků se aktualizují u cílového, tak i u zdrojového prvku. Tento způsob se většinou používá u formulářů, a to například na komponentech, jako je textové pole nebo zaškrtávací políčko [12].

2.3.6 UWP

Universal Windows Platform (UWP), vývojářská platforma, která umožňuje aplikace spouštět na různých zařízeních s operačním systémem Windows 10, včetně nejnovější verze Windows 11, ale také na Windows Phone nebo Xbox One zařízeních. Při vývoji na této platformě je třeba brát v potaz, že nelze aplikace spouštět na starších operačních systémech jako je Windows 7, Vista či XP, protože nejsou podporované z důvodu designu platformy. Důraz byl kladen na univerzálnost, optimalizaci a schopnost automaticky uzpůsobit uživatelské rozhraní napříč různými zařízenými a velikostmi obrazovek, a to od telefonu přes notebook, až po stolní počítač. Další výhodou je jednoduchá distribuce a aktualizace těchto aplikací, a to za pomoci Microsoft Store. Ta je výhodná nejen pro vývojáře, ale i pro uživatele. Stejně jako u WPF se zde využívá programovací jazyk C# a designový jazyk XAML, který pomáhá snadnému vývoji aplikací [13].

2.4 .NET Core

.NET Core představuje open-source, multiplatformní framework pro vývoj aplikací napříč zařízeními pro různé operační systémy. Jedná se v podstatě o nástupce .NET Framework, protože poslední verze 4.8.1 byla vydána v létě roku 2022 a předchozí v roce 2019. První verze .NET Core byla vydána už v roce 2016. Jak bylo zmíněno, jedná se o open-source framework, a tím pádem je jeho zdrojový kód dostupný veřejnosti. Díky tomu mohou vývojáři framework přizpůsobit svým potřebám a potřebám vyvíjené aplikace. Tím, že je multiplatformní, zrychluje navíc vývoj napříč operačními systémy, stačí napsat jeden kód a ten po zkompilování poběží na všech daných systémech. Zbývá jen přizpůsobit grafické rozhraní uživatele [14].

V roce 2020 přišla nová verze .NET s označením číslicí 5. Tato verze přinesla sloučení .NET Core s .NET Framework pod jednu platformu. Aktuálně nejnovější verzí je .NET 8. Tou Microsoft posílil zabezpečení, zvýšil výkon a zefektivnil údržbu. Tím, že kybernetické hrozby jsou na denním pořádku, byla přidána podpora hashovacího algoritmu SHA-3, která posiluje zabezpečení hesel a dalších citlivých dat. Microsoft také vylepšil práci se soubory JSON v knihovně „System.Text.Json“, a to hlavně v oblasti deserializace dat z tohoto souboru. Zároveň se také vylepšilo rozhraní .NET MAUI, které podporuje Android API 34 a pro operační systém iOS verzi 17 (XCode 15) [15].

2.5 Rozdíl mezi .NET Framework a .NET Core

Obě platformy jsou oblíbené, liší se ale v několika ohledech. .NET Framework je starší a podporuje pouze Windows. .NET Core je novější a multiplatformní. Dalším rozdílem je jejich správa. .NET Framework je udržován Microsoftem, zatím co .NET Core je open-source, a tím k jeho správě může přistupovat i komunita.

2.5.1 Který je lepší

Nelze jednoznačně určit, který z frameworků je lepší, protože každý má svá specifika, stejně jako jsou požadavky cílové aplikace. Hlavní specifikum je operační systém a cílová skupina. Zda má být určena pouze pro Windows nebo je implementována i na další operační systémy. Jestliže vývoj začne od nuly, je dobré zvolit již od začátku .NET Core. Nejen, že ho Microsoft upřednostňuje a zaměřuje na něj svůj vývoj, je také open-source, a tím ostatní vývojáři přispívají k jeho zlepšení nebo přidávají další rozšíření, která můžou usnadnit vývoj ostatním vývojářům. Samozřejmě je zde i možnost začít projekt tvořit v .NET Frameworku a později migrovat na .NET Core, avšak to s sebou může přinést komplikace. .NET Core nemusí podporovat všechny knihovny a pokud aplikace běžela stabilně v prostředí .NET Frameworku, nemusí tomu tak být v prostředí .NET Core. Pokud je prioritou výkon, je lepší zvolit .NET Core, který je prostě novější. Naopak .NET Framework je tu delší dobu a přináší stabilnější prostředí [16].

2.6 .NET Standard

Microsoft se postupným vývojem .NET platformy dostal do bodu, kdy měl tři hlavní větve:

- .NET Framework
- .NET Core
- Xamarin[1]

[1] Xamarin je opensourcová platforma pro vývoj mobilních aplikací, která umožňuje vytvářet nativní aplikace pro Android, iOS a Windows, a to pomocí sdíleného kódu v jazyce C#. Založené projekty v Xamarinu je možné migrovat na jeho nástupce .NET MAUI [17].

Každá platforma byla totiž navržena pro jiné účely. Pokud by uživatel chtěl vyvíjet nejdříve v jedné a pak následně v druhé, musel by se naučit obě nebo nejlépe všechny platformy. Proto se Microsoft rozhodl vydat .NET Standard. Se standardem **.NET Standard 2.0** sblížil všechny platformy. Díky standardizaci API mohl být vytvořený kód sdílený napříč různými částmi ekosystému .NET.

Poslední vydanou verzí je **.NET Standard 2.1** vydána v roce 2018. Novější verze tohoto standardu již nebude, a to i z důvodu integrace nového rozhraní API do nových verzí .NET 5 a výše. Proto je lepší cílit vývoj aplikací na co nejnovější verze .NET [18].

3 Multiplatformní vs nativní vývoj

Vývoj aplikací se rozděluje na dva hlavní tábory, a to nativní, který zaměřuje vývoj pro daný operační systém a má přístup k jeho nativním nástrojům, a na vývoj multiplatformní, který již dle názvu naznačuje, že má za cíl danou aplikaci dostat na co nejvíce operačních systémů najednou. Dále existuje ještě třetí možnost, jak vyvíjet aplikace a tou je hybridní vývoj. Jedná se převážně o kombinaci webových a nativních technologií.

3.1 Nativní vývoj

O nativním vývoji hovoříme jako o nutnosti vyvinout aplikaci pro každou platformu zvlášť, to sebou přináší vyšší náklady, pokud je cílem vytvořit aplikaci pro dva či více operačních systémů. Příkladem může být Android a iOS. Vyšší náklady nejvíce spočívají v potřebě ideálně dvou týmů vývojářů. Nehledě na to, že každý z těchto dvou operačních systémů využívá jiný programovací jazyk při tvorbě aplikace. U androidu se využívá programovací jazyk Kotlin, u iOS zase Swift. Na druhou stranu to přináší i výhody ve využití nativních nástrojů daného systému. To přispívá k lepší optimalizaci, výkonu aplikace, ale také zlepšuje zážitek ze samotného používání. Hlavní nevýhody spočívají v delším vývoji, náročném testování, a hlavně nutnosti dvou programátorských týmů.

3.2 Multiplatformní vývoj

Klíč k úspěchu u multiplatformního vývoje spočívá v nabídkách služeb pro co nejvíce zákazníků najednou na různých platformách. Ať už se jedná o desktopovou, mobilní nebo kombinaci těchto platforem dohromady. Při vývoji tímto způsobem se sdílí většina programového kódu mezi různé platformy a operační systémy. Řádově toto bývá odhadem kolem 80 až 90 procent v závislosti na konkrétních požadavcích aplikace. Zbylá procenta pak slouží k doladění a vylepšení finální aplikace u konkrétních operačních systémů, aby vše běželo, jak má. Primární výhodou je rychlost a snížení nákladů při samotném vývoji, jelikož bude ve většině případů stačit jeden tým. Jestliže je aplikace dobře napsaná, uživatel nemá šanci rozpoznat, zda byl vývoj nativní nebo multiplatformní. Zároveň pokud se v aplikaci vyskytne nějaká chyba, bude její oprava mnohem rychlejší a opraví se zároveň i u ostatních operačních systémů.

Může se zdát, že multiplatformní vývoj je ta nejlepší a nejvýhodnější cesta, ale ne všichni se na něj spoléhají, protože má i své chyby. Jako příklad lze uvést nedostupnost některých nativních nástrojů dané platformy. Ty jsou občas potřeba a pokud nejsou hned dostupné, musí se pak využít balíčky třetích stran. Ty nemusí být vždy úplně na škodu, ale mohou přinášet jistá rizika v podobě bezpečnosti, nesprávného fungování, nebo mohou být dokonce zastaralé a neaktualizované. Dále může nastat případ, kdy určité knihovny či frameworky nebudou kompatibilní s danou verzí operačního systému. Poté je následně potřeba hledat alternativy. To může zdržovat a být při vývoji i lehce frustrující. Ale pokud je primární cíl oslovit co nevíce zákazníků za krátkou dobu je multiplatformní vývoj ideální volbou [19].

4 Ukládání dat

Ukládání dat je klíčovým prvkem všech moderních aplikací. Tato data je zapotřebí ukládat na bezpečná uložiště, pokud se jedná o velmi citlivá data. Dále je efektivně roztřídit na uživatelská, konfigurační a další. Existuje více možností, jak data ukládat a následně spravovat. Tou nejzákladnější je lokální uložiště. Dnes už nebývá tak časté, ale stále si konkrétní data může aplikace uložit pouze na daném zařízení a budou dostupná jen z něj.

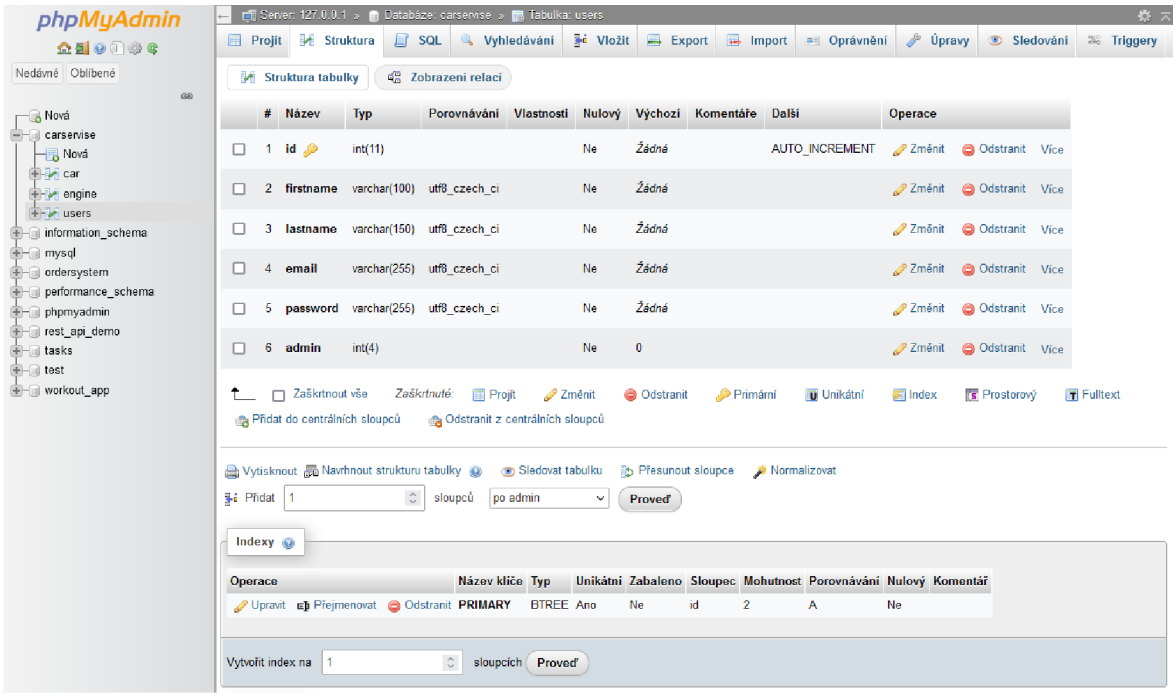
Nejčastějším řešením jsou databáze. Ty mohou být relační nebo strukturované do souborů. V poslední době se čím dál více setkáváme s variantou cloudových uložišť. Tento způsob se velice často kombinuje s lokáním uložištěm. To přispívá k možnosti využívat aplikaci jak off-line, kdy budou data ukládaná lokálně a po připojení k online síti aktualizována a zálohována na cloudové uložiště. Při manipulaci s daty je potřeba dbát na jejich zabezpečení, včetně šifrování, ale také uplatnit autentizaci a autorizaci. Zvolení vhodného uložiště není zcela jednoduché a je zapotřebí brát v potaz data, se kterými se pracuje, následně škálovatelnost a výkon.

4.1 MySQL

MySQL je relační databázový systém, který je bezplatný, ale je i dostupný v placené verzi pod názvem MySQL Enterprise Edition. O tento bezplatný databázový systém se stará společnost Oracle. Je populární díky své jednoduchosti, spolehlivosti a rozsáhlé podpoře. SQL (Structured Query Language) je standardizovaný jazyk k vyhledávání dat v databázi za pomoci dotazů. Data v tomto systémů jsou strukturována do tabulek, kde každá tabulka obsahuje sloupce, které definují strukturu a datové typy jednotlivých dat. Řádky následně představují jednotlivé záznamy dat. Tyto tabulky obsahují klíče, kdy primární klíč je jednoznačný identifikátor záznamu řádku v tabulce, zatímco cizí klíče vytváří vazby mezi tabulkami. Jednotlivé tabulky jsou propojené vazby mezi sebou [20]. Rozlišují se tři primární vazby.

- **1:1** – Jednomu záznamu z první tabulky odpovídá právě jeden záznam z tabulky druhé
- **1:N** – Jednomu záznamu z první tabulky odpovídá více záznamů v tabulce druhé
- **M:N** – Jakémukoliv záznamu v první tabulce odpovídá libovolný počet záznamů z tabulky druhé. Tato vazba je však v reálu řešená přes asociační tabulku, ve které se nachází cizí klíče odkazující na záznamy v obou tabulkách.

Zároveň pro správu tohoto databázového systému je open-source rozhraní PHPmyAdmin, které běží ve webovém prohlížeči. Pomocí grafického rozhraní lze aplikovat dotazy nad databází. Případně si i nechat data vyexportovat, a to buď ve formátu SQL, CSV a dalších. Toto rozhraní bylo navrženo, tak aby bylo snadno instalovatelné na serverech s podporou PHP a MySQL.



Obr. 4: Prostředí PHPmyAdmin
Zdroj: Autor

4.2 API

API neboli application programming interface, je rozhraní pro komunikaci aplikace s externími systémy. API rozhraní se používají pro budování jednoduchého způsobu připojení a integrace softwarových systémů. Poskytují opakovaně použitelné rozhraní, ke kterému se mohou snadno připojit různé aplikace. Samotné API, ale nenabízí uživatelské rozhraní, protože obvykle nejsou vidět na povrchu a zároveň s nimi většinou nebude přímo komunikovat koncový uživatel, ale typicky nějaká koncová

aplikace. Velice často se API využívá u mobilních, cloudových a webových aplikací. Kompletní rozhraní API se obvykle skládá:

- Z klienta nebo aplikace, ta volá rozhraní API a zpracovává data poskytnutá rozhraní. Zároveň je tento klient zodpovědný za zkušenost u koncového uživatele
- Dále z rozhraní API, které poskytuje aplikaci data
- A také z platforma API, která spravuje rozhraní API

Nejpoužívanějším druhem je API s architekturou REST. REST je architektonický styl, jenž popisuje, jak navrhovat distribuované systémy, které využívají internetový protokol HTTP ke komunikaci. Architektura API, ale může být mnohem více než jen správné použití principů REST. Architektura může odkazovat na kompletní řešení, které se skládá ze samotného API, ale také klienta (například mobilní aplikace) a několika dalších komponent [21].

5 .NET MAUI

.NET MAUI – Multi-platform App UI (User Interface) je bezplatné multiplatformní rozhraní společnosti Microsoft. Slouží pro tvorbu nativních mobilních a desktopových aplikací. Při vytváření pomáhá programovací jazyk C# a designový jazyk XAML. První verze .NET MAUI byla vydána na jaře roku 2022 a hned od jeho počátku mohly vytvořené aplikace běžet na operačních systémech Windows, Android, iOS, ale i na macOS, a to vše z jediného sdíleného základu. To je i mimochodem hlavní cíl rozhraní .NET MAUI, aplikovat co největší část logiky a uživatelského rozhraní aplikace do jednoho základu. Takže stačí vytvořit jeden projekt pro všechny již výše zmíněné operační systémy. Aplikace pro různé operační systémy se kompilují vždy z jazyka C#, do nich určených. Pro Android to je intermediate language, který se následně při spuštění zkompiluje do nativního kódu. U Windows se využívá knihovna WinUI3 pro nativní aplikaci.

.NET MAUI je ve své podstatě nástupcem Xamarin.Forms, které původně umožňovaly tvorbu aplikací pro operační systémy Android a iOS. Jedním z rozdílů je podpora nativních ovládacích prvků u .NET MAUI, které může využít autentických ovládacích prvků pro každou platformu, což přispívá lepšímu zážitku z aplikace. Naproti tomu Xamarin.Forms používal vlastní komponenty, což občas mohlo vést ke kompromisům. Další předností .NET MAUI je opětovné načítání zdrojového kódu za provozu. To přispívá k lepšímu vývoji, protože není potřeba pozastavit běh aplikace, ale dá se provést změna za běhu. Toto platí i pro designový jazyk XAML, takže se dá měnit vzhled aplikace za běhu a tím lépe opravovat její nedostatky [22].

5.1 Ovládací prvky

Ovládací prvky jsou základním kamenem každé aplikace. Bez ovládacích prvků by se nedalo pracovat s žádnou aplikací. V .NET MAUI je celá řada těchto prvků. Typickým ovládacím prvkem je tlačítko. Každé komponentě se přiřazují vlastnosti v jazyce XAML. Tyto vlastnosti určují vzhled daného prvku a jeho chování při používání. Komponenty se dají definovat dvojím způsobem, a to buď v jazyce XAML anebo v programovacím jazyce C#. Jednodušší a přehlednější je to však v prvním zmíněným.

Další velice často používanou komponentou je **Entry**. Jak z názvu vyplývá, jedná se o vstupní prvek. V HTML by se označil jako input, tedy vstupní pole. Stejně tak u jiných komponent se dají nastavit různé vlastnosti, a tím upravit samotný vzhled.

Menším nedostatkem .NET MAUI oproti dříve používaným WinForms je absence datagridu, který není, co by defaultní komponenta dostupná. Pokud je třeba využít něco podobného, co nabízel WinForms, lze využít doplňujících balíčků nuget od jiných společností. Ty nabízejí i jiná řešení pro další komponenty. Je možné využití placených i neplacených komponent.

Další alternativou je využití zabudovaného **TableView**, který obsahuje určité nevýhody při používání. Samotný vzhled komponenty je převážně designovaný pro mobilní zařízení než pro desktop, pokud se využije zobrazení buněk **TextCell** nebo i **ImageCell**. Vlastní vzhled se dá pak definovat pomocí **ViewCell**, kde programátor udělá vlastní vzhled [23].

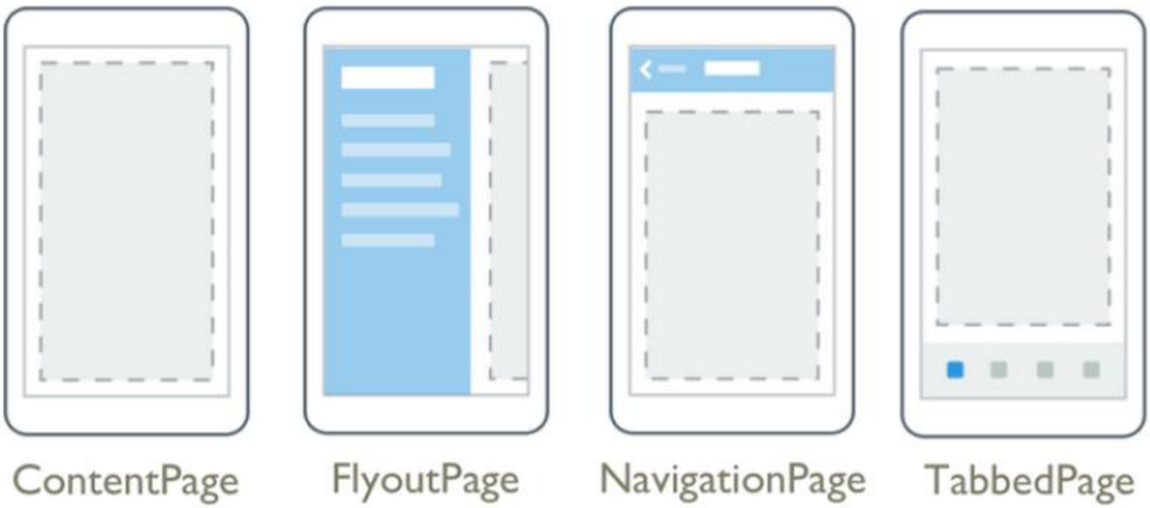
5.2 Stránky

Hlavním prvkem pro tvorbu obsahu stránky je ContentPage, která obsahuje uživatelské rozhraní s ovládacími prvky. Obsahuje Content, který má vždy vlastnost View.

Další stránkou je FlyoutPage. Tato možnost nabízí menu pro uživatele, které se většinou zobrazuje na levé straně obrazovky. Na mobilních zařízeních nám téměř vždy toto menu překryje část obsahu, na větších zařízeních je pak menu zobraceno celé.

Stránka NavigationPage, jak z anglického názvu vyplývá, jedná se o navigační stánku. V případě .NET MAUI se jedná o navigační lištu, na které se nachází titulek stránky a popřípadě tlačítko pro vrácení zpět na předchozí stranu. Dále se v listě mohou nacházet další tlačítka nebo funkce, a to například tlačítko pro odhlášení uživatele nebo rozbalovací nabídka s více možnostmi.

Poslední stránkou je TabbedPage. Obsahuje viditelné menu, které se nachází na spodní nebo horní stráně obrazovky. Každá položka tohoto menu má podřízený obsah ContentPage s uživatelským rozhraním. Vždy je zobrazena pouze jedna daná stránka, na které se uživatel právě nachází. Pokud by chtěl na jinou musí se přepnout.



Obr. 5: Uživatelské rozhraní
Zdroj: [24]

5.2.1 Rozložení stránek

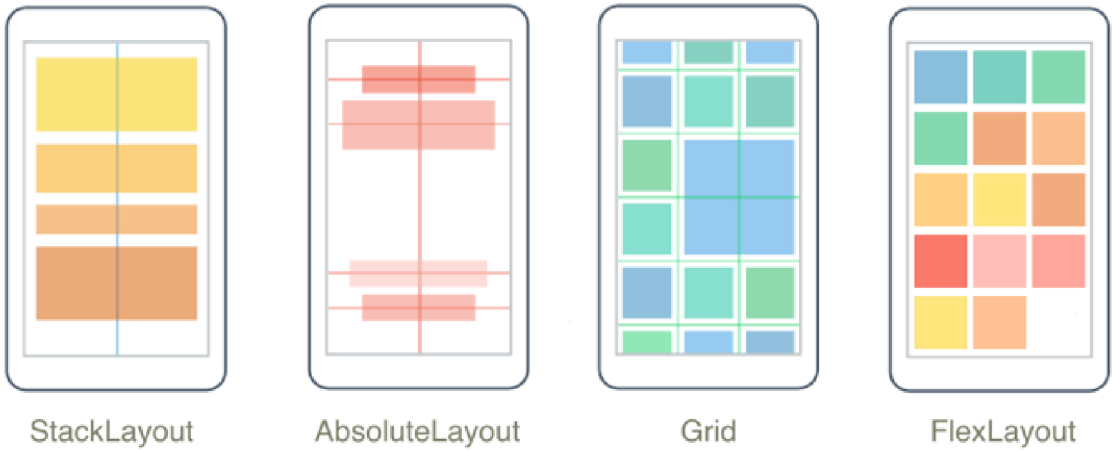
Každá stránka nebo část stránky může mít své uspořádání dle potřeby. Toto rozložení definuje, jak bude uživateli zobrazeno grafické rozhraní. Každý typ rozložení má svůj řád a zároveň ovlivňuje uspořádání a velikost podřízených komponent.

Prvním způsobem rozložení stánky je **StackLayout**. Často také nadřazené rozložení. Zajišťuje řazení komponent do zásobníku. Z tohoto zásobníku jsou komponenty vytažené a vykreslené za sebou. Zápis se může lišit, záleží na orientaci a řazení komponent. Vývojář může zvolit zápis **StackLayout** a k němu nastavit vlastnost **Orientation**, která nabývá hodnoty Horizontal nebo Vertical, která je zároveň nastavená jako výchozí stav. Obě orientace se dají zapsat rozdílně. Zápis je následující: **VerticalStackLayout** a **HorizontalStackLayout**. V názvu je následně daná definice orientace komponent.

Druhý způsob rozložení je **AbsoluteLayout**. Tento způsob není využíván příliš často, protože určuje absolutní umístění komponent. Pokud je třeba některé komponenty překrývat, tento způsob to umožňuje. Pozice komponent je určena od levého horního rohu obrazovky a využívá jednotek nezávislých na zařízeních. Umístění se dá zapsat pomocí souřadnic x, y. Ty je ještě možné doplnit o šířku a výšku.

Třetí možností je **Grid** neboli česky mřížka. Určuje uspořádání do řádků a sloupců. Dále umožňuje definovat různorodé množství řádků a sloupců dle potřeby. Jednotlivé mřížky mohou být přes více sloupců nebo řádků najednou. Velikost řádků a sloupců může být buď pevně daná nebo se velikost přizpůsobí komponentě či prvků umístěným v daném místě.

Poslední možnost je **FlexLayout**. Tento je podobný StackLayoutu, protože si také ukládá komponenty do zásobníků a zobrazuje je svisle nebo vodorovně podle předem dané definice. Jeho výhoda spočívá v nastavení přetečení množství položek za pomoci vlastnosti **Wrap**. Pokud je tato vlastnost nastavena a nacházelo by se příliš mnoho komponent na jednom řádku, další komponenty se zařadí na řádek nový. Další vlastností je **JustifyContent**, která zajistí práci s prostorem kolem položek. Samozřejmě těchto vlastností je nespočet [25].



Obr. 6: Možnosti rozložení stránek
Zdroj: [25]

5.3 Vývojové prostředí Visual Studio

Vývojové prostředí – anglicky integrated development environment zkráceně IDE. Jsou vývojová prostředí pro vývojáře k jednoduší práci, správě kódu, se spoustou nástrojů na jednom místě.

Jedno z takových prostředí pro snadný vývoj nabízí i Microsoft, a tím je komplexní vývojové prostředí Visual Studio. To přináší vývojářům lepší psaní kódu, snadné ladění a sestavení kódu aplikace. Při vývoji kódu lze využít i umělou inteligenci, která je v současné době hodně populární. Použít se dá GitHub Copilot a IntelliCode. IntelliCode je ve výchozím nastavení sady Visual Studio. Tato umělá intelligence se snaží predikovat dokončení kódu daného řádku na základě aktuálního kódu. Nebo analyzuje opakující se kód a ten se pokusí zjednodušit, či dokonce navrhuje argumenty. Vše toto zcela zdarma. Na druhé straně je zde GitHub Copilot, který je placený, ale o to více zase nabízí. Využívá pokročilejších nástrojů strojového učení z veřejně dostupných zdrojových kódů na uložišti GitHub, a tím zlepšuje návrhy kódu. Alternativně si lze celý

kód nechat vygenerovat a okomentovat. Je to skvělá věc pro usnadnění a zrychlení vývoje. Dále vývojové prostředí nabízí správu verzí za pomoci Gitu. Další funkcí je i Visual Studio Live Share. Tato funkce umožňuje sdílet svůj projekt s kolegy a společně na něm spolupracovat. Součástí jsou i funkce pro ladění a testování kódu. Poslední verzí a aktuální verzí je Visual Studio 2022 [26].

6 Vzorová aplikace

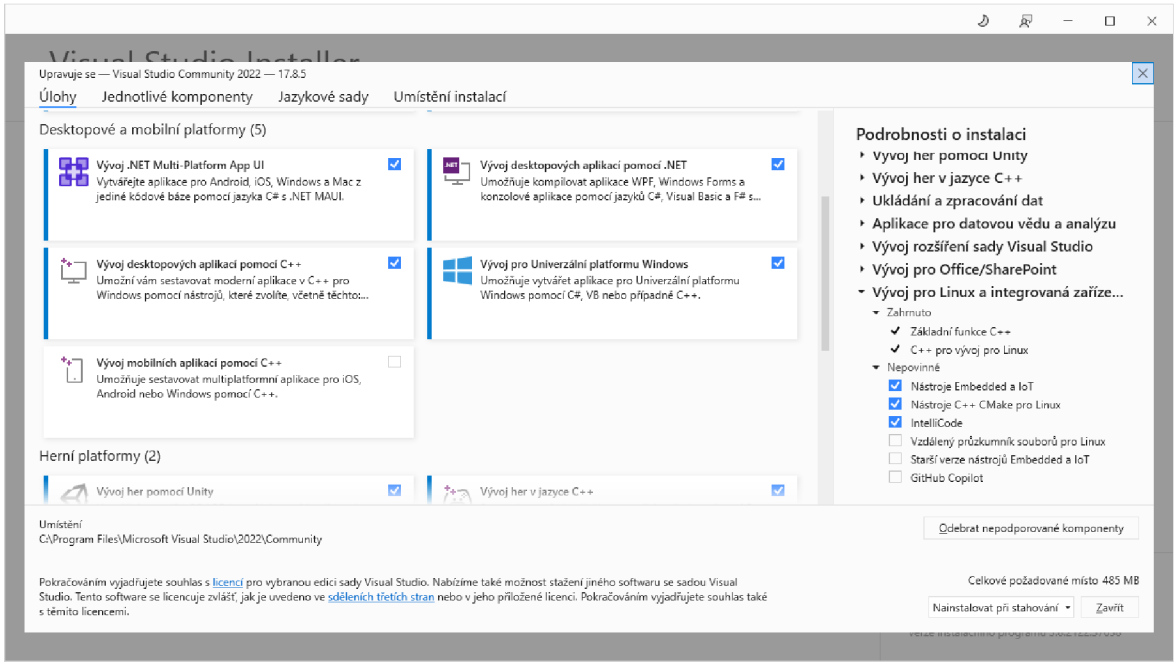
Vzorová aplikace servisní knížky pro automobil má za úkol odrážet funkcionality a možnosti nabízené frameworkem .NET MAUI. To znamená, aby byla aplikace ideálně použitelná na desktopovém zařízení a zároveň na mobilním. Samozřejmě každá platforma má svá omezení, a to může přinést i nějaké kompromisy. Zároveň bylo cílem vytvořit takovou aplikaci, aby se data přenášela mezi platformami. Pokud se udělají změny v mobilní části, aby se následně tyto změny projevily také v desktopové verzi aplikace nebo v jakémkoliv jiném zařízení, kde je aplikace nainstalována.

6.1 Požadavky

Uživatel se zaregistruje do aplikace pod svým jménem a emailem. Po přihlášení uvidí seznam svých vozidel s parametry, které jí zadal. Tyto technické údaje o vozidle je možné měnit dle potřeby. Pokud by měl uživatel nějaký údaj navíc, který není přímo ve formuláři, doplní jej do poznámky o vozidle. Následně bude zapisovat jednotlivé opravy svého vozidla. Po těchto opravách bude zapisovat stav kilometrů a datum, aby věděl, kdy naposledy byla daná oprava provedena. Budou následovat informace o jakou část vozidla se jedná a o jaké komponenty. Tyto díly podrobně popíše z popisu na webových stránkách. K těmto komponentům bude moci přiložit odkaz na webovou stránku, ze které díly objednal. Pokud mění některé díly často, jako například motorový olej a olejový filtr, nebude muset pokaždé hledat daný díl na stránkách obchodníka. Jednoduše klikne na odkaz, který ho přesměruje na stránku daného obchodu.

6.2 Založení projektu ve Visual studiu

Před samotným založením projektu je nejprve důležité stáhnout Visual Studio ze stránek Microsoftu. Toto vývojové prostředí je nabízeno ve třech variantách. Přičemž verze Community je zcela zdarma a nabízí dostatek funkcí pro běžný vývoj aplikací. Po stažení konkrétní verze, se sada Visual Studia začne instalovat. Také je potřeba zvolit konkrétní sady, které je potřeba nainstalovat. Pro vývoj v .NET MAUI je nejdůležitější zvolit: **Vývoj .NET Multi-Platform App UI**, viz obrázek 7. Zároveň v podrobnosti o instalaci se dají zvolit další nepovinné údaje. Mezi nimi je možnost doinstalovat i umělou inteligenci pro pomoc s vývojem GitHub Copilot.



Obr. 7: Instalační sada Visual Studio
Zdroj: Autor

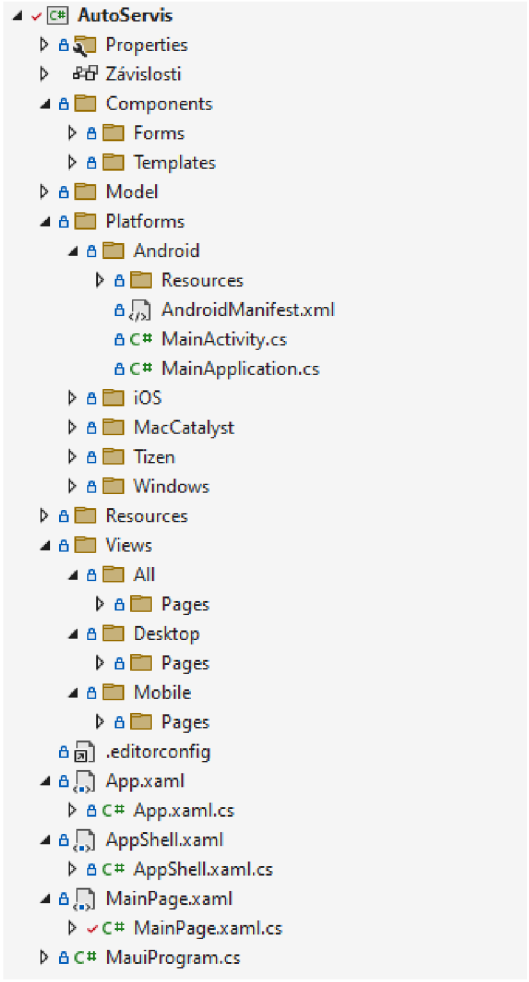
Po dokončení instalace je potřeba spustit již samotné Visual Studio a v něm vytvořit projekt. Pro .NET MAUI se nabízí hned tři možnosti pro vytvoření projektu, přičemž každá z nich má trochu jiný účel.

- Aplikace .NET MAUI – je primární technologie pro tvorbu multiplatformní aplikací. Tato technologie byla popsána v kapitole 5.
- Aplikace .NET MAUI Blazor – tato možnost spojuje technologie .NET MAUI a Blazor. Tímto způsobem umožňuje vytvořit již známou multiplatformní verzi a doplnit jí ještě o její webovou aplikaci. Díky technologii WebAssembly je možné spouštět kód napsaný v C# přímo ve webovém prohlížeči.
- Knihovna tříd .NET MAUI – poslední možnost nachází využití v potřebě vytvářet komponenty či další věci, které se budou následně sdílet mezi další projekty.

Pro vzorovou aplikaci je nejlepší volbou první varianta, která přinese vygenerovanou strukturu projektu a všechny potřebné věci pro start vývoje aplikace.

6.3 Struktura projektu

Nejdříve je třeba popsat strukturu projektu, část důležitých věcí projektu vytvoří přímo sada Visual studio. Není to však vždy úplně dokonalé a je zapotřebí to rozšířit dle potřeby projektu, k lepšímu přehledu. Je důležité vědět, co která část dělá a má na starosti.



Obr. 8: Struktura projektu .NET MAUI
Zdroj: Autor

- **Properties** – obsahuje soubor **lanchSettings.json**, slouží pro konfiguraci nastavení pro spuštění a ladění aplikace. Dále v něm lze nakonfigurovat profily, argumenty atd.
- **Závislosti** – se rozdělují podle operačního systému. V nich se následně dají nalézt balíčky NuGet, knihovny třetích stran, ale taky sady SDK, což je sada vývojových nástrojů.
- **Components** – tato složka není v základu vytvořená při generování projektu pomocí sady Visual Studia. Napomáhá rozšířit projekt o společné komponenty, které jsou stejné napříč všemi platformami. Nejvýznamnější podsložkou je

Forms. Ta obsahuje společné formuláře a jejich jednotlivá textová pole, rozbalovací seznam, zaškrtávací tlačítka a podobně.

- **Model** – tento adresář není běžnou součástí, ale byl vytvořen pro nutnost projektu. Obsahuje logiku aplikace a jednotlivé třídy, které jsou potřebou pro aplikaci.
- **Plaforms** – obsahuje dalších pět podsložek podle platformy, ve kterých se nachází specifické soubory pro každou z nich. Ty se následně můžou upravovat dle potřeby. U platformy Android v souboru **MainActivity.cs** je přepsaná metoda OnCreate a je v ní definováno otočení na výšku. Z důvodu, aby se projekt neotočil na šířku.

```
protected override void OnCreate(Bundle savedInstanceState)
{
    base.OnCreate(savedInstanceState);
    RequestedOrientation = ScreenOrientation.Portrait;
}
```

Zdrojový kód 2: Nastavení orientace obrazovky na výšku u Androidu
Zdroj: Autor

- **Resourses** – obsahuje jednotlivé prostředky aplikace, mezi které spadají obrázky, ikony, různé druhy písma nebo i barvy ve složce **Styles**.
- **Views** – je další adresář, který nevytvoří přímo Visual Studio. Cílem tohoto adresáře je oddělit mobilní (mobile) a desktopovou (desktop) část. Je to hlavně z důvodu, že počítače mají mnohem větší obrazovky na využití než telefon, který ji má poměrně malou. Následně jsou zde ještě jednotlivé stránky.

Po jednotlivých adresářích se nachází již jednotlivé soubory, a to většinou s příponou souboru **.xaml**. Ty obsahují výchozí design aplikace. Většina z nich se však ještě rozšiřuje o soubor s příponou **.xaml.cs** a už obsahuje programovou část s programovacím jazykem C#.

- **App.xaml** – definuje prostředky pro celou aplikaci
- **App.xaml.cs** – definuje počáteční stránku aplikace. V případě aplikace se odkazuje na přihlašovací stránku a tu ještě rozlišuje podle platformy.

```
using AutoServis.Views.Desktop.Pages.Login;
using AutoServis.Views.Mobile.Pages.Login;
namespace AutoServis
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            //MainPage = new AppShell();

            #if ANDROID || IOS
                MainPage = new NavigationPage(new MobileLogin());
            #else
                MainPage = new NavigationPage(new DesktopLogin());
            #endif
        }
    }
}
```

Zdrojový kód 3: Definice počáteční stránky
Zdroj: Autor

- **AppShell.xaml** – slouží pro definici vícestránkového rozhraní aplikace
- **Mainpage.xaml** – definuje vizuální rozhraní dané stránky aplikace
- **Mainpage.xaml.cs** - Určuje hlavní logiku dané stránky. Zde se obsluhují jednotlivé komponenty jako jsou tlačítka a další. Ty následně vykonají potřebný kód po stisku.
- **MauiProgram.cs** – definuje startovací třídu a další nastavení aplikace. Ve výchozím stavu se jedná o třídu App [27].

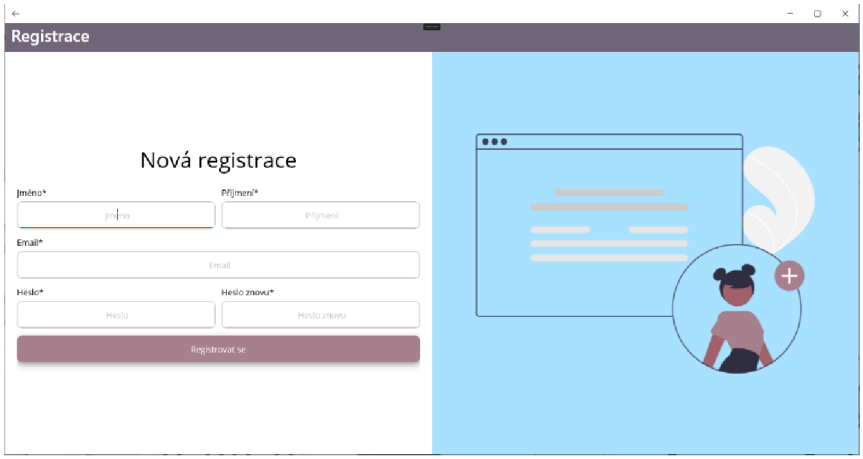
6.4 Koncept uživatelského rozhraní

Cílem uživatelského prostředí je jednoduchost a intuitivní ovládání pro uživatele. To znamená, že uživatel se bude v aplikaci snadno orientovat. Zároveň je třeba brát na zřetel, že aplikace poběží na desktopovém zařízení, kde bude vše ve větším zobrazení, a proto může být zobrazeno více detailů, a tím se může lehce lišit od mobilní verze. Je to hlavně způsobeno velikostí obrazovky, kde se dá využít většího prostoru na desktopovém zařízení. U počítače uživatel primárně komunikuje skrze klávesnici a myš. V tomto případě je zapotřebí se co nejvíce zaměřit na zobrazení tlačítek. U mobilního zařízení naopak uživatel využívá prst pro dotyk na display. Proto kromě kliknutí na tlačítka se dají využít i gesta pro ovládání. Mezi nejběžnější gesta patří dvojité poklepání, potáhnutím prstem do strany apod. Zároveň je u mobilního zařízení

třeba více dbát na responsivní design, kvůli omezenému prostoru. Důležitým prvkem je i to, že uživatelská rozhraní na mobilním a desktopovém zařízení si budou co nejvíce podobná jen s menšími rozdíly. To i mimo jiné zajišťují stejné formuláře.

Obr. 9: Uživatelské rozhraní na telefonu
Zdroj: Autor

Vzhledem k tomu, že aplikace bude mít stejné funkce na obou zařízeních je důležité, aby i design aplikace byl co možná s nejmenšími rozdíly. Samozřejmostí je rozdílné ovládání, ale aby základ byl co nejvíce stejný. K tomu přispívá i složka **Components** popsaná v kapitole 6.3, kde jsou vytvořené formuláře a ostatní komponenty, které mají stejnou funkci a podobný design, aby se příliš nelišil mezi platformami. Mimo jiné u mobilního zařízení se počítá s tím, že bude vždy telefon na výšku, a to je nastavené i v kódu. Popsáno v kapitole 6.3.

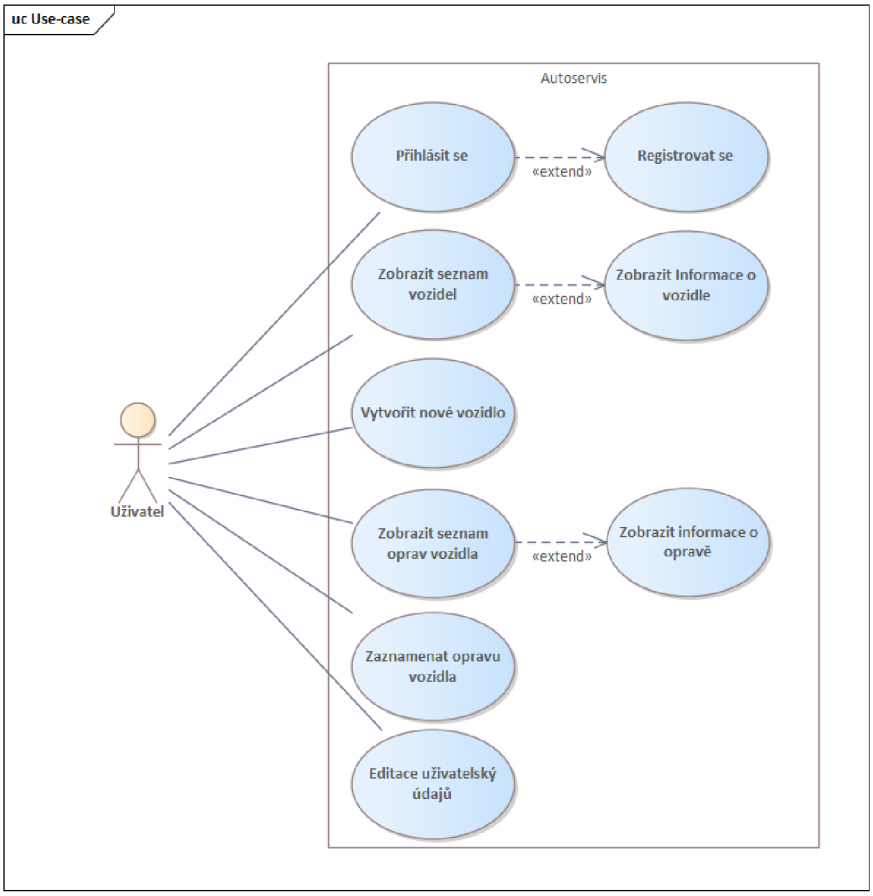


Obr. 10: Uživatelské rozhraní na desktopu
Zdroj: Autor

Při porovnání obrázků č. 9 a č. 10, je design aplikace podobný. Samotný formulář je zcela totožný a nejsou v něm vidět žádné rozdíly. To uživatelům pomůže se lépe orientovat v aplikaci.

6.5 Případy užití

Vzhledem k tomu, že aplikace bude mít stejné funkcionality jak na desktopovém, tak i mobilním zařízení, je zcela jedno jakou verzi bude uživatel využívat. Proč je vlastně dobré mít dvě verze a nejen mobilní. Mobilní zařízení máme stále po ruce a cokoliv se děje, hned se obratem dozvíme. Na rozdíl od desktopu, který je většinou na jednom místě. Každý uživatel se může rozhodnout, jaká varianta mu vyhovuje lépe. Pro lepší porozumění případům užití byl využit use-case diagram.



Obr. 11: Use-case diagram případu užití aplikace

Zdroj: Autor

Aplikace vyžaduje od uživatele účet, musí se registrovat a následně přihlásit. Po přihlášení se zobrazí seznam vozidel. Pokud ještě nemá žádné vozidlo, může si jej přidat. Jednotlivá vozidla v seznamu může uživatel mazat a upravovat. Dále si může zobrazit více informací o vozidle, kde budou detailnější informace společně

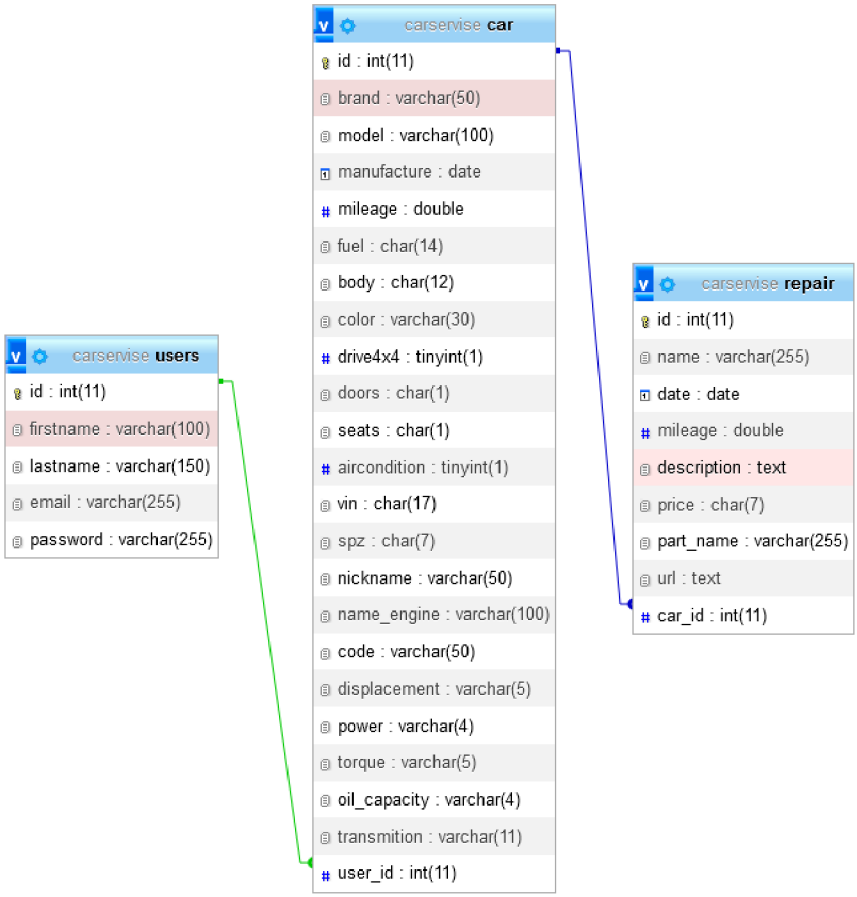
se seznamem oprav. Je zde i formulář pro záznam nové opravy. Jednotlivé opravy může rozkliknout, pro detailnější popis o opravě, kterou prováděl.

Mobilní zařízení najde hlavní využití při opravování vozidla, kdy si uživatel zapíše stav tachometru, opíše číslo dílu a další podrobnosti přímo v dílně a nemusí si vše pamatovat nebo psát na papír a poté zapisovat do počítače.

6.6 Návrh databáze

Aby aplikace mohla fungovat na mobilním a zároveň desktopovém zařízení je potřeba jedna společná databáze. Pro tuto aplikaci byla zvolena databáze MySQL. V ní se nachází celkem tři tabulky.

1. Users – slouží pro ukládání potřebných informací o uživateli jako je jméno, příjmení, email a heslo pro přihlášení.
2. Car – v této tabulce jsou uloženy všechny detailní informace o vozidle a motoru.
3. Repair – poslední tabulka slouží pro ukládání jednotlivých oprav vozila.



Obr. 12: Databázová struktura
Zdroj: Autor

Jednotlivé tabulky jsou propojené pomocí cizích klíčů k identifikaci záznamu v databázi. Vztahy mezi jednotlivými tabulkami je vždy 1:N.

- Tabulka users má vztah 1:N s tabulkou car. Protože jeden uživatel může mít několik vozidel.
- Tabulka car má vztah 1:N s repair, jelikož jedno vozidlo může mít několik oprav.

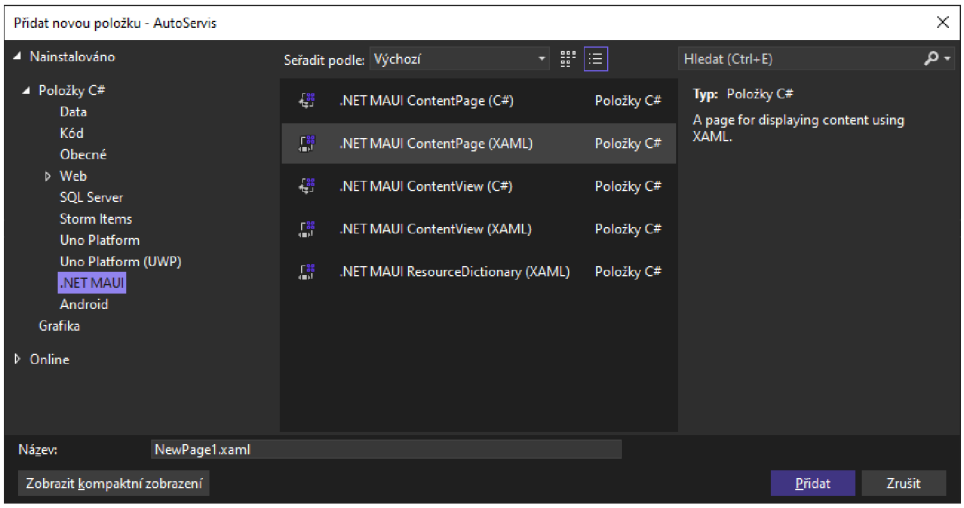
Zároveň toto řešení přináší nevýhodu v podobě nutnosti stálého připojení k internetu z důvodu komunikace s databází. To může být pro uživatele nepříjemné a zároveň to může mírně zpomalit chod aplikace.

7 Implementace vzorové aplikace

Tato část práce se detailněji zaměří na implementaci jednotlivých funkcionalit a komponent v rámci .NET MAUI. Pro lepší pochopení a porozumění budou prezentovány ukázky zdrojových kódů společně s obrázky z uživatelského rozhraní aplikace.

7.1 Vytvoření stránky

Vytvoření stránky je prvním krokem při tvorbě. Aplikace se skládá z mnoha stránek. Díky vývojovému prostředí **Visual Studio** jde pomocí několika kliknutí vytvořit jednotlivé stránky. Vývojář klikne na složku, kam chce novou stránku přidat a vybere možnost **přidat**. Otevře se dialogové okno, ve kterém je pět možností. V tomto projektu se využívají pouze dvě možnosti, a to vždy s využitím XAML jazyka. Po zvolení se vytvoří stránka se základním obsahem a potřebnými informacemi pro její načtení.



Obr. 13: Vytvoření stránky

Zdroj: Autor

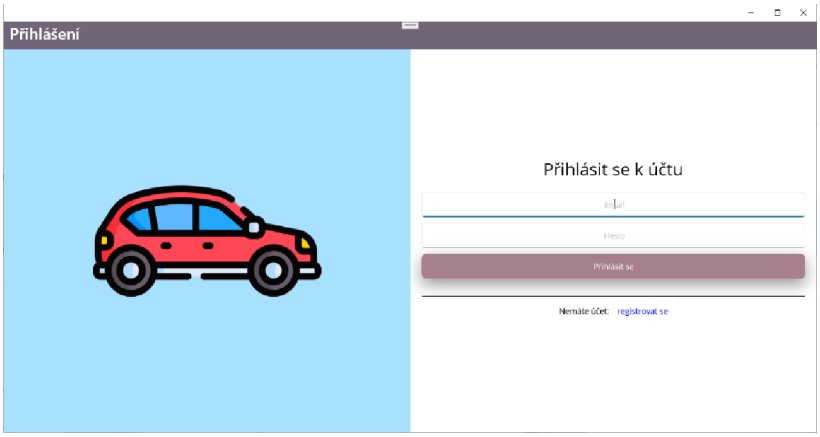
7.1.1 ContentPage

ContentPage je výsledná stránka, která zobrazuje obsah a vzhled uživateli. V XAML souboru se musí definovat hlavička. Kdy první řádek definuje XML verzi dokumentu. Druhý řádek definuje element ContentPage s prvky .NET MAUI a třetí definuje jazyk XAML. Čtvrtý řádek určuje název třídy kódu C# s logikou a obslužnými metodami, a dále propojením na XAML soubor.

```
1   <?xml version="1.0" encoding="utf-8" ?>
2   <ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4                x:Class="AutoServis.Views.Desktop.Pages.Login.DesktopLogin"
5                xmlns:components="clr-namespace:AutoServis.Components.Forms"
6                Title="Přihlášení">
7
8       <Grid>
9           <Grid.ColumnDefinitions...>
13          <Grid.RowDefinitions...>
16          <Grid Grid.Column="0" Grid.Row="0" Background="{StaticResource MyLightBlue}"...>
19          <Grid Grid.Column="0" Grid.Row="1" Background="{StaticResource MyLightBlue}"...>
22
23          <Grid Grid.Column="1" Grid.Row="2" BackgroundColor="White">
24              <ScrollView>
25                  <StackLayout VerticalOptions="CenterAndExpand" HorizontalOptions="CenterAndExpand" MaximumWidthRequest="750">
26                      <components:LoginForm Margin="20"/>
27                  </StackLayout>
28              </ScrollView>
29          </Grid>
30      </Grid>
31  </ContentPage>
```

Obr. 14: Zdrojový kód Content page
Zdroj: Autor

Řádek číslo pět je důležitý z pohledu propojení na jiné soubory nebo zobrazení, které se následně vloží do obsahu stránky. V tomto případě je napojení na komponenty a formuláře. Název za **xmlns:** si vývojář už může určit dle svého. Pomocí tohoto názvu se daná komponenta, která je nejčastěji typem ContentView, následně zavolá pro zobrazení. Zápis načtení takové komponenty je na řádku dvacet šest, kdy se nejprve zadá název, dvojtečka a za ní název komponenty, respektive ContentView. Dále se v elementu ContentPage zadává název stánky.



Obr. 15: Ukázka ContentPage, přihlašovací stránka
Zdroj: Autor

7.1.2 ContentView

ContentView je další možností, která se nabízí při vytváření stránek. Ale nejedná se o plnohodnotnou stránku, ale pouze o pohled, který se následně vkládá do ContentPage. Dobře se kombinuje s funkcí data binding, která umožňuje snadné vložení dat.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AutoServis.Components.Templates.CarRepair"
             x:Name="CarRepairTemplate">

<!-- Komponenty pro ovládání -->

</ContentView>
```

Zdrojový kód 4: ContentView
Zdroj: Autor

Strukturou se příliš neliší od ContenPage, až na název elementu, kdy je to zcela stejné a rozpoznatelné. V C# kódu je napojená třída s jejím názvem. V C# kódu se třída RepairForm dědí z ContentView. Propisují se i do XAML souboru. Pokud by se jednalo o jiný druh pohledu nebo stránky, bude se dědit z něj. Poté následuje konstruktor třídy, v němž se volá metoda InitializeCoponent pro inicializaci komponent. Pod ní pokračuje zbytek třídy a její obslužné události a metody.

```
namespace AutoServis.Components.Forms;

public partial class RepairForm : ContentView
{
       public RepairForm()
       {
              InitializeComponent();
       }
  // Zbytek třídy...
}
```

Zdrojový kód 5: ContentView C# kód
Zdroj: Autor

7.2 Využití data binding v praxi

Data binding je nedílnou součástí při vytváření projektu v .NET MAUI. Pomáhá zajistit komunikaci mezi uživatelským rozhraním a logikou samotné aplikace. Na obrázku níže, je mobilní zobrazení jednotlivých vozidel uživatele. Vzhled seznamu vozidel je téměř vždy stejný. Jediné, co se nejvíce liší, je obrázek, který napomáhá uživateli rozlišit rozdíl mezi vozidly na elektrický a hybridní pohon. Stále stejný vzhled a minimum kódu bez nutného dlouhého opakování zajišťuje šablona s využitím data bindingu.

Obr. 16: Úvodní strana aplikace mobilní zařízení s využitím SwipeView
Zdroj: Autor

Prvním krokem k zprovoznění této funkce je nastavení toku dat do daného XAML souboru. Je nezbytné nastavit **BindingContext**. Aktuální část je tvořená pro mobilní vzhled, a proto je využito zobrazení **SwipeView**, které přidává možnosti potažení prstem s dalšími akcemi. Zde se uživateli zobrazí možnost editace nebo smazání dané položky.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AutoServis.Components.Templates.CarDetail"
             x:Name="this">

    <SwipeView BindingContext="{Binding Source={x:Reference this}}">
        <SwipeView.LeftItems>
            <SwipeItems Mode="Reveal">
                <SwipeItem
                    Text="Smazat"
                    IconImageSource="delete.png"
                    BackgroundColor="LightPink"
                    Invoked="OnDeleteSwipeItemInvoked" />
            </SwipeItems>
        </SwipeView.LeftItems>
        <SwipeView.RightItems>
            <SwipeItems>
                <SwipeItem
                Text="Upravit"
                IconImageSource="edit.png"
                BackgroundColor="LightYellow"
                Invoked="OnEditSwipeItemInvoked" />
            </SwipeItems>
        </SwipeView.RightItems>
```

Obr. 17: Zdrojový kód SwipeView a nastavení data binding přes BindingContext
Zdroj: Autor

Zmíněné SwipeView obsahuje **BindingContext**, který určuje tok dat pro komunikaci mezi uživatelským rozhraním a logikou aplikace. V něm je používáno rozšíření **Binding** a vlastnost source (zdroj). V něm se nachází x:Reference (odkaz) s názvem this. Tento název se zároveň musí uvést v ContentView pod atribut x:Name. Po správném nastavení přichází na řadu **Binding**, který naváže konkrétní vazby k místům, kde mají být zobrazena data nebo určité hodnoty.

```
<Grid Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3" Padding="0">
    <Label Text="{Binding CarName}" FontSize="Large" HorizontalOptions="Center"/>
</Grid>
<Grid Grid.Row="1" Grid.Column="0" Padding="0" Margin="10">
    <Image Source="{Binding CarImage}" Aspect="AspectFit" MaximumWidthRequest="80"/>
</Grid>
<Grid Grid.Row="1" Grid.Column="1" Padding="0" Margin="10">
    <VerticalStackLayout>
        <Label Text="{Binding CarManufacture}"/>
        <Label Text="{Binding CarMileage}"/>
        <Label Text="{Binding CarColor}"/>
        <Label Text="{Binding CarSPZ}"/>
        <Label Text="{Binding CarId}" IsVisible="false"/>
    </VerticalStackLayout>
</Grid>
<Grid Grid.Row="1" Grid.Column="2" Padding="0" Margin="10">
    <VerticalStackLayout>
        <Label Text="{Binding CarFuel}"/>
        <Label Text="{Binding CarEngineName}"/>
        <Label Text="{Binding CarPower}"/>
        <Label Text="{Binding CarGearbox}"/>
    </VerticalStackLayout>
</Grid>
<!-- Mobilní zařízení -->
<Grid Grid.Row="2" Grid.ColumnSpan="3" Padding="0" Margin="0" IsVisible="{OnPlatform False, Android=True, iOS=True}">
    <Button
        Text="Více info"
        Margin="5"
        Background="{x:StaticResource MySecondPrimary}"
        Clicked="MoreCarInfoClicked"/>
</Grid>
<!-- Pro počítač -->
<Grid Grid.Row="2" Grid.Column="0" Padding="0" Margin="0" IsVisible="{OnPlatform True, Android=False, iOS=False}">
```

Obr. 18: Nastavení data binding
Zdroj: Autor

Pod **SwipeView** se nachází obsah s informacemi o vozidle. V tomto obsahu jsou jednotlivá textová pole (labely), které mají vlastnost **Text= {Binding název}**. Vlastnost **Text** definuje obsah, který se má zobrazit na obrazovce. Vzhledem k tomu, že tento text bude vždy jiný na základě uložených dat v databázi, je zde přiřazen **Binding**. Ten nastaví vazbu na C# kód, ve kterém jsou atributy pro konkrétní hodnoty. Tyto atributy mají pak vlastnost **get** pro získání hodnoty a **set** pro nastavení.

```
public static readonly BindableProperty CarNameProperty =
BindableProperty.Create(nameof(CarName), typeof(string), typeof(CarDetail),
string.Empty);


public string CarName
{
    get => (string)GetValue(CarNameProperty);
    set => SetValue(CarNameProperty, value);
}
```

Zdrojový kód 6: Binable proměnné a její vlastnosti
Zdroj: Autor

Po nastavení data bindingu v XAML souboru a v C# kódu se následně pokračuje do místa s využitím dané šablony. V tomto místě se nejdříve načtou data z databáze do seznamu (listu) a z něj následně do šablony. Po nastavení data binding v XAML souboru a v C# kódu se následně pokračuje do místa s využitím dané šablony. V tomto místě se nejdříve načtou data z databáze. Aplikace vytvoří instanci třídy API, zkontroluje, zda je připojení na internet. Pokud ano, pokračuje dál k vytvoření HTTP GET požadavku na API pro seznam vozidel s id uživatele. Následuje kontrola HTTP požadavku, zda byl úspěšný. V případě, že ne, vypíše se uživateli o tom hláška. V opačné situaci se vrací úspěšná odpověď 2xx a pokračuje se na uložení dat do textového řetězce. Ještě před ním se nastavuje možnost pro deserializaci JSON dat. V tomto případě jde o vlastní boolovský konvertor. Předposledním krokem je desirializace řetězce dat JSON, do seznamu objektů typu car. Na závěr se volá funkce **ShowUserCar()**, která vykreslí seznam objektů pomocí šablony.

```
private async void LoadUserCars()
{
    API api = new API();

    if (api.checkConnectivity())
    {
        await DisplayAlert("Oznámení", "Není připojení k internetu. Proto nemohli být načteny data", "Zavřít");
        return;
    }

    HttpResponseMessage responseMessage = await api.client.GetAsync($"car/listcaruser?id={user.id}");
    if (responseMessage.IsSuccessStatusCode)
    {
        JsonSerializerOptions options = new JsonSerializerOptions
        {
            Converters = { new BoolConverter() },
        };
        string getResponseString = await responseMessage.Content.ReadAsStringAsync();
        cars = JsonSerializer.Deserialize<List<Car>>(getResponseString, options);

        ShowUserCars();
    }
    else
    {
        await DisplayAlert("Oznámení", "Při načítání dat z databáze došlo k chybě.", "Zavřít");
    }
}
```

Obr. 19: Zdrojový kód čtení dat z databáze pomocí API
Zdroj: Autor

Danou šablonu je potřeba přidat do hlavního zobrazení. Zde mohou být zobrazeny i jiné komponenty. Nejdříve je potřeba smazat všechny potomky. Poté následuje cyklus, který čte jednotlivé objekty seznamu. V cyklu se vytvoří objekt CarDetail a do složených závorek se napíšou atributy, kterým se přiřadí konkrétní hodnoty o vozidle. Následně se tento objekt (šablona) přidá jako potomek do verticalView. Po skončení cyklu se přidá tlačítko, které umožňuje přidání nového vozidla.

```
// Zobrazení jednotlivých vozidel uživatele
Počet odkazů: 3
private void ShowUserCars()
{
    verticalLayout.Children.Clear();
    int margin = 0;
    if (DevicePlatform.WinUI == DeviceInfo.Platform) margin = 10;
    foreach (Car car in cars)
    {
        string carImage = "gas_car.png";
        if (car.fuel == "Elektro" || car.fuel == "Hybrid") carImage = "electro_car.png";
        var carView = new CarDetail
        {
            Padding = margin,
            MaximumWidthRequest = 1000,
            CarId = car.id,
            CarName = $"{car.brand} {car.model}",
            CarManufacture = car.manufacture.ToShortDateString(),
            CarSPZ = car.spz,
            CarColor = car.color,
            CarMileage = car.mileage.ToString(),
            CarFuel = car.fuel,
            CarEngineName = car.name_engine,
            CarPower = car.power,
            CarGearbox = car.transmition,
            CarImage = carImage,
        };
        verticalLayout.Children.Add(carView);
    }
    verticalLayout.Children.Add(addNewCarButton);
}
```

Obr. 20: Zdrojový kód přiřazování hodnot do šablony
Zdroj: Autor

7.2.1 Nahrání dat bez data binding

Data binding je primární cesta, jak dostat data do popisů, vstupních polí a dalších komponent. Mimo jiné může fungovat na principu aktualizace dat ze vstupního pole do dalších komponent. Kromě využití data binding se dá přistupovat ke komponentám pomocí jejich názvu x:Name, který je uveden ve vlastnostech v XAML souboru.

```
<!-- XAML kód -->
<VerticalStackLayout
    x:Name="LoadingIndicator"
    HorizontalOptions="Center"
    Margin="5, 12"
    IsVisible="false">
    <ActivityIndicator
        IsRunning="True"
        Color="{x:StaticResource MySecondPrimary}"
        HeightRequest="70"
        WidthRequest="70"/>
    <Label Text="Probíhá ukládání dat..."
           FontSize="Medium"
           Margin="5, 20"/>
</VerticalStackLayout>


// C# kód zobrazení ActivityIndicator s Label
LoadingIndicator.IsVisible = true;
```

Zdrojový kód 7: Použití x:Name
Zdroj: Autor

Pomocí tohoto názvu se dá v .cs souboru přistupovat k dané komponentě a přiřazovat nebo nastavovat další vlastnosti. Toto funguje, pokud jsou ve stejné hierarchii. Pokud by bylo potřeba do nějaké komponenty nahrát data z jiné části, musí se tyto komponenty vyhledat a načíst do proměnných, než se s nimi bude dále pracovat. Po nalezení dané komponenty se vybere vlastnost dle potřeby a doplní se potřebná data z třídy Car. Vzhledem k tomu, že je potřeba doplnit text do textových polí využije se vlastnosti **Text**. U různých komponent se toto může lišit. Jako například u **Switch** je pro zaškrtnutí vlastnost **IsToggled**, tak u **CheckBox** je pro zaškrtnutí **IsChecked**. Větší problém je u **Picker**, kde se musí vyhledat item, který byl zvolen a jeho Index. Ten určí, která položka má být zobrazena. U **RadioButton** to funguje obdobným způsobem.

```
// Načtení inputů
Entry brandInput = (Entry)NewCarForm.FindByName("brandInput");
CheckBox checkbox_4x4 = (CheckBox)NewCarForm.FindByName("checkbox_4x4");
Picker fuelPicker = (Picker)NewCarForm.FindByName("fuelPicker");


// Nastavení hodnot inputům
brandInput.Text = car.brand;
checkbox_4x4.IsChecked = car.drive4x4;
SetPicker(fuelPicker, car.fuel);


// Vybrání správné hodnoty v Pickeru
private void SetPicker(Picker picker, string item)
{
    if (picker.ItemsSource is IList<string> fuelItems)
    {
        int index = fuelItems.IndexOf(item);

        // Zkontrolujte, zda hodnota byla nalezena ve zdroji dat
        if (index != -1)
        {
            picker.SelectedIndex = index;
        }
    }
}
```

Zdrojový kód 8: Načtení komponent a vybrání hodnoty v Picker
Zdroj: Autor

7.3 Uživatelské rozhraní

Uživatelské rozhraní se dá tvořit více způsoby. Nejjednodušší způsob je univerzální vzhled aplikace, který bude zcela stejný na každé platformě. Při jeho použití se nemusí využít všechny nabízené možností dané platformy. Dalším možným přístupem je vytvoření designu pro každou platformu zvlášť. Tento způsob může být zdlouhavý a tím i nákladný. Ale lze u něj využít možnosti komponent, které jsou stejné pro všechny operační systémy, jako jsou formuláře, pomůže to výrazně usnadnit vývoj. Navzdory lehce odlišným vzhledům se bude aplikace stále chovat stejně. Aplikace má jednotlivá Views, která jsou stejná pro všechna zařízení, nebo rozdělena na desktopové a mobilní zařízení. Tím se odlišují jednotlivé vzhledy aplikace.

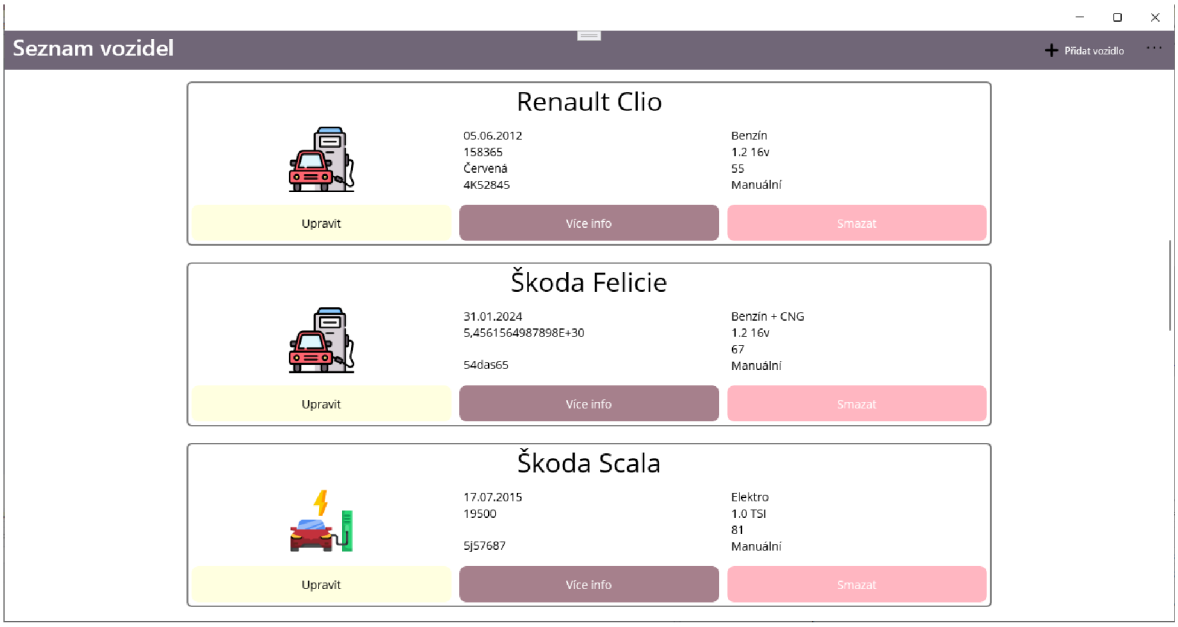
```
<!-- Mobilní zařízení -->
<Grid Grid.Row="2" Grid.ColumnSpan="3" Padding="0" Margin="0"
IsVisible="{OnPlatform False, Android=True, iOS=True}">
    <Button
        Text="Více info"
        Margin="5"
        Background="{x:StaticResource MySecondPrimary}"
        Clicked="MoreCarInfoClicked"/>
</Grid>
<!-- Pro počítač -->
<Grid Grid.Row="2" Grid.Column="0" Padding="0" Margin="0" IsVisible="{OnPlatform
True, Android=False, iOS=False}">
    <Button
        Text="Upravit"
        TextColor="Black"
        Margin="5"
        BackgroundColor="LightYellow"
        Clicked="OnEditSwipeItemInvoked"/>
</Grid>
<Grid Grid.Row="2" Grid.Column="1" Padding="0" Margin="0" IsVisible="{OnPlatform
True, Android=False, iOS=False}">
    <Button
        Text="Více info"
        Margin="5"
        Background="{x:StaticResource MySecondPrimary}"
        Clicked="MoreCarInfoClicked"/>
</Grid>
<Grid Grid.Row="2" Grid.Column="2" Padding="0" Margin="0" IsVisible="{OnPlatform
True, Android=False, iOS=False}">
    <Button
        Text="Smazat"
        Margin="5"
        BackgroundColor="LightPink"
        Clicked="OnDeleteSwipeItemInvoked"/>
</Grid>
```

Zdrojový kód 9: Ukázka OnPlatform ve vlastnosti IsVisible
Zdroj: Autor

Není to jediný způsob, jak pomoci odlišit vzhled aplikace. V designovacím jazyce XAML je rozšíření pro vlastnosti **OnPlatform**. Toto rozšíření, které je přímo zabudované v tomto jazyce umožňuje měnit vzhled u jednotlivých komponent. Lze u něj nastavit například odsazení vnější (Margin) nebo vnitřní (Padding), viditelnost a další vlastnosti. Po zvolení vlastnosti se určí platforma. Pokud se nezadá nic nebo default, definuje se tím výchozí hodnota. Další platformy jsou iOS, Android a pro Windows je to WinUI.

Úvodní strana pro mobilní zařízení využívá komponenty SwipeView obr. 16, pro potažení do boku. Při potažení do jedné ze stran se zobrazí tlačítko pro editaci nebo pro smazání vozidla. Tato možnost se však běžně u monitorů pro desktopová zařízení nenachází, i když u spousty notebooků už je. Proto je zapotřebí úvodní stranu rozšířit o dvě další tlačítka, která umožní uživatelům na počítači, taktéž editovat a mazat vozidla podle své potřeby. Tyto komponenty jsou zároveň uložené v jedné šabloně.

To napomáhá obsluze metod pro tlačítka, není potřeba psát programový kód pro každé tlačítko zvlášť.



Obr. 21: Úvodní strana aplikace desktopové zařízení
Zdroj: Autor

7.3.1 SwipeView

SwipeView je komponenta pro zařízení s dotykovou obrazovkou a nezáleží na jakém operačním systému aplikace bude spuštěna. Komponenta zpříjemňuje používání aplikace, protože se v ní nemusí nacházet další tlačítka. Využívá se především u mobilních zařízení, kde jsou malé obrazovky a dají se ovládat gesty. Swipeview má jednotlivé Itemy. Ty se dále rozdělují na levou a pravou stranu, záleží, na jakou stranu vývojář chce, aby uživatel táhnul. V tomto případě se využívají obě strany. Mode u SwipeItems má dva režimy: Reveal a Execute, ten pro změnu hned po potažení vykoná danou akci. Dále se do SwipeItem nadefinují vlastnosti, které ovlivní vzhled při potažení do strany. Důležitý je Invoked, který vyvolá metodu, která se má vykonat. V případě potřeby se dá daná metoda zavolat i z tlačítka, ale zde už z Clicked.

```
<SwipeView BindingContext="{Binding Source={x:Reference this}}">
    <SwipeView.LeftItems>
        <SwipeItems Mode="Reveal">
            <SwipeItem
                Text="Smazat"
                IconImageSource="delete.png"
                BackgroundColor="LightPink"
                Invoked="OnDeleteSwipeItemInvoked" />
        </SwipeItems>
    </SwipeView.LeftItems>
    <SwipeView.RightItems>
        <SwipeItems>
            <SwipeItem
            Text="Upravit"
            IconImageSource="edit.png"
            BackgroundColor="LightYellow"
            Invoked="OnEditSwipeItemInvoked" />
        </SwipeItems>
    </SwipeView.RightItems>
<!-- Volání stejné metody jako ve SwipeView -->
<Button
    Text="Smazat"
    Margin="5"
    BackgroundColor="LightPink"
    Clicked="OnDeleteSwipeItemInvoked"/>
```

Zdrojový kód 10: SwipeView
Zdroj: Autor

7.3.2 Lišta na úvodní straně

Lišta na horní straně obrazovky vždy nese název podle stránky, které je právě otevřená. Tento název se může odstranit a nemusí se tam vůbec nacházet. Pro uživatele to ale přináší jednoduchou orientaci, kde právě je. V horní liště se mohou nacházet i další položky pro obsluhu, a to pomocí ToolBarItems, které se přidají do aktuální stránky. V něm se definují za pomoci ToolBarItem jednotlivé itemy s vlastnostmi. Jeho řazení se dělí na dva základní: Default/Primary a Secondary. Výchozí zobrazí tlačítko pro kliknutí. Na desktopu zobrazí ikonu a text. Na mobilní pouze ikonu bez textu. Druhý způsob Secondary, ukáže na konci obrazovky tři tečky. Pod nimi se nachází menu, které zobrazí další itemy, ty se pak už řadí podle vlastnosti Priority. Každý item má obslužnou událost Clicked, která se vykoná po kliknutí na daný prvek. První item aktualizuje seznam vozidel na úvodní stránce. Z databáze si stáhne data, ty znovu zobrazí uživateli. Druhý otevře novou stránku a zobrazí formulář pro přidání nového vozidla. To samé vykoná i třetí jen s rozdílem, že zobrazí na nové stránce informace o aplikaci. Čtvrtý otevře stránku pro editaci uživatelských dat a poslední zavolá vyskakovací okénko s hláškou, jestli se uživatel chce opravdu odhlásit.

```
<ContentPage.ToolbarItems>
    <ToolbarItem
        Text="Aktualizovat seznam"
        IconImageSource="refresh.png"
        Clicked="refreshContentClick"/>
    <ToolbarItem
        Text="Přidat vozidlo"
        IconImageSource="add.png"
        Clicked="addNewCarClick"/>
    <ToolbarItem
        Text="O aplikaci"
        Order="Secondary"
        Clicked="AboutAppClick"
        Priority="0"/>
    <ToolbarItem
        Text="Uživatel"
        Order="Secondary"
        Clicked="userInfoClick"
        Priority="1"/>
    <ToolbarItem
        Text="Odhlásit se"
        Clicked="LogOut"
        Order="Secondary"
        Priority="2"/>
</ContentPage.ToolbarItems>
```

Zdrojový kód 11: ToolbarItems
Zdroj: Autor

7.3.3 Tlačítko

Většina událostí a formulářů je napojená na kliknutí tlačítka, které vykoná očekávanou úlohu v obslužné metodě. Tlačítko se příliš od ostatních komponent neliší. Jeho oživení a zprovoznění není nic složitého.

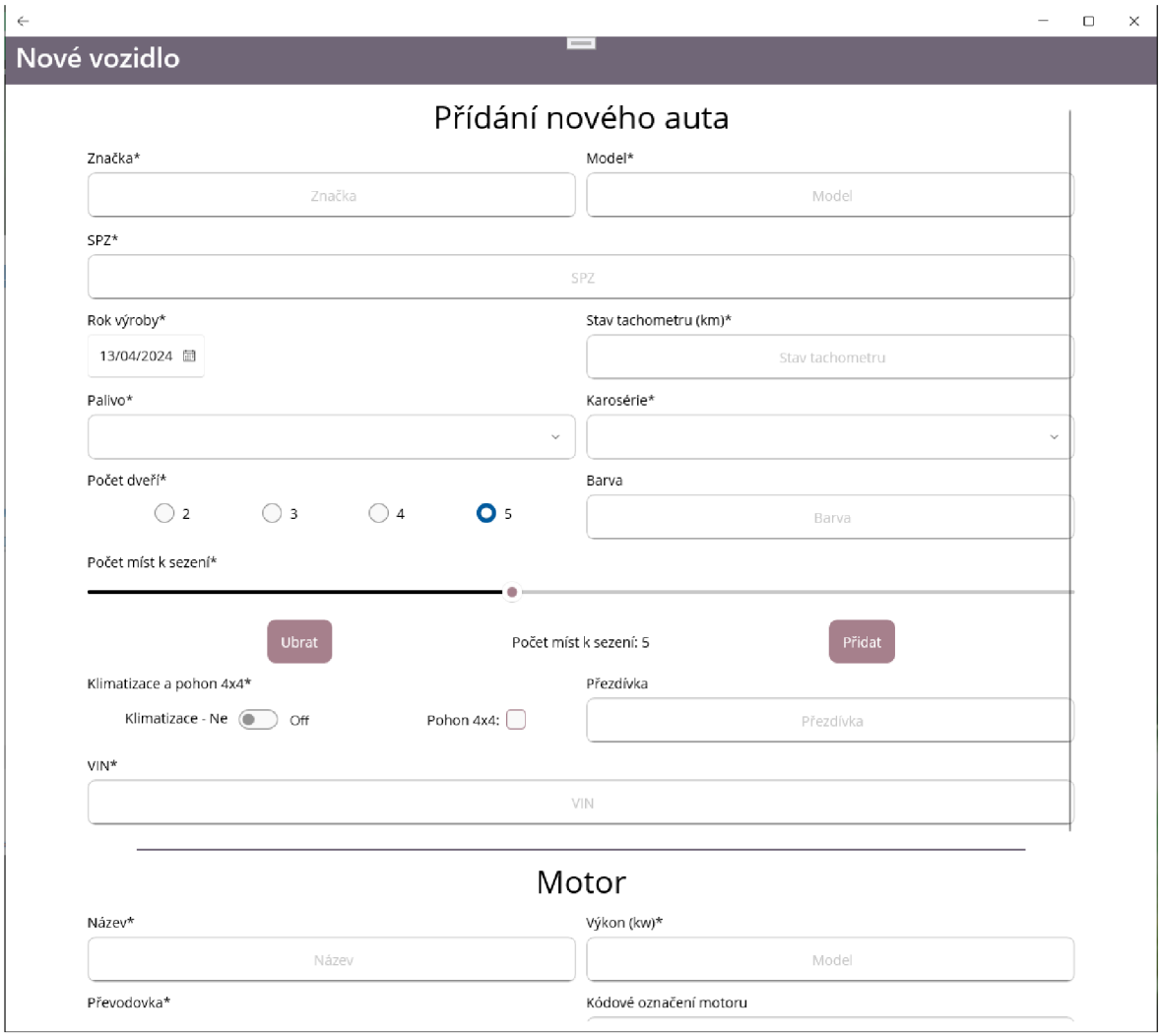
```
<Button x:Name="BtnEndForm"
        Text="Uložit opravu"
        Clicked="saveRepairCar_Clicked"
        Margin="0,20,0,30"
        BackgroundColor="{x:StaticResource MySecondPrimary}">
    <Button.Shadow>
        <Shadow Brush="Black" Offset="2,5" Radius="3" Opacity="0.3" />
    </Button.Shadow>
</Button>
```

Zdrojový kód 12: Tlačítka
Zdroj: Autor

Definuje se element button a v něm potřebné vlastnosti a události. Nejzajímavější je třetí řádek, ve kterém je událost Clicked, která volá obslužnou metodu při kliknutí. Řádek pod ním určuje vnější odsazení prvku. V tlačítku se nachází pod element Button.Shadow, který umožní vytvořit stín pod tlačítkem elementem Shadow. Tento určuje barvu, posunutí na ose x a y od středu tlačítka, poloměr rozmazání a průhlednost stínu.

7.4 Formulář

Formuláře jsou nedílnou součástí každé aplikace. Nachází se úplně všude. Nejzákladnější formulář je pro registraci a přihlášení, následují další k vyplnění dat. V této aplikaci je jeden velký formulář, který obsahuje nespočet textových polí k vyplnění údajů o vozidle, které používá k dopravě. Formulář obsahuje jednotlivé komponenty, které budou následně popsány.



Obr. 22: Formulář pro nové vozidlo na desktopové zařízení
Zdroj: Autor

7.4.1 Entry

Jedná se o textové pole, do kterého uživatel vyplňuje jednotlivé informace. Vstupní pole obsahuje velké množství vlastností, jako je barva textu nebo placeholderu, velikost písma, rozlišení, zda má být vše malými nebo velkými písmeny, nastavení Cursoru, mezer a apod. Je důležité u něj určit název. Zároveň lze na konci zobrazit tlačítko pro vymazání pole. V tomto případě je to nastavené pouze pokud je uživatel kliknutý

v tomto poli a edituje ho. Další zajímavé položky ovlivní spíše mobilní zařízení, kde není fyzická klávesnice. Tyto položky jsou Keyboard a ReturnType. Keyboard určuje, jaká klávesnice se má uživateli zobrazit. Zda defaultní, na email, telephone pro telefonní čísla, nebo Numeric pouze pro číselnou klávesnici. ReturnType určuje co má následovat po vyplnění daného vstupního pole. Pokud je hodnota na Next nebo Go, pokračuje na další pole. V případě klíčového slova Done, se místo šipky zobrazí v pravém dolním rohu zatržítko a zavře se klávesnice. Je dobré zmínit, že Entry může obsahovat i metody TextChanged, který se volá při stisku klávesy nebo Completed, ten se zavolá po dokončení editace a vykoná danou metodu.

```
<Entry
 x:Name="mileageInput"
 Placeholder="Stav tachometru"
 Keyboard="Numeric"
 ReturnType="Next"
 ClearButtonVisibility=
"WhileEditing"
 HorizontalTextAlignment="Center"
 TextChanged="OnEntryTextChange"
 MaxLength="12"
 TextTransform="Lowercase"/>
```

```
private void OnEntryTextChange(
    object sender,
    TextChangedEventArgs e)
{
 string text = ((Entry)sender).Text;
 string result = "";
 for(int i = 0; i < text.Length; i++)
 {
  if (text[i] >= '0' && text[i] <= '9'
  || text[i] == ' ') result += text[i];
 }
 ((Entry)sender).Text = result;
}
```

Zdrojový kód 13: Textové pole Entry s obslužnou metodou
Zdroj: Autor

7.4.2 Picker

Picker, často označovaný jako Select v HTML. Slouží k výběru položek z definovaného seznamu. Tento Picker má jednotlivé položky v ItemSource a v něm definované pole s možnostmi k výběru. Ty jsou pak uvedeny jako textový řetězec.

```
<!-- Kód v XAML -->
<Picker
x:Name="fuelPicker"
HorizontalTextAlignment="Center">
    <Picker.ItemsSource>
        <x:Array Type="{x:Type x:String}">
            <x:String>Benzín</x:String>
            <x:String>Diesel</x:String>
            <x:String>Benzín + LPG</x:String>
            <x:String>Benzín + CNG</x:String>
            <x:String>Hybrid</x:String>
            <x:String>Elektro</x:String>
        </x:Array>
    </Picker.ItemsSource>
</Picker>

// Programový kód
if (fuelPicker.SelectedIndex == -1 || bodyPicker.SelectedIndex == -1)
{
    App.Current.MainPage.DisplayAlert("Oznámení", "Nebyly vyplněny všechny
údaje", "Ok");
    return;
}

string fuel = fuelPicker.ItemsSource[fuelPicker.SelectedIndex].ToString();
```

Zdrojový kód 14: Picker
Zdroj: Autor

Následný výběr položky v tomto seznamu je už jednoduchý. Zkontroluje se, zda byla v Pickeru vybrána položka pomocí vlastnosti SelectedIndex. Pokud by nebyla, bude vracet vybraný index -1. Pokud byla, stačí vzít ItemSource Pickeru a vybrat položku z pole pomocí již zmíněného SelectedIndex.

7.4.3 Slider

Slider, také známý jako **posuvník**, slouží k výběru číselné hodnoty pomocí tažení doleva nebo doprava. Při použití slideru se musí nastavit minimální a maximální hodnota, pro stanovení hranic při výběru čísla. **Value** určuje výchozí hodnotu, která bude zobrazena na posuvníku pomocí kuličky. Dále je možné ještě nastavit barvu pro čáru nebo kuličku na posuvníku.

Využití této komponenty je poměrně snadné, ale má určité nedostatky. První z nich je, že neposkytuje zobrazení aktuální hodnoty, což je z uživatelského hlediska dost nepraktické. Proto je skoro nutností tuto komponentu rozšířit o **Label**, který bude zobrazovat aktuální hodnotu. To se řeší pomocí události ValueChanged, která se volá při změně hodnoty na posuvníku.

```
<Slider
x:Name="sliderSeat"
Minimum="2"
Maximum="9"
Value="5"
MinimumTrackColor="Black"
ThumbColor="{x:StaticResource MySecondPrimary}"
ValueChanged="OnSliderValueChanged" />
```

Zdrojový kód 15: Slider
Zdroj: Autor

Druhou nedokonalostí je nemožnost nastavit celá čísla v rámci vlastností. Proto se budou při použití slideru zobrazovat desetinná čísla. Toto je nevhodné například u určování počtu dveří. Zde je potřeba pracovat pouze s celými čísly. Toto se řeší pomocí události OnSliderValueChanged, kde se načte nová hodnota z parametru **e.NewValue**. Tato hodnota se ihned zaokrouhlí pomocí funkce **Math.Round**. Může se použít i zaokrouhlování směrem nahoru nebo dolů, ale vhodnější je symetrické zaokrouhlení, které zaokrouhluje na obě strany. Po zaokrouhlení se uloží nová hodnota do slideru a Labelu.

```
private void OnSliderValueChanged(object sender, ValueChangedEventArgs e)
{
    double value = Math.Round(e.NewValue);
    sliderSeat.Value = value;
    displayLabel.Text = $"Počet míst k sezení: {value}";
}
```

Zdrojový kód 16: Obslužná událost Slideru
Zdroj: Autor

7.4.4 DatePicker

DatePicker je sice jednoduchá komponenta, ale má svůj účel. Slouží k výběru datumu. Vzhledově je podobná komponentě Entry. Při konfiguraci v jazyce XAML stojí za zmínku nastavení rozsahu datumu, od kdy do kdy může uživatel vybírat. Následně je potřeba určit formát zobrazení datumu. Pokud nebude formát nastaven bude využit výchozí Americký styl zápisu datumu. V C# kódu pak pomocí .Date se získá datum s datovým typem DateTime.

```
<!-- DatePicker v XAML -->
<DatePicker x:Name="manufactureDate" HorizontalOptions="FillAndExpand"
Format="dd/MM/yyyy"/>
// Programový kód v C#
DateTime manufature = manufactureDate.Date;
```

Zdrojový kód 17: DatePicker
Zdroj: Autor

7.4.5 Switch

Switch slouží jako přepínací tlačítko. Jedná se v podstatě o variantu checkboxu jen s odlišným vzhledem. Nabývá pouze hodnot true nebo false, a to z vlastnosti IsToggled. Switch obdobně jako Slider nemá textové doplnění, tedy přesněji má, ale pouze na zařízení windows, kde se ukazuje on a off. Na zařízeních s jiným operačním systémem se tento text nenachází a je potřeba tento prvek doplnit o Label s textem. Událost Toggled reaguje na stisk komponenty a v tomto případě nastavuje hodnotu Labelu s informací pro uživatele. Opět je zde nedostatek kdy se text on, off za komponentou nedá jednoduše odstranit.

```
<!-- DatePicker v XAML -->
<Switch x:Name="airConditioningSwitch" Toggled="SwitchChange"/>
// Programový kód v C#
bool airCondition = airConditioningSwitch.IsToggled;
```

Zdrojový kód 18: Switch
Zdroj: Autor

7.4.6 RadioButton

RadioButton jsou tlačítka vždy pro výběr maximálně jedné možnosti ze seznamu. Zvykem bývá, že u jednoho z nich je nastavená vlastnost IsChecked na true. Komponentu je potřeba dát do skupiny pomocí GroupName u jednotlivých radioButtonů nebo podobným způsobem u nadřazeného prvku.

```
<RadioButton
    Content="Manuální"
    GroupName="gearboxRB"
    CheckedChanged="gearboxChange"
    IsChecked="true"/>
<RadioButton
    Content="Automatická"
    GroupName="gearboxRB"
    CheckedChanged="gearboxChange"/>
```

Zdrojový kód 19: RadioButtonu
Zdroj: Autor

Chyby některých předešlých komponent zmizely, ale objevily se další problémy. A tím je získání hodnoty z této skupiny. Protože **GroupName** nefunguje stejným způsobem jako **x:Name**, nedá se zeptat v C# kódu jednoduchým příkazem „Dej mi zvolený radiobutton a jeho content.“ Pro získání aktuální hodnoty se využije události CheckChanged.

```
private void gearboxChange(object sender, CheckedChangedEventArgs e)
{
    if (sender is RadioButton radioButton && e.Value)
    {
        transmition = radioButton.Content?.ToString();
    }
}
```

Zdrojový kód 20: Obslužná metoda RadioButtonu
Zdroj: Autor

Object sender reprezentuje odesílatele události a proměnná **CheckedChangedEventArgs** informuje o stavu RadioButtonu. Poté se v podmínce kontroluje, zda je **sender** instancí třídy RadioButton a zároveň je **true** pomocí **e.Value**. Pokud podmínka byla splněna, uloží se do globální proměnné **transmition** z obsahu radiobuttonu. Operátor**?** se používá pro bezpečný přístup k obsahu, což znamená, že pokud je obsah **null**, vrátí se **null** a nebude vyvolána žádná výjimka.

7.4.7 AktivityIndicator

ActivityIndicator je velice jednoduchý prvek. Jeho klíčovou vlastností je zobrazit načítací kolečko uživateli. U tohoto kolečka se dá měnit barva, jeho velikost a zda se má točit. Alternativou, k tomuto prvku by byl progressBar, ten je však složitější na implementaci, pokud je účelem zobrazit přesné trvání vykonávané aktivity.

Obr. 23: Formulář pro vozidlo mobilní verze s ukázkou ActivityIncidatoru
Zdroj: Autor

AktivityIndicator se zobrazí po stisknutí tlačítka, dává tak uživateli informaci o zpracovávání dat z formuláře. Tyto data se načtou z jednotlivých polí a uloží do proměnných, vytvoří se objekt, který se následně pošle pomocí HttpClientu jako post nebo put, podle toho, zda uživatel vytváří nové vozidlo nebo aktualizuje informace u stávajícího.

7.5 Uložení dat z formuláře do databáze

Po vyplnění všech povinných údajů následuje zpracování dat z formuláře a jejich uložení do databáze. Na začátku se zkontrolují vyplněné povinné položky formuláře. Následuje kontrola textového pole, kde jsou po uživateli vyžadovány pouze číselné hodnoty. Pod touto podmínkou se nachází kontrola komponenty picker. Pokud by uživatel ze seznamu nevybral žádnou položku vrátí picker -1 a bude upozorněn dialogovým okénkem. Po provedení všech kontrol se uloží jednotlivá data do proměnných podle jejich názvu.

```
private async void newCarClick(object sender, EventArgs e)
{
    if (IsInputEmpty(brandInput.Text) &&
        IsInputEmpty(modelInput.Text) &&
        IsInputEmpty(vinInput.Text) &&
        IsInputEmpty(spzInput.Text) &&
        IsInputEmpty(nameEngineInput.Text) &&
        IsInputEmpty(powerInput.Text))
    {
        App.Current.MainPage.DisplayAlert("Oznámení", "Nějaký z povinných údajů nebyl vyplněn.", "Ok");
        return;
    }
    // Kontrola zda bylo do vstupního pole zadané pouze číslo
    double mileage = 0;
    if (!Double.TryParse(mileageInput.Text, out mileage))...

    // Kontrola zda byl zvolen prvek v komponentě picker
    if (fuelPicker.SelectedIndex == -1 || bodyPicker.SelectedIndex == -1)...

    string brand = brandInput.Text.Trim();
    string model = modelInput.Text.Trim();
    DateTime manufature = manufactureDate.Date;
    string fuel = fuelPicker.ItemsSource[fuelPicker.SelectedIndex].ToString();
    string body = bodyPicker.ItemsSource[bodyPicker.SelectedIndex].ToString();
    string color = "";
    if (!IsInputEmpty(colorInput.Text)) color = colorInput.Text.Trim();
```

Obr. 24: Zpracování dat z formuláře první část
Zdroj: Autor

Dále se pokračuje zobrazením načítacího kolečka a skrytím tlačítka pro odeslání formuláře. Tímto způsobem se zamezí opakovanému klikání na tlačítko a zároveň uživatel vidí, že aplikace vykonává nějakou činnost. Vytvoří se instance třídy API a zkontroluje se připojení k internetu. Pokud připojení není, zobrazí se uživateli hláška, aby své zařízení připojil k sítí. Pokračuje se vytvořením instance třídy Car s daty, která byla uložena do jednotlivých proměnných. Instance má první parametr (id) -1, protože se jedná o nové vozidlo. Následně se nalezne rodičovská stránka MobileNewCar pro případné uložení dat do listu na hlavní stránce se seznamem vozidel. Poté se data o vozidle odešlou pomocí HTTP POST požadavku na cílovou adresu „car/create“ společně s objektem car. Následně proběhne zpracování požadavku a čeká se na odpověď serveru. Pokud je odpověď úspěšná (kód 2xx), bude uživatel informován a automaticky přesměrován zpět na úvodní stranu. V případě neúspěchu ukládání se informuje uživatel a zobrazí se mu opět tlačítko, aby mohl data znovu odeslat.

```
    // C# kód zobrazení ActivityIndicator s Label
    LoadingIndicator.IsVisible = true;
    BtnEndForm.IsVisible = false;

    API api = new API();
    if (api.checkConnectivity())...

    Car car = new Car(-1, brand, model, manufature, mileage, fuel, body,
        color, drive4x4, doors, seats, airCondition, vin, spz, nickname,
        name_engine, code, displacement, power, torque, oil_capacity, transmition,
        id);
    HttpResponseMessage response;
    /* Nalezení rodičovské stránky,
     * po uspěšném uložení do db se zavolá metodu SaveToList
     * Ta uloží nová data do listu na úvodní stráně */
    MobileNewCar parentPage = FindParentMobileNewCar(this);

    response = await api.client.PostAsJsonAsync("car/create", car);
    if (response.IsSuccessStatusCode)
    {
        // Přidání nového vozidla do listu
        if (parentPage != null)
        {
            parentPage?.SaveToList(null, true);
        }
        await App.Current.MainPage.DisplayAlert("Oznámení", "Vozidlo bylo úspěšně vytvořeno." +
            "\n\nPo stisku tlačítka budete přesměrování na úvodní stránku 😊", "OK");
        await Navigation.PopAsync();
    }
    else App.Current.MainPage.DisplayAlert("Chyba", "Nastala neočkávaná chyba. Zkus se to znovu", "Ok");
    LoadingIndicator.IsVisible = false;
    BtnEndForm.IsVisible = true;
}
```

Obr. 25: Zpracování dat z formuláře druhá část
Zdroj: Autor

7.5.1 Aktualizace dat

V případě aktualizace dat se použije HTTP PUT požadavek a zavolá se metoda PutAsJsonAsync. Do ní se umístí dva parametry. Prvním je adresa požadavku na server a druhým instance objektu car. Zároveň tyto metody klienta API rovnou serializují data do formátu JSON, což eliminuje potřebu provádět tento proces ručně před odesláním.

```
HttpResponseMessage response = await api.client.PutAsJsonAsync("car/update", car);
```

Zdrojový kód 21: HTTP PUT požadavek na server
Zdroj: Autor

7.5.2 Mazání dat

Mazání dat probíhá velice podobným způsobem. Aplikace se uživatele dotáže pomocí dialogového okénka, zda chce danou položku opravdu smazat. Toto ošetření je běžnou praxí, protože by se jinak mohlo stát, že si uživatel omylem smaže záznam, který nechce.

```
// Kontrola zda chce opravdu daný záznam smazat
bool answer = await App.Current.MainPage.DisplayAlert("Smazat", $"Opravdu si
přejete opravu {RepairName}?", "Ano", "Ne");
if (!answer) return;


// HTTP požadavek na smazání opravy vozidla
HttpResponseMessage response = await api.client.
    DeleteAsync($"repair/delete?id={RepairId}");
```

Zdrojový kód 22: HTTP Delete požadavek
Zdroj: Autor

Nejprve se pomocí displayAlert ověří, zda uživatel skutečně chce smazat danou položku. Poté se vyhodnotí odpověď. Pokud je ano, následuje asynchronní HTTP DELETE požadavek na smazání položky. V metodě DeleteAsync je adresa s identifikátorem položky (id) za otazníkem pro smazání položky.

7.6 Otevírání a zavírání stránek

V .NET MAUI jsou jednotlivé části aplikace tvořeny do stránek, které jsou zobrazovány uživateli. Tyto stránky se skládají postupně na sebe. Uživatel má seznam otevřených stránek, ale vždy vidí jen tu aktuální.

```
public async void EditCar(int carId)
{
    Car car = cars.FirstOrDefault(c => c.id == carId);
    MobileCars mobileCars = this;
    await Navigation.PushAsync(new MobileNewCar(car, user.id, mobileCars));
}
```

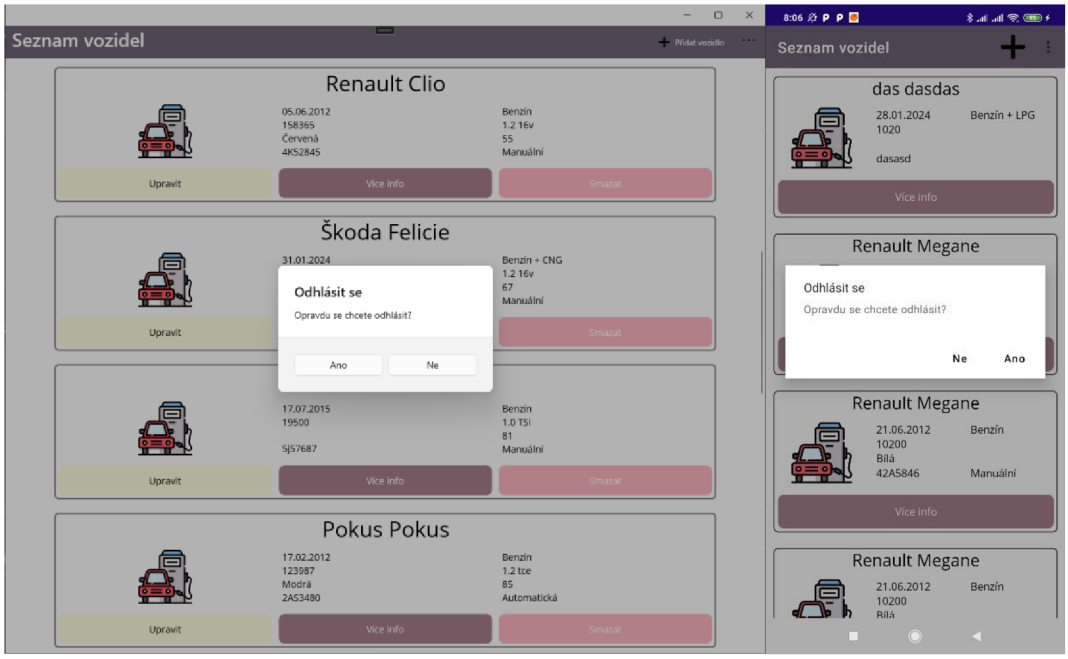
Zdrojový kód 23: Otevření nové stránky
Zdroj: Autor

Pro otevření nové stránky se volá interface **Navigation** a funkce **PushAsync** v jejichž závorkách je uvedený název stránky, která se má právě otevřít. V tomto případě se otevře formulář pro editaci informací o vozidle s parametry vozidlo, id uživatele a aktuální stránkou. V případě zavření stránky se opět využívá stejného rozhraní, ale volá se metoda **PopAsync** bez žádných parametrů.

7.7 Display alert

Display alert je dialogové vyskakovací okénko, které má za úkol informovat uživatele o stavu. Případně mu dá na výběr z možnosti ano/ne. Může v něm zobrazit pole pro doplnění jakéhokoli textu, ale nejčastěji se používá k informování uživatele. Zároveň je na něm krásně vidět vlastnost .NET MAUI, která využívá nativní prvky daného operačního systému. Pro Android je to takzvaný **AlertDialog**, a Windows

nativní komponenta. To znamená, že sice vzhled těchto oken bude mírně odlišný, ale funkcionalita úplně stejná.



Obr. 26: Dialogové okno Windows a Android
Zdroj: Autor

7.7.1 Asynchronní funkce

Asynchronní funkce se definují za pomoci dvou slovíček, a to **async** za modifikátory přístupu a před návratový typ metody v jazyce C#. Uvnitř této funkce se využívá klíčové slovo **await**, pro označení místa s čekáním na dokončení asynchronní operace. Její využití může být různé. Využívá se například u dialogových oken, kde se čeká na reakci uživatele, než se bude dál pokračovat. Dále velice často u HttpClienta při komunikaci s webovými službami, API a dalšími síťovými zdroji, kde se čeká na odpověď serveru. Dále se v .NET MAUI využívá při otvírání a zavírání stránek aplikace. To je vidět ve zdrojovém kódu č. 23, kde je uvedený zdrojový kód.

7.8 TabbedPage

Vytvořit stránku tabbedPage jde téměř pár kliknutími. V prostředí visual studia jsou pro tvorbu stránky jen ContentPage. To znamená, že se musí přepsat původní název ContentPage na druhém řádku v XAML a C# kód, aby se dalo pracovat s tabbedPage.

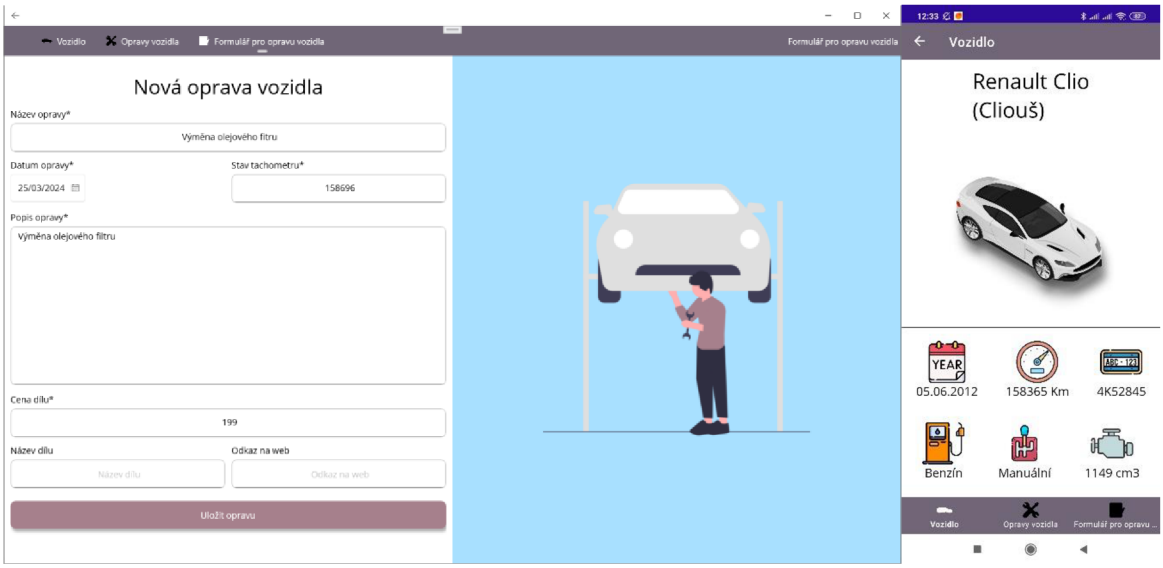
```xml
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="AutoServis.Views.All.Pages.CarDetail.AllCarDetailTabbedPage"
            xmlns:android="clr-namespace:Microsoft.Maui.Controls.PlatformConfiguration.AndroidSpecific;assembly=Microsoft.Maui.Controls"
            android:TabbedPage.ToolbarPlacement="Bottom"
            android:TabbedPage.IsSwipePagingEnabled="False"
            xmlns:controls="clr-namespace:AutoServis.Views.All.Pages.CarDetail"
            xmlns:control="clr-namespace:AutoServis.Components.Templates"
            BarBackgroundColor="{x:StaticResource MyPrimary}"
            BarTextColor="White"
            SelectedTabColor="White"
            UnselectedTabColor="Black"
            CurrentPageChanged="OnCurrentPageChange">

    <ContentPage
        Title="Vozidlo"
        IconImageSource="sport_car.png"
        x:Name="AboutCarPage">
        <ScrollView>
            <VerticalStackLayout x:Name="verticalViewCarInfo">
                <!-- Vložení šablony CarInfo -->
            </VerticalStackLayout>
        </ScrollView>
    </ContentPage>
    <ContentPage
        Title="Opravy vozidla"
        IconImageSource="car_repair.png">
        <ScrollView MaximumWidthRequest="1000">
            <VerticalStackLayout x:Name="verticalViewCarRepair">

            </VerticalStackLayout>
        </ScrollView>
    </ContentPage>
    <controls:AllCarDetailFormRepair IconImageSource="form.png"/>
</TabbedPage>
```

Obr. 27: Zdrojový kód stránky TabbedPage
Zdroj: Autor

Několik prvních řádků je stejných a příliš se od klasické stránky neliší. Řádek číslo pět přidává možnost pracovat s platformou Android. Následující řádek udává, že menu na přepínání mezi stránkami bude umístěno v dolní stráně obrazovky. Ve výchozím stavu je nahoře. Bohužel to samé nelze nastavit u operačního systému Windows, kde se menu bude vždy nacházet v horní části obrazovky. Řádek číslo sedm zakazuje přepínaní mezi stránkami tažením do strany. Důvod tohoto opatření je, že při kombinaci se **SwipeView**, docházelo k přepínání stránek na místo zobrazení možností. Další dva řádky kódu přidávají možnost přidat stránku a komponenty z Templates. Pod nimi se pouze nachází stylování a vzhled lišty TabbedPage. Na čtrnáctém řádku je umístěna událost CurrentPageChanged. O ní více v kapitole 7.8.2. Od řádku šestnáct se pak nachází jednotlivé stánky, které si uživatel může zobrazit.



Obr. 28: Ukázka TabbedPage desktopu a mobilní zařízení
Zdroj: Autor

7.8.1 TableView

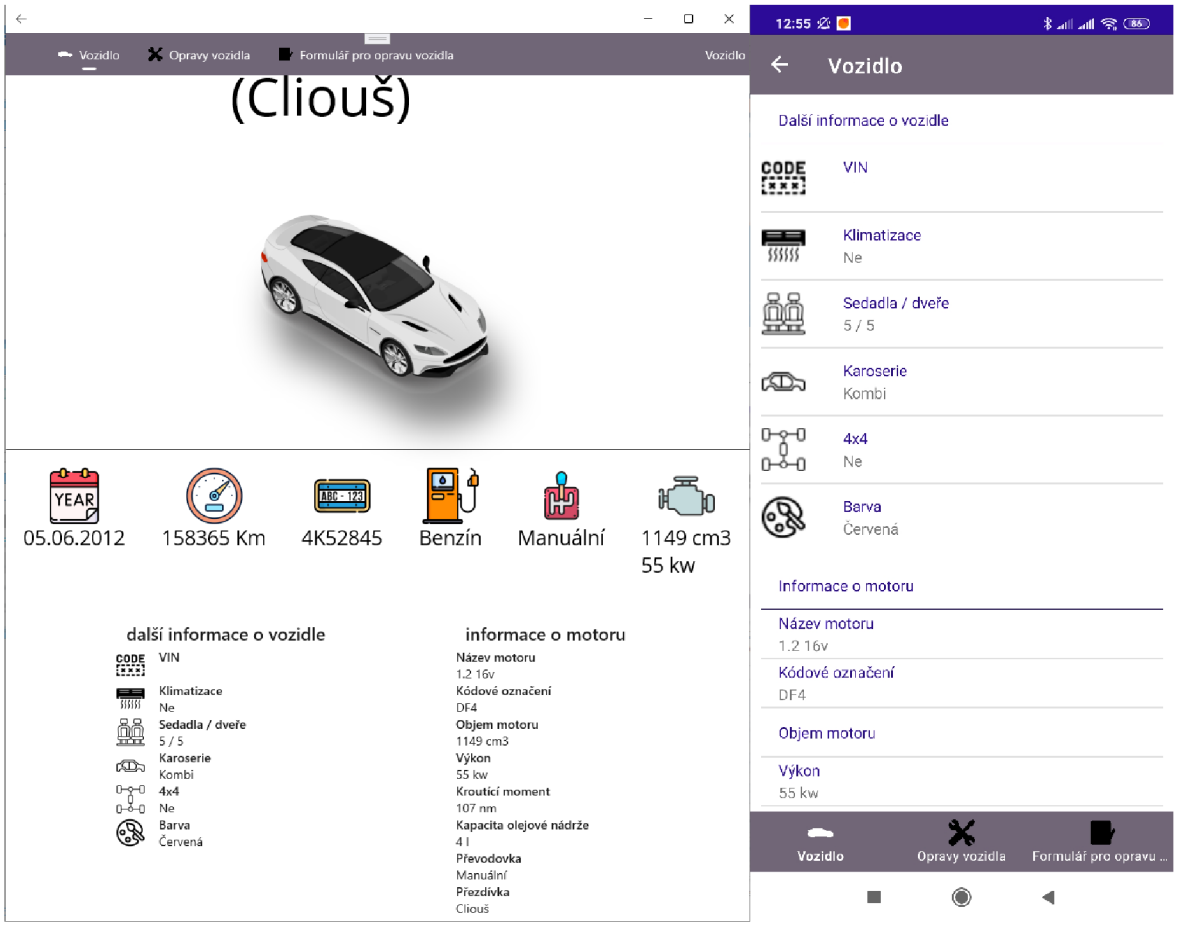
Komponenta tableView je různorodá. Ve své podstatě nenahrazuje datagrid, ale slouží jako zobrazovač dat. Má vlastnost **Intent**, která mění vzhled, ale pouze na zařízení s operačním systém iOS. Poté následuje TableRoot a TableSection, kde se definuje nadpis a v něm pak jednotlivé buňky.

```
<TableView Intent="Data" Margin="5">
    <TableRoot>
        <TableSection Title="Informace o motoru">
            <TextCell
                Text="Název motoru"
                Detail="{Binding CarEngine}"/>
            <TextCell
                Text="Kódové označení"
                Detail="{Binding CarCode}"/>
            <TextCell
                Text="Objem motoru"
                Detail="{Binding CarDisplacement2}"/>
            <TextCell
                Text="Výkon"
                Detail="{Binding CarPower}"/>
            <TextCell
                Text="Kroutící moment"
                Detail="{Binding CarTorque}"/>
            <TextCell
                Text="Kapacita olejové nádrže"
                Detail="{Binding CarOil}"/>
            <TextCell
                Text="Převodovka"
                Detail="{Binding CarGearbox}"/>
            <TextCell
                Text="Přezdívka"
                Detail="{Binding CarNickname}"/>
        </TableSection>
    </TableRoot>
</TableView>
```

```
<TableView Intent="Data" Margin="5">
    <TableRoot>
        <TableSection Title="Další informace o vozidle">
            <ImageCell
                Text="VIN"
                Detail="{Binding CarVIN}"
                ImageSource="vin_icon32.png"/>
            <ImageCell
                Text="Klimatizace"
                Detail="{Binding CarAirCondition}"
                ImageSource="aircondition_icon32.png"/>
            <ImageCell
                Text="Sedadla / dveře"
                Detail="{Binding CarSeatDoor}"
                ImageSource="seats_icon32.png"/>
            <ImageCell
                Text="Karoserie"
                Detail="{Binding CarBody}"
                ImageSource="body_icon32.png"/>
            <ImageCell
                Text="4x4"
                Detail="{Binding CarDrive}"
                ImageSource="allwheel_icon32.png"/>
            <ImageCell
                Text="Barva"
                Detail="{Binding CarColor}"
                ImageSource="color_icon32.png"/>
        </TableSection>
    </TableRoot>
</TableView>
```

Obr. 29: Zdrojový kód dvou TableView
Zdroj: Autor

Buňky jsou buď pevně definované se svými vlastnostmi, jako je **TextCell**. V tomto elementu se pod vlastností **Text** napíše nadpis buňky a v **Detail**u textový obsah buňky. Další možností je **ImageCell**, kde je před nadpis a detail umístěn obrázek. ImageCell přináší nepříjemné překvapení s manipulací obrázků, kdy se jím nedá nastavit velikost. Při vložení příliš velkého obrázku se buňka roztáhne podle jeho velikosti, ale to pouze na zařízení s operačním systémem Windows. Na Androidu se obrázek přizpůsobí. Z tohoto je vidět, že .NET MAUI je více koncipován na mobilní zařízení než desktopová. I ze samotného vzhledu TableView to takto působí.



Obr. 30: Desktopový a mobilní vzhled TableView
Zdroj: Autor

7.8.2 Událost CurrentPageChanged

Tato zmíněná událost se zavolá vždy při přepnutí mezi stránkami. Tato událost by nebyla nutná, ale TableView v kombinaci s TabbedPage funguje neúplně. Problém nastal při přepínání mezi stránkami. Po přepnutí zpět na úvodní stránku, kde jsou podrobnější informace o vozidle, se vymazaly data z TableView, ale pouze na Operačním systému Windows. Jednoduchým řešením je vždy znovu vykreslit tabulky na úvodní straně na platformě Windows.

```
private void OnCurrentPageChange(object sender, EventArgs e)
{
       #if WINDOWS
              if (this.CurrentPage == AboutCarPage)
              {
                     if (this.car != null) ShowCarInfo(this.car);
                     return;
              }
       #endif

       if (this.CurrentPage == RepairCarpage)
       {
        LoadRepairsFromDatabase();
       }
}
```

Zdrojový kód 24: Obsluha události pouze pro Windows
Zdroj: Autor

V kódu se nachází podmíněný blok s příkazem if. Tento blok se aktivuje pouze pokud bude aktuální platforma Windows. Uvnitř podmíněného bloku se nachází podmínka kontrolující stránku. Jestliže je aktuální stránka shodná s první, pokračuje se dál na kontrolu, zda existuje objekt Car. Pokud není null, zavolá se metoda ShowCarInfo s parametrem auta a znovu se vykreslí dané tabulky. Pod podmíněním blokem se nachází ještě jedna podmínka, která načítá všechny opravy vozidla z databáze, pokud je otevřená stránka RepairCarpage (Opravy vozidla).

7.9 Otevření internetového prohlížeče

Při zápisu údajů o opravě vozidla má uživatel možnost volitelně uvést název a URL adresu dílu. Pokud zadá URL adresu, bude mít možnost při prohlížení detailu o opravě na ni kliknout a přejít tak na webovou stránku s daným dílem.

```
private async void OpenLink(object sender, EventArgs e)
{
    try
    {
        if (url != "")
        {
            Uri uri = new Uri(url);
            await Browser.Default.OpenAsync(uri,
BrowserLaunchMode.SystemPreferred);
        }
        else throw new Exception();
    }
    catch (Exception ex)
    { // An unexpected error occurred. No browser may be installed on the device.
        await DisplayAlert("Oznámení", "Při otvírání prohlížeče došlo k chybě",
"ok");
    }
}
```

Zdrojový kód 25: Otevření webového prohlížeče
Zdroj: Autor

Po kliknutí na URL adresu se zavolá událost OpenLink. V ní se nachází blok try-catch pro zachycení a zpracování výjimek, které by mohli nastat. Pokračuje se na kontrolu proměnné url, jestli není prázdná. Když ne, vytvoří se instance třídy Uri s parametrem url. Následuje asynchronní funkce pro otevření defaultního prohlížeče. Pokud by byla proměnná url prázdná došlo by k vyhození výjimky, která se zachytí v bloku catch.

```
<queries>
	<intent>
		<action android:name="android.intent.action.VIEW" />
		<data android:scheme="http"/>
	</intent>
	<intent>
		<action android:name="android.intent.action.VIEW" />
		<data android:scheme="https"/>
	</intent>
</queries>
```

Zdrojový kód 26: Úprava AndroidManifest.xml
Zdroj: Autor

Do souboru **AndroidManifest.xml** je potřeba přidat následující queries/intent, pokud je cílová verze Androidu nastavená na Android 11 (R API 30) nebo vyšší. Je to z důvodu, že Android používá požadavky na viditelnost balíčku [28]. Pro Windows a iOS není potřeba nic dalšího přidávat. Stačí pouze implementovat zdrojový kód č. 25.

Obr. 31: Detail opravy a otevření dílu v prohlížeči pomocí URL adresy
Zdroj: Autor

7.10Čtení obrázků a barev z resourses

Přistup k těmto prvkům v projektu je poměrně přímočarý. Vše se nachází ve složce **Resources**. V této složce se nachází podsložky pro další práci. Mezi nimi fonty, obrázky, styly atd. Všechny tyto podsložky jsou pak dostupné v celém projektu.

Přístup ke složce Images s obrázky je jednoduchý. V XAML části se vezme komponenta pro obrázek, kterou je Image. Má vlastnost Source a do ní se napíše pouze název obrázku. Jedinou podmínkou je, že pojmenování obrázku musí být malými písmeny a odděleno pouze podtržítkem.

```
<Image Source="year_icon.png" Aspect="AspectFit" HeightRequest="64"/>
```

Zdrojový kód 27: Image
Zdroj: Autor

Pokud je potřeba využívat stejné barvy v celém projektu, je dobré je definovat do složky Style do souboru Colors.xaml. Je to slovník, kam se uloží základní definice a pak se k němu bude přistupovat pomocí klíčového slova uloženého v x:Key. Napíše se prvek Color a do něho barva v hexadecimální podobě.

```
<Color x:Key="Tertiary">#2B0B98</Color>
<Color x:Key="MyLightBlue">#A8E0FF</Color>
<Color x:Key="MyPrimary">#706677</Color>
<Color x:Key="MySecondPrimary">#A6808C</Color>
<Color x:Key="MySecond">#CCB7AE</Color>
```

Zdrojový kód 28: Slovník Color.xaml v složce Styles
Zdroj: Autor

Přístup k barvám už vyžaduje definici přístupové cesty. V základu umí .NET MAUI více druhů barev, a to nejen z resourses. Pokud je potřeba načíst barva z resourses musí se definovat složené závorky a v nich StaticResource a poté název barvy ze zmíněného x:Key.

```
<Button Text="Přihlásit se" Clicked="ClickLogin"
        BackgroundColor="{StaticResource MySecondPrimary}">
    <Button.Shadow>
        <Shadow Brush="Black" Offset="0,10" Radius="20" Opacity="0.4" />
    </Button.Shadow>
</Button>
```

Zdrojový kód 29: Ukázka načtení barvy z Resources
Zdroj: Autor

8 Výsledky a závěr

V této práci byl rozebrán a popsán framework .NET MAUI. Cílem bylo vyzkoušet, zda se tento nový framework dá využít pro tvorbu desktopových aplikací, protože se jedná o nástupce Xamarin.Forms, který umožňoval vytvářet aplikace pro Android a iOS.

Se samotným .NET MAUI se dobře pracuje, je přehledný, včetně jasné struktury projektu. Využití pro desktopové řešení určitě má, ale některé komponenty působí, že byly navrhovány více pro mobilní zařízení viz. TableView. Rovněž přepínání mezi stránkami, tento dojem potvrzují. Díky stránkám se aplikace dobře člení a rozděluje se obsah. Pokud se ale vývoj zaměří zároveň na desktop a mobilní zařízení, jako v této práci, působí to dojmem, že tvoříme pouze pro mobilní zařízení a desktopovou část necháváme na druhém místě.

Zároveň se v projektu povedlo implementovat zajímavý design, který určitě zaujme spoustu uživatelů. Společně s tímto se podařil i konzistentní design napříč zařízeními. To znamená, pokud uživatel otevře aplikaci na desktopovém zařízení a následně na mobilním, bude aplikace téměř stejná jen s drobnými rozdíly. Největším rozdílem je seznam vozidel, kde na mobilním zařízení nejsou zobrazena dvě tlačítka pro editaci a smazaní, protože jsou skrytá pod komponentou SwipeView. Takže uživatel musí potáhnout do boku pro odhalení těchto tlačítek.

.NET MAUI nabízí funkci poklepání, například dvojitého. Tato funkce by byla užitečná v některých částech mobilní části aplikace. Bohužel tato možnost je dostupná u obrázku, ale u jiných komponent ne, proto nebyla implementována.

Skvělou vlastností je také rozdělení obsahu na jednotlivé stránky. Díky tomuto rozdělení, není příliš mnoho informací v jedné části aplikace, a to zlepšuje uživatelskou přívětivost a přehlednost. Pro možnost mít více stránek vedle sebe, slouží rozložení stránky TabbedPage, které umožňuje rychlé přepínání mezi nimi. Přepínání mezi stránkami není tak rychlé a plynulé, z důvodu načítání dat z databáze. Zároveň se do aplikace nepodařilo zakombinovat rozložení FlyoutPage. Tato možnost mohla přinést další možnosti navigace pro uživatele.

Při práci s frameworkem .NET MAUI bylo zjištěno, že chybí komponenta Datagrid, která byla ve starém WinForms, sloužila pro práci s objemnějšími daty. Je zde alternativa v podobě TableView. Tato komponenta je hodně nedostačující z hlediska potřeby, práce s daty. Její výhodou je rozumný vzhled a určení spíše na mobilním zařízení, ale její použití na desktopovém zařízení je specifický.

Dále práce s některými komponentami, nebyly úplně dle očekávání. U některých z nich chyběli nějaké funkce, které by vývojář očekával.

Nejdůležitější věcí je, že aplikace funguje na desktopovém a mobilním zařízení a zobrazuje stejný obsah. A to bylo i primárním cílem při tvorbě aplikace. Aby změna provedená uživatelem v desktopové části byla propsána i do mobilní části a naopak. Díky tomu nezáleží na tom, z jakého zařízení se uživatel přihlásí a začne přidávat nebo upravovat data. Chybí zde úplná synchronizace. Pokud je uživatel přihlášen na obou zařízeních zároveň (desktopovém i mobilním), a provede změny na jednom z nich, nemohou se propsat ihned, ale až po aktualizaci pohledu nebo po odhlášení.

Díky jazyku XAML se dá vytvořit krásné a bohaté uživatelské rozhraní, které ocení každý uživatel. Je tu možnost oddělit mobilní a desktopovou část, a to různými stránkami nebo jen částečně v jazyce XAML pomocí **OnPlatform**. Dá se tvořit i jeden vzhled pro obě platformy, ale následně se obětovává desktopové uživatelské rozhraní. Zároveň je výhodou využití nativních komponent operačního systému.

Oproti jiným technologiím je .NET MAUI krok vpřed díky modernímu přístupu a nejnovějším technologiím. I když by si mohl něco přenést od svých předchůdců. Rozhodně by se mohla objevit komponenta datagrid, která umožňuje lepší práci s objemnými daty, která v aktuální verzi není.

.NET MAUI je skvělý framework, který má budoucnost a dají se v něm vytvářet zajímavé aplikace. Jeho vývoj by bylo lépe začít vytvářet na desktopovém zařízení a po dokončení se přesunout na mobilní, aby uživatelé na obou platformách měli vzhledově i funkčně stejnou aplikaci.

9 Seznam použité literatury

[1] KOĎOUSKOVÁ, Barbora. *Co je webová a desktopová aplikace a jaký je mezi nimi rozdíl?* [online]. 11.06.2023 [cit. 28.11.2023]. Dostupné z: https://www.rascasone.com/cs/blog/desktop-web-aplikace

[2] Microsoft, *Co je .NET? Úvod a přehled* [online]. 15.11.2023 [cit. 01.12.2023]. Dostupné z: https://learn.microsoft.com/cs-CZ/dotnet/core/introduction

[3] LIBERTY, Jesse. *Programming: Building .NET Applications with C#* [online]. O'Reilly Media, 22.02.2005 [cit. 02.12.2023]. Dostupné z: https://books.google.cz/books?id=Ns4e0ko-S7QC&dq=C%23&lr=&hl=cs&source=gbs_navlinks_s

[4] Microsoft, *Novinky v jazyce C# 12* [online]. 15.11.2023 [cit. 02.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/csharp/whats-new/csharp-12#ref-readonly-parameters

[5] Microsoft, *Začínáme s .NET Framework* [online]. 22.09.2022 [cit. 03.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/framework/get-started/

[6] HARTINGER, David. *Lekce 1 – Úvod do C# a .NET Frameworku* [online]. (neuvedeno) [cit. 13.04.2024]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

[7] THUAN, L. THAI a HOANG, Lam. *.NET Framework Essentials: introducing the .NET Framework* [online]. O'Reilly Media, 2003 [cit. 03.12.2023]. Dostupné z: https://books.google.cz/books?lr=&hl=cs&id=74mxhEyaKs8C&dq=.net+framework&q=

[8] GITTLEMAN, Arthur. *Computing With C# And The .NET Framework* [online]. Jones & Bartlett Learning, 2003 [cit.03.12.2023]. Dostupné z: https://books.google.cz/books?id=oyg8GaIhtMoC&dq=.net+framework&lr=&hl=cs&source=gbs_navlinks_s

[9] Microsoft, *Průvodce pro desktop (model Windows Forms s .NET)* [online]. 13.10.2023 [cit. 03.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/desktop/winforms/overview/?view=netdesktop-8.0

[10] SELLS, Chris a GRIFFITHS, Ian. *Programming WPF: Building Windows UI with Windows Presentation Foundation* [online]. O'Reilly Media, 2007 [cit. 09.12.2023]. Dostupné z: https://books.google.cz/books?id=558i6t1dKEAC&dq=WPF&lr=&hl=cs&source=gbs_navlinks_s

[11] Microsoft, *Přehled XAML (WPF .NET)* [online]. 13.10.2023 [cit. 09.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/desktop/wpf/xaml/?view=netdesktop-8.0

[12] Microsoft, *Přehled datových vazeb (WPF .NET)* [online]. 13.10.2023 [cit. 10.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/desktop/wpf/data/?view=netdesktop-8.0

[13] Microsoft, *What's a Universal Windows Platform (UWP) app?* [online]. 20.04.2023 [cit. 10.12.2023]. Dostupné z: https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

[14] NAGEL, Christian. *Professional C# 7 and .NET Core 2.0* [online]. John Wiley & Sons, 21.03.2018 [cit. 15.12.2023]. Dostupné z: https://books.google.cz/books?id=prdSDwAAQBAJ&dq=&lr=&hl=cs&source=gbs_navlinks_s

[15] MEHTA, Parag. *What is the difference Between .NET 7 and .NET 8?* [online]. 27.11.2023 [cit. 15.12.2023]. Dostupné z: https://positiwise.com/blog/difference-between-net-7-and-net-8

[16] Education-Wiki.com, *.Net Core vs .Net Framework – 8 hlavních rozdílů, které byste měli vědět* [online] (nedatováno) [cit. 16.12.2023]. Dostupné z: https://cs.education-wiki.com/2427394-.net-core-vs-.net-framework

[17] Microsoft, *Co je Xamarin?* [online]. 05.03.2024 [cit. 13.04.2024]. Dostupné z: https://learn.microsoft.com/cs-cz/xamarin/get-started/what-is-xamarin

[18] Microsoft, *Novinky v rozhraní .NET Standard* [online]. 10.05.2023 [cit. 16.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/standard/whats-new/whats-new-in-dotnet-standard?tabs=csharp

[19] ANS Plus, *Nativní vs multiplatformní vývoj aplikací* [online] (neuvedeno) [cit. 16.12.2023]. Dostupné z: https://www.asnplus.com/cs/blog/nativni-multiplatformni-vyvoj-aplikaci

[20] DUBOIS, Paul. *MySQL, Fifth Edition* [online]. Addison-Wesley, 28.03.2013 [cit. 17.12.2023]. Dostupné z: https://books.google.cz/books?id=JgFTUsIC0bUC&dq=&lr=&hl=cs&source=gbs_navlinks_s

[21] BIEHL, Matthias. *API* Architecture [online]. CreateSpace Independent Publishing Platform, 22.05.2015 [cit. 15.04.2024]. Dostupné z: https://www.google.cz/books/edition/API_Architecture/6D64DwAAQBAJ?hl=cs&gbpv=0

[22] Microsoft, *Co je .NET MAUI?* [online] 14.11.2023 [cit. 20.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/what-is-maui?view=net-maui-8.0

[23] Microsoft, *Ovládací prvky* [online] 13.02.2024 [cit. 20.02.2024]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/user-interface/controls/?view=net-maui-8.0

[24] Microsoft, *ContentPage* [online], 13.02.2024 [cit. 13.04.2024]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/user-interface/pages/contentpage?view=net-maui-8.0

[25] Microsoft, *Rozložení* [online], 13.02.2024 [cit. 20.02.2024]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/user-interface/layouts/?view=net-maui-8.0

[26] Microsoft, *Co je Visual Studio?* [online]. 24.10.2023 [cit. 20.12.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/visualstudio/get-started/visual-studio-ide?view=vs-2022

[27] SHARMA, K. Anoop. *Project Structure of .NET MAUI Application* [online]. 14.08.2023 [cit. 02.01.2024]. Dostupné z: https://www.c-sharpcorner.com/article/project-structure-of-net-maui-application/

[28] Microsoft, *Prohlížeč* [online], 13.02.2024 [cit. 13.04.2024]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/platform-integration/appmodel/open-browser?view=net-maui-8.0&tabs=android

10 Přílohy

Github repositář vzorové aplikace: https://github.com/MarekRuzic/AutoServis

Github repositář API pro komunikaci se vzorovou aplikací: https://github.com/MarekRuzic/API_carService

11 Zadání práce z IS (eVŠKP)

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Marek Růžička**
Studium:	I2100272
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Desktopové aplikace v .NET Core**
Název bakalářské práce AJ:	Desktop Applications in .NET Core

Cíl, metody, literatura, předpoklady:

Cíl: Prozkoumat možnosti tvorby multiplatformních desktopových aplikací v prostředí .NET Core a vytvořit vzorovou aplikaci s využitím získaných poznatků.

Osnova:

1. Úvod
2. Možnosti tvorby desktopových aplikací v .NET Framework vs .NET Core
3. Multiplatformní vs nativní vývoj
4. .NET MAUI
5. Vzorová aplikace
6. Závěr

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.
Datum zadání závěrečné práce:	26.1.2021