FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Tvorba rozšíření prototypu aplikace pro zpracování a analýzu telemetrických dat

Studijní program: B0613A140005 – Informační technologie
Studijní obor: B0613A140005AI – Aplikovaná informatika
Autor práce: **Fedor Elchaninov**
Vedoucí práce: Ing. Marián Lamr, Ph.D.
Konzultant: Ing. Přemysl Svoboda

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Tvorba rozšíření prototypu aplikace pro zpracování a analýzu telemetrických dat

Jméno a příjmení:	**Fedor Elchaninov**
Osobní číslo:	M20000011
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Seznamte se s problematikou tvorby mobilních aplikací, webových aplikací ASP.NET Core, a problematikou využití telemetrických dat pro analýzu jízdy na motocyklu.
2. Na základě získaných znalostí navrhněte další rozšíření existujícího prototypu aplikace pro zpracování a analýzu telemetrických dat.
3. Návrhy vytvořené na základě konzultací implementujte do mobilní a webové aplikace.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 až 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] FLANAGAN, David, 2020. The Definitive Guide: Master the World's Most-Used Programming Language. 7th ed. O'Reilly. ISBN 978-1491952023.
[2] FREEMAN, Adam, 2023. Pro ASP.NET Core 7. 10th ed. Manning. ISBN 978-1633437821.
[3] BEAULIEU, Alan, 2020. Learning SQL: Generate, Manipulate, and Retrieve Data. 3rd ed. O'Reilly. ISBN 978-1492057611.
[4] FRAIN, Ben, 2020. Responsive web design with HTML5 and CSS: develop future-proof responsive websites using the latest HTML5 and CSS techniques. 3rd ed. Birmingham, England: Packt Publishing. ISBN 978-1-83921-156-0.
[5] KREJCAR, Ondrej, 2011. Modern Telemetry. B.m.: InTech. ISBN 978-953-307-415-3.

Vedoucí práce:	Ing. Marián Lamr, Ph.D. Ústav mechatroniky a technické informatiky
Konzultant práce:	Ing. Přemysl Svoboda Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 12. října 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.
Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.
Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.
Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.
Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.
Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.
Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

13. 5. 2024 Fedor Elchaninov

Tvorba rozšíření prototypu aplikace pro zpracování a analýzu telemetrických dat

Abstrakt

Tato bakalářská práce je rozdělena na dvě hlavní témata. První z nich je implementace vybraných algoritmů pro zpracování telemetrických dat a vytvoření vizualizačního rozhraní pro analýzu výstupních dat. Druhé téma je věnováno tvorbě uživatelského rozhraní pro mobilní aplikaci. Vyvinutá řešení jsou navržena pro následnou integraci do stávajícího prototypu telemetrického systému.

Klíčová slova: .NET, ASP.NET Core, .NET MAUI, MVC, MVVM, Telemetrie

Development of an extension for a prototype application for telemetry data processing and analysis

Abstract

This bachelor thesis is divided into two main topics. The first is the implementation of selected algorithms for telemetry data processing and the creation of a visualization interface for output data analysis. The second topic is devoted to the development of a user interface for a mobile application. The developed solutions are designed for subsequent integration into an existing telemetry system prototype.

Keywords: .NET, ASP.NET Core, .NET MAUI, MVC, MVVM, Telemetry

Poděkování

Rád bych poděkoval svému vedoucímu práce, Ing. Mariánu Lamrovi, Ph.D a konzultantovi Ing. Přemyslu Svobodovi. Tato práce je výsledkem jejich přímého zapojení při vypracování. Dále bych rád poděkoval celé akademické obci TUL za možnost být její součástí. Speciální poděkování patří mé rodině a přátelům za jejich upřímnou podporu.

Obsah

Seznam zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CLR	Common Language Runtime
CSS	Cascading Style Sheets
DTO	Data Transfer Object
EF	Entity Framework
FCL	Framework Class Library
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	The HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identity
JSON	JavaScript Object Notation
JIT	Just-In-Time
LINQ	Language Integrated Query
MAUI	Multi-platform App User Interface
MVC	Model-View-Controller
MVVM	Model-View-Viewmodel
RTK	Real Time Kinematic
SQL	Structured Query Language
SVG	Scalable Vector Graphics
UI	User Interface
UWP	Universal Windows Platform
UX	User Experience
WPF	Windows Presentation Foundation
WWW	World Wide Web
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

V první části bakalářské práce se seznámíme s platformou .NET a využitím jejích komponent pro řešení problematiky vývoje mobilních a webových aplikací. V souvislosti s vývojem webových aplikací budeme podrobně analyzovat framework ASP.NET Core, pozornost budeme věnovat aplikačním a utilitárním frameworkům, nebudou přehlédnuty i návrhové vzory, jejichž praktickou implementací se budeme zabývat v následujících kapitolách. Poté se zaměříme na vývoj moderních mobilních aplikací na platformě .NET MAUI, její nové funkce a rozdíly oproti jejímu předchůdci - platformě Xamarin.

Druhá část bakalářské práce se zaměří na využití telemetrických dat k analýze jízdy na motocyklu. Zde si definujeme telemetrický systém, telemetrická data a popíšeme typ dat, se kterými budeme pracovat v následujících kapitolách. Ukážeme si stávající telemetrický systém a následně se zaměříme na definici kinematiky v reálném čase.

Třetí část bude věnována implementaci nových rozšíření pro webovou aplikaci. Zde představíme požadavky na funkčnost, uživatelské rozhraní, interakci mezi jednotlivými komponentami webové aplikace a podrobně popíšeme použité algoritmy a jejich implementaci. Dále se zaměříme na popis vytvoření webové aplikace pro vizualizaci telemetrických dat a výstupů implementovaných algoritmů s důrazem na použité návrhové vzory a techniky.

V poslední části bakalářské práce se budeme zabývat tvorbou uživatelského rozhraní mobilní aplikace podle požadovaného grafického návrhu. Nejprve se seznámíme s existující mobilní aplikací implementovanou na platformě Xamarin a poté se budeme zabývat konkretními požadavky na design a funkčnost. Následně se zaměříme na implementaci požadavků s využitím funkcí nové platformy .NET MAUI, kde se soustředíme na její jednotlivé součásti.

2 Vývoj webových a mobilních aplikací na platformě .NET

Vývoj mobilních a webových aplikací jsou dvě odlišné oblasti softwarového inženýrství, které se liší jak z hlediska technologií, tak i požadavků uživatelů a prostředí, ve kterém jsou používány. V této kapitole definujeme problematiku vývoje softwaru, ať už se jedná o mobilní nebo webovou aplikaci, a dále vysvětlíme, jak komponenty platformy .NET pomáhají tuto problematiku řešit.

Při vývoji softwaru jsou frameworky jedním ze základních nástrojů, které vývojáři používají. Framework neboli softwarový rámec je platforma, která poskytuje základ pro vývoj softwarových aplikací. Je možné si jej představit jako šablonu fungujícího programu, kterou lze selektivně upravovat přidáváním kódu. Využívá sdílené zdroje - například knihovny, obrazové soubory a referenční dokumenty - a spojuje je do jednoho balíčku. Tento balíček lze upravit podle konkrétních potřeb projektu. Pomocí frameworku může vývojář přidávat nebo nahrazovat funkce, aby aplikaci poskytl nové funkce [1]. Hlavním úkolem frameworku je vytvořit vývojáři takové prostředí, aby nemusel dělat nic jiného než psát funkční kód produktu. Seznam činností, které pomáhá řešit použití frameworku, lze definovat následovně:

- `Zrychlení procesu vývoje:` framework je opakovaně použitelná sada kódu, která má pomoci splnit určitý úkol. Obvykle urychluje vývoj tím, že poskytuje předpřipravená řešení běžných problémů, jako je přihlašování nebo připojení k rozhraní API, takže není nutné psát stejný kód znovu a znovu [2].
- `Konzistence:` mnoho frameworků používá standardizované struktury a konvence kódu, které musí vývojáři dodržovat. Tato standardizace umožňuje organizacím maximalizovat soudržnost a konzistenci napříč produkty a týmy.
- `Spolehlivost:` nejpopulárnější frameworky jsou dobře udržované, pravidelně aktualizované a opravované. Jsou také kompatibilní s různými prohlížeči a zařízeními, takže inženýři mohou vytvářet a dodávat produkty bez obav z chyb nebo problémů s kompatibilitou [3].
- `Komunita:` populární frameworky používá obrovské množství firem a vývojářů, čímž se tvoří komunita, jejíž vzájemná interakce pomáhá jednotlivým vývojářům přebírat cizí zkušenosti nebo řešit problémy.

Díky tomuto seznamu výhod se moderní vývoj softwaru neobejde bez použití frameworků. V následující sekci si představíme konkrétní framework, jeho strukturu a výhody jeho použití.

2.1 Platforma .NET

.NET je open source platforma pro vytváření desktopových, mobilních a webových aplikací, které lze spustit v jakémkoli operačním systému. Systém .NET zahrnuje nástroje, knihovny a jazyky, které podporují moderní, škálovatelný a vysoce výkonný vývoj softwaru. Platformu .NET podporuje a udržuje aktivní komunita vývojářů [4].

.NET je univerzální a umožňuje vývojářům vytvářet širokou škálu typů softwaru pro různá zařízení a platformy. Její komplexní sada frameworků, jazyků a nástrojů podporuje vývoj:

- `Webové aplikace`: pomocí ASP.NET mohou vývojáři vytvářet dynamické webové stránky, webové aplikace a webové služby. ASP.NET Core, modernější multiplatformní framework, umožňuje vývoj vysoce výkonných aplikací připojených k internetu v cloudu, například webových rozhraní API pro mobilní zařízení.
- `Desktopové aplikace`: klientské aplikace pro počítače se obvykle vyvíjejí pomocí technologií Windows Presentation Foundation (WPF) nebo Windows Forms [5]. WinForms nabízí přímočarý přístup k vytváření desktopových aplikací, zatímco WPF poskytuje robustnější framework pro vytváření bohatých uživatelských rozhraní pomocí XAML.
- `Mobilní aplikace`: s Xamarinem, který je součástí ekosystému .NET, mohou vývojáři vytvářet nativní mobilní aplikace pro Android, iOS a Windows pomocí jazyku C#. Xamarin.Forms umožňuje vývoj multiplatformních uživatelských rozhraní z jediné sdílené kódové základny. .NET MAUI (Multi-platform App UI) je dalším vývojem Xamarin.Forms, který rozšiřuje podporu i pro desktopové aplikace.
- `Mikroslužby a kontejnery`: platforma .NET je vhodná pro vytváření mikroslužeb - malých, modulárních a nezávisle nasaditelných služeb. Díky lehké a modulární architektuře je ASP.NET Core ideální pro kontejnerová prostředí a přímo podporuje Docker kontejnery.
- `Knihovny a frameworky`: vývojáři mohou vytvářet opakovaně použitelné knihovny a rámce, které poskytují funkce jiným aplikacím. Ty lze distribuovat prostřednictvím NuGet, správce balíčků .NET, a zpřístupnit je tak celosvětové komunitě .NET.

Kromě výše uvedených typů softwaru umožňuje platforma .NET vytvářet cloudové, herní a IoT aplikace, jakož i aplikace založené na strojovém učení a umělé

inteligenci pomocí frameworku ML.NET. Hlavním zdrojem dokumentace k platformě .NET jsou stránky MSDN (Microsoft Developer Network), na které se odkazuje v sekci použité literatury.

2.1.1 Framework Class Library

Knihovna Framework Class Library (FCL) je ucelená kolekce opakovaně použitelných typů, včetně tříd, rozhraní a datových typů, která je součástí prostředí .NET Framework a umožňuje přístup k funkcím systému [6].

- `Vstup/výstup souborů (I/O)`: třídy pro čtení ze souborů a datových proudů a zápis do nich.
- `Interakce s databázemi`: nástroje pro připojení k databázím, provádění příkazů a správu dat.
- `Vývoj webových aplikací`: komponenty pro vytváření dynamických webových stránek, služeb a rozhraní API.
- `Datové struktury`: implementace běžných datových struktur, jako jsou seznamy, fronty, pole a slovníky.
- `Síťování`: knihovny pro síťové operace, včetně požadavků HTTP, programování soketů a dalších.
- `Manipulace s dokumenty XML`: nástroje pro analýzu, validaci a manipulaci s dokumenty XML.
- `Zabezpečení`: funkce pro ověřování, šifrování a správu oprávnění.
- `Uživatelské rozhraní`: typy pro vytváření grafických uživatelských rozhraní (GUI) pro desktopové aplikace.

Poskytnutím standardizované sady tříd a typů FCL výrazně snižuje množství kódu, který musí vývojáři psát, čímž podporuje rychlý vývoj aplikací a zajišťuje konzistenci všech aplikací .NET.

2.1.2 Common Language Runtime

Common Language Runtime (CLR) je nástroj, který řídí spouštění programů napsaných v některém z několika podporovaných jazyků a umožňuje jim sdílet společné objektově orientované třídy napsané v některém z těchto jazyků. Je součástí prostředí .NET Framework společnosti Microsoft [7]. Poskytuje řadu důležitých služeb, které tvoří páteř modelu spouštění frameworku .NET, včetně:

- `Správa paměti`: automatická správa životnosti objektů prostřednictvím garbage collection, která pomáhá předcházet únikům paměti a dalším problémům souvisejícím s pamětí.

- `Typová bezpečnost`: zajištění přísné typové kontroly za běhu, která zajišťuje, že všechny operace s objekty jsou typově bezpečné.
- `Zpracování výjimek`: strukturovaný přístup k detekci a obnově chyb, který umožňuje vytvářet robustní a odolné aplikace.
- `Správa vláken`: podpora vícevláknových aplikací, která umožňuje aplikacím provádět více operací současně, a to způsobem, který maximalizuje výkon a rychlost odezvy.
- `Interoperabilita`: schopnost spolupracovat s kódem napsaným v jiných jazycích, což umožňuje integraci široké škály knihoven a volání API do aplikací .NET.

Kromě toho CLR za běhu kompiluje kód intermediate language (IL), nízkoúrovňovou instrukční sadu nezávislou na platformě, do nativního strojového kódu pomocí překladače Just-In-Time (JIT). Tento proces zajišťuje, že aplikace .NET mohou být spuštěny na jakékoli podporované platformě bez úprav a zároveň dosahují vysokého výkonu díky optimalizaci pro konkrétní prostředí.

2.2 ASP.NET Core

Podíváme se nyní na důležitou součást platformy .NET - ASP.NET Core a na to, jak její funkcionalita pomáhá řešit problematiku tvorby webových aplikací.

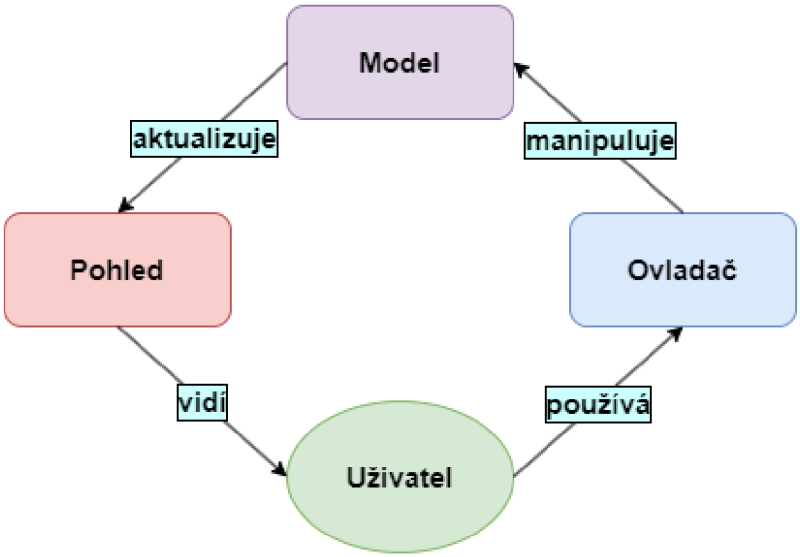
ASP.NET Core je multiplatformní framework pro vytváření webových aplikací a služeb. Skládá se z platformy pro zpracování požadavků HTTP, frameworků pro vytváření aplikací a utilitárních frameworků s pomocnou funkcionalitou [8]. Je nástupcem proprietárního frameworku ASP.NET a na rozdíl od něj je modulární (tj. je určitou abstrakcí nad jeho jednotlivými komponentami) a je šířen pod licencí MIT. Modulární framework řeší běžnou problematiku vývoje, kterou lze rozdělit do několika hlavních částí:

- `Kompatibilita a verze:` umožňuje snadnou aktualizaci na nové verze frameworku díky správě balíčků pomocí NuGet. Podporuje side-by-side verzování, takže různé aplikace vyvíjené na jednom počítači mohou být zaměřeny na různé verze ASP.NET Core.
- `Škálovatelnost:` ASP.NET Core přichází s vysoce modulární strukturou, což umožňuje vývojářům využívat pouze ty komponenty a závislosti, které jsou pro jejich aplikaci nezbytné. To vede k lepší správě závislostí a jednoduššímu aktualizování.
- `Vkládání závislostí:` framework má integrovanou podporu pro vkládání závislostí, což je důležitý prvek pro snadnější testování a udržování aplikací.

Pojmy škálování a vnoření závislostí úzce souvisejí s návrhovým vzorem MVC, který je součástí rozhraní .NET.

2.2.1 MVC

Jako hlavní schéma rozdělení dat aplikace byl zvolen vzor MVC (Model-View-Controller), protože takto navržený modul lze snadno začlenit do stávající struktury aplikace. MVC odděluje data aplikace, uživatelské rozhraní a řídicí logiku do tří samostatných komponent, což umožňuje jasné oddělení jednotlivých aspektů. [9]. Diagram MVC je zobrazen na obrázku 2.1.



Obrázek 2.1: Diagram MVC

Ovladač

Ovladač funguje jako prostředník mezi modelem a zobrazením. Přijímá vstupy od uživatele, volá objekty modelu za účelem získání dat nebo provedení operací a poté předává data zobrazení k prezentaci. Ovladač tedy řídí tok dat mezi modelem a zobrazením, a také kontroluje interakci uživatele se zobrazením [10].

Zobrazení

Zobrazení se používá pro veškerou logiku uživatelského rozhraní aplikace. Zobrazení jsou komponenty, které zobrazují data aplikace uživateli. V prostředí .NET MVC se zobrazení obvykle vytvářejí pomocí syntaxe zobrazovací jednotky Razor, která vývojářům umožňuje vkládat kód C# přímo do jazyka HTML. Rozhraní umožňuje také tradičnější způsob vytváření UI pomocí JavaScript kódu. Zobrazení načítá data z modelu a vykresluje uživatelské rozhraní, které tato data zobrazuje.

Model

Model představuje dynamickou strukturu dat aplikace nezávislou na uživatelském rozhraní. Přímo spravuje data, logiku a pravidla aplikace. V aplikaci .NET MVC jsou

modely třídy jazyka C#, které slouží k ukládání dat aplikace a manipulaci s nimi. Mohou obsahovat pravidla pro ověřování dat, která vynucují obchodní logiku.

2.2.2 Entity Framework Core

Entity Framework Core je rozšiřitelná a multiplatformní verze Entity Frameworku, objektově-relačního mapovacího rámce (ORM) společnosti Microsoft pro ADO.NET. Slouží jako základní nástroj pro přístup k datům v aplikacích .NET, zejména v rámci ASP.NET Core, a nabízí efektivnější způsob interakce s databázemi pomocí doménově specifických objektů, čímž minimalizuje množství kódu specifického pro databázi, který musí vývojáři psát.

Entity Framework usnadňuje vytváření datového modelu mapováním databázových tabulek na třídy .NET. Vývojáři mohou k interakci s databází používat LINQ, který EF převádí na dotazy SQL. Tato abstrakce umožňuje zaměřit se více na obchodní logiku aplikace, než na specifické vlastnosti databáze [11].

2.3 .NET a mobilní platformy

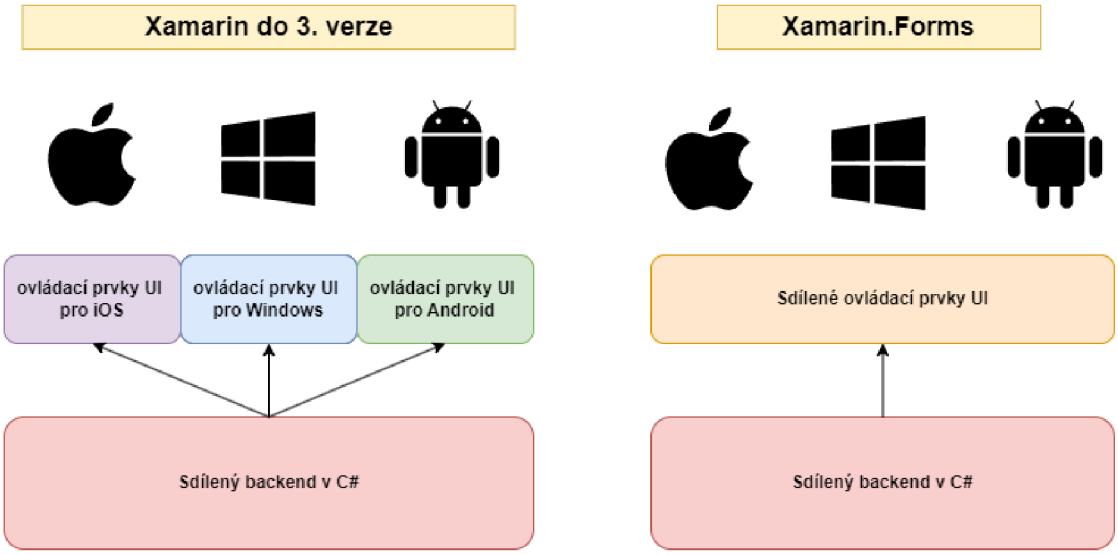
Nyní je jasné, jaké nástroje platforma .NET nabízí pro vývoj webových aplikací. Dalším krokem bude představení řešení pro tvorbu multiplatformních aplikací, včetně aplikací pro mobilní platformy Android a iOS.

Než se podíváme na moderní řešení nabízené platformou .NET, MAUI, stojí za to seznámit se s jeho předchůdcem, Xamarinem.

2.3.1 Xamarin

Xamarin, představený v roce 2011, se stal klíčovým frameworkem, který vývojářům umožňuje vytvářet multiplatformní mobilní aplikace pomocí jazyka C# a frameworku .NET. Poskytl jedinečnou nabídku, která umožnila sdílení kódu napříč platformami, čímž výrazně zkrátila dobu vývoje a snížila zdroje potřebné k nasazení aplikací v systémech iOS, Android a Windows. Akvizice Xamarinu společností Microsoft v roce 2016 znamenala významný milník, který začlenil Xamarin do ekosystému .NET a posílil závazek společnosti Microsoft k vývoji napříč platformami [12].

Jádrem architektury Xamarinu je framework Mono, open-source implementace frameworku .NET, který umožňuje provoz na různých platformách. Aplikace Xamarin se kompilují do nativního kódu, což jim umožňuje dosahovat výkonnostních charakteristik srovnatelných s aplikacemi napsanými v jazycích nativní platformy [13]. Tradiční přístup, představený v roce 2011, předpokládá jeden společný základ kódu pro různé platformy, ale různé implementace uživatelského rozhraní. Představení Xamarin.Forms tento přístup změnilo, a umožňuje používat přenosné podskupiny ovládacích prvků, které jsou namapovány na nativní ovládací prvky systémů Android, iOS nebo Windows, čímž odpadá nutnost vyvíjet samostatná řešení pro jednotlivé platformy. Porovnání těchto dvou přístupu je na obrázku 2.2.



Obrázek 2.2: Architektura Xamarinu

Inovativní přístup Xamarinu k multiplatformnímu vývoji se soustředí na použití jazyků C# a XAML, což vývojářům umožňuje využívat jejich znalosti z prostředí .NET při vytváření mobilních aplikací. Jazyk C# nabízí typově bezpečný, objektově orientovaný jazyk s robustními funkcionalitou, jako je např. LINQ, což usnadňuje sofistikovanou aplikační logiku a manipulaci s daty napříč platformami. XAML je využit pro definování uživatelských rozhraní deklarativním způsobem, což podporuje jasné oddělení problematiky mezi uživatelským rozhraním a obchodní logikou [14].

2.3.2 XAML

XAML je deklarativní značkovací jazyk. Při použití v programovacím modelu .NET zjednodušuje XAML vytváření uživatelského rozhraní aplikace .NET [15]. Použití jazyka XAML v kontextu vývoje multiplatformních aplikací je dáno několika vlastnostmi. Poskytuje především sjednocenou definici uživatelského rozhraní napříč platformami. To znamená, že vývojář má možnost definovat uživatelské rozhraní pro všechny cílové systémy bez zaměření na konkrétní implementace. Umožňuje centralizaci stylů a prostředků, což usnadňuje konzistentní vzhled celé aplikace a zároveň minimalizuje duplicitu kódu. A protože syntaxe XAML je založena na XML, umožňuje definovat prvky uživatelského rozhraní jako hierarchické struktury. Prvky XAML se mapují přímo na instance objektů CLR, zatímco atributy XAML se mapují na vlastnosti a události CLR těchto objektů.

2.3.3 .NET MAUI

.NET MAUI je moderní multiplatformní rozhraní, které Microsoft představila jako evoluci a nástupce Xamarin.Forms. Bylo vyvinuto v reakci na potřebu univerzálnějšího a aktualizovaného frameworku, který by byl schopen řešit problémy současného vývoje multiplatformních aplikací, poté, co Xamarin oznámil konec své životnosti.

Přestože je MAUI nástupcem Xamarin.Forms a sdílí s ním určitou funkcionalitu, existuje řada zásadních rozdílů:

- `Struktura projektu:` v .NET MAUI soubory a kód závislé na platformě jsou udržovány ve složkách platformy a pod cílovými názvy souborů platformy, zatímco v Xamarinu jsou udržovány v rámci různých projektů.
- `Sestavení a spuštění:` na rozdíl od Xamarinu, který se při sestavení aplikací spoléhá na proprietární .NET Framework, podporuje MAUI rozhraní CLI, které je součástí open-source platformy .NET.
- `Správa prostředků:` v Xamarinu je třeba udržovat soubory zdrojů pro každou platformu zvlášť, v MAUI veškeré zdroje lze udržovat na jednom místě. Díky podpoře formátu SVG navíc není nutné ukládat obrázky s rozlišením pro zařízení specifická pro danou platformu.
- `Podporované platformy a verze:` hlavní rozdíl v podpoře platforem mezi Xamarinem a .NET MAUI je v podpoře systému Windows. Xamarin podporuje UWP, zatímco .NET MAUI podporuje WinUI.
- `.NET 6:` jak bylo uvedeno výše, díky integraci MAUI do platformy .NET mají vývojáři k dispozici novou funkcionalitu z C#10 a .NET 6, která dříve v Xamarinu nebyla k dispozici [16].
- `Grafická API:` v Xamarinu není žádné přímé rozhraní API, které by umožňovalo kreslení. Ale multiplatformní grafická funkcionalita v .NET MAUI poskytuje kreslicí plátno pro kreslení a malování tvarů pomocí rozhraní GraphicsView.

Hlavním bodem je jednoznačně integrace MAUI přímo do platformy .NET, kromě výše uvedených rozdílů je zodpovědná za plnou podporu Hot Reload - funkce vývojového prostředí Visual Studio, která umožňuje aplikovat změny v kódu bez restartování aplikace [17].

2.3.4 MVVM

Rostoucí složitost vnitřní logiky mobilních aplikací způsobuje stejné vyzvy jako v případě vývoje webových aplikací. Monolitická aplikace se složitě testuje a často nepředpokládá multifunkčnost určitého kódu díky jeho opakovanému použití. pro řešení teto problematiky v kontextu vývoje mobilních aplikací na platformě .NET existují návrhové vzory, které definují strukturu aplikace. Jedním z nejpopulárnějších je architektonický vzor MVVM(Model-View-Viewmodel). MVVM je varianta vzoru MVC, která je přizpůsobena moderním platformám pro vývoj uživatelského rozhraní, kde za zobrazení odpovídá spíše návrhář než klasický vývojář [18]. Vzor rozděluje konstrukci aplikace na tři jednotlivé komponenty:

Model

Model je zodpovědný za správu dat aplikace. Je to zapouzdření dat a obchodních pravidel, která řídí přístup k těmto datům a jejich aktualizaci. Model informuje viewmodel o všech změnách v datech, který následně aktualizuje zobrazení [19].

View

Účelem zobrazení je informovat viewmodel o akci uživatele. Tato vrstva pozoruje viewmodel a neobsahuje žádnou aplikační logiku.

Viewmodel

Viewmodel je spojovacím bodem mezi zobrazením a modelem. Implementuje a vystavuje veřejné vlastnosti a příkazy, které zobrazení používá prostřednictvím datové vazby. Pokud dojde ke změně stavu, viewmodel informuje zobrazení prostřednictvím oznamovacích událostí [20]. Schéma MVVM vzoru je znázorněno na obrázku **2.3**.

Obrázek 2.3: Diagram MVVM

Důležitou roli v komunikaci mezi zobrazením a viewmodel ve vzoru MVVM hrají datové vazby. Datová vazba v MVVM funguje pomocí vazebního mechanizmu nebo frameworku, který sleduje vlastnosti modelu zobrazení a podle toho aktualizuje zobrazení. Vazbový mechanizmus může také zpracovávat uživatelské vstupy a události a vyvolávat příkazy pro viewmodel. Použití vazebního mechanizmu k synchronizaci zobrazení s viewmodel umožňuje snadnější testování a údržbu kódu. Navíc umožňuje sdílení viewmodel mezi různými zobrazeními nebo platformami [21].

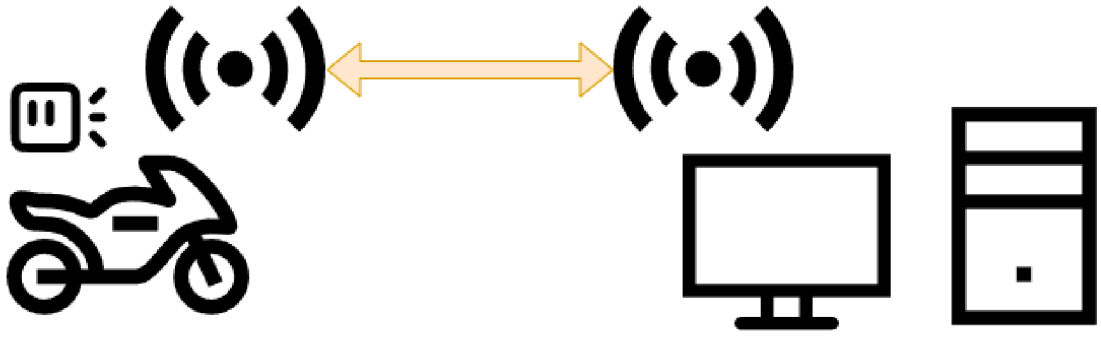
MVVM poskytuje jasný a strukturovaný přístup ke správě architektury aplikace. Trend vývoje napříč platformami a potřeba robustnějších a škálovatelnějších aplikací navíc jen podpoří zavádění architektury MVVM [22].

3 Využití telemetrie pro analýzu jízdy na motocyklu

V dnešním světě, kde se technologie a data stávají stále více propojenými s každodenním životem, hrají zpracování a následná analýza dat klíčovou roli v mnoha odvětvích. Jedním z takových příkladů je analýza jízd na motocyklu, kde se telemetrická data využívají k zlepšení bezpečnosti a výkonnosti. Tato kapitola se zaměří na popis použití telemetrických dat v kontextu webové aplikace navržené specificky pro analýzu motocyklových jízd.

3.1 Telemetrická data

Než začneme popisovat sbíraná data a jejích použití, je vhodné definovat pojmy telemetrie, telemetrický systém a telemetrická data. Telemetrie je technologie, která umožňuje dálkové měření, řízení a hlášení informací [23]. Telemetrický systém je systém, který automaticky zaznamenává a přenáší data ze vzdálených zdrojů do informačního systému na jiném místě za účelem monitorování a analýzy. Telemetrická data mohou být přenášena pomocí rádia, infračerveného záření, ultrazvuku, GSM, satelitu nebo kabelu, v závislosti na aplikaci [24]. Obecné schéma komunikace v telemetrickém systému je na obrázku 3.1.



Obrázek 3.1: Komunikace v telemetrickém systému

Telemetrická data v rámci vyvíjené aplikace se vztahují k souboru informací získaných z různých senzorů umístěných na motocyklu, které monitorují a zaznamenávají různé aspekty jízdy v reálném čase. Senzory jsou součástí zařízení, které

sbírá data a odesílá je na server pro následnou analýzu. Pro účely zapínání, vypínání a sledování stavu tohoto zařízení využívá tento telemetrický systém mobilní aplikaci, jejíž vývoj bude podrobně popsán v 5. kapitole.

3.2 Formát dat a jejich význam

Nasbíraná data jsou následně uložena na Microsoft SQL Serveru v tabulce. Tabulka je databázový objekt, který obsahuje všechna data v databázi. V tabulce jsou data logicky uspořádána ve formátu řádků a sloupců. Každý řádek představuje jedinečný záznam a každý sloupec představuje pole v záznamu [25].

Záznamy jsou pořizovány s frekvencí 25 snímků za sekundu. Jeden záznam v tabulce obsahuje 95 sloupců s údaji popisujícími jízdu v daném čase. Tyto sloupce lze rozdělit do tří samostatných skupin:

- `Identifikační:` slouží k jednoznačné identifikaci záznamu. Jedná se o sloupce s ID záznamu v tabulce, a ID cesty.
- `Informační:` jsou to sloupce, jejichž hodnoty popisují metadata cesty, například: popis místa, světelné a povětrnostní podmínky, e-mail uživatele a název snímacího zařízení.
- `Telemetrické:` nejdůležitější skupina dat z hlediska funkčnosti, která popisuje telemetrické údaje v určitém časovém okamžiku. Je vhodné se jí věnovat podrobněji.

Základ telemetrických dat tvoří tyto údaje:

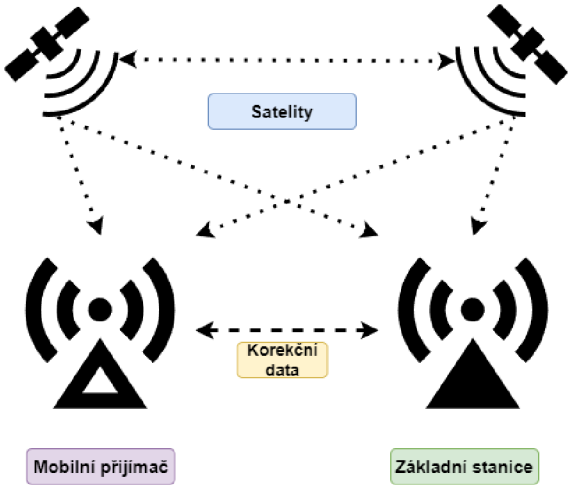
- `Čas:` čas snímání vyjádřený časovým razítkem.
- `Lat, Lon:` souřadnice GPS vyjádřené v hodnotách zeměpisné šířky a délky.
- `Ax, Ay, Az:` představují zrychlení ve směru x, y a z. Zrychlení je míra změny rychlosti v čase.
- `Vx, Vy, Vz:` označují rychlost ve směru x, y a z. Rychlost(anglicky velocity) je vektorová veličina, která označuje rychlost změny polohy objektu vzhledem k času. Zahrnuje nejen rychlost (skalární veličina), ale také směr pohybu.
- `Gx, Gy, Gz:` jedná se o složky gravitační síly (nebo gravitačního zrychlení) působící na zařízení nebo objekt ve směru x, y a z.

3.3 Real Time Kinematic

Real Time Kinematic GPS je typ technologie GPS, která využívá kombinaci signálů GPS a místní základnové stanice k poskytování vysoce přesných údajů o poloze. Na rozdíl od tradičních systémů GPS, které se spoléhají pouze na data ze satelitů, systémy RTK GPS využívají další data z blízké základnové stanice ke zlepšení přesnosti dat GPS. Díky tomu lze získat polohová data s přesností na centimetry, což je ideální pro širokou škálu aplikací [26].

3.3.1 RTK systém

Síťová RTK je založena na použití několika široce rozmístěných permanentních stanic. V závislosti na implementaci jsou údaje o poloze z permanentních stanic pravidelně předávány do centrální zpracovatelské stanice. Uživatelské terminály RTK vysílají svou přibližnou polohu do centrální stanice, která na vyžádání vypočítá a předá opravné informace nebo opravenou polohu uživatelskému terminálu RTK. Výhodou tohoto přístupu je celkové snížení počtu potřebných základnových stanic RTK. V závislosti na implementaci mohou být data přenášena prostřednictvím mobilních rádiových spojů nebo jiných bezdrátových médií [27]. Přehled RTK je zobrazen na obrázku 3.2.



Obrázek 3.2: Koncepční model RTK

V prototypu aplikace, jejíž rozšíření je cílem této práce, je RTK nezbytný. Analýza jízdy a její porovnání s jinými vyžaduje vysokou úroveň přesnosti, které lze dosáhnout pomocí RTK. Připojení k systému RTK se provádí pomocí mobilní aplikace. O mobilní aplikaci si povíme více v 5. kapitole.

4 Webová aplikace

Tato kapitola se bude věnovat popisu implementace vyvinutých rozšíření pro webovou aplikaci. Zde bude vysvětlena struktura programu, a jednotlivé návrhové modely a techniky. Nejprve si definujme požadavky a dále se zaměříme na popis imlementace jednotlivých algoritmů a modulu pro jejich vizualizaci.

4.1 Stanovení požadavků

Vývoj webových aplikací začíná definováním požadavků na funkčnost a omezení pro implementaci. Většina požadavků na rozšiřitelné moduly byla získána v počáteční fázi vývoje, v průběhu vývoje byly upřesňovány prostřednictvím konzultací.

4.1.1 Implementace

Jelikož se tato práce zabývá vývojem rozšiřitelných modulů pro existující aplikaci, jsou možnosti implementace omezeny technologiemi použitými při vývoji původní aplikace.

Serverová strana

Serverová část zdrojové aplikace je napsána pomocí frameworku ASP.NET Core s využitím výše popsaného architektonického vzoru MVC. Proto by měly být vyvíjené moduly psány v souladu s tímto přístupem. Pro komunikaci s databází se používají komponenty Entity Framework, které zajišťují konzistentní přístup k práci s daty. Řízení přístupu k aplikaci a konfigurace aplikace nepatří mezi úlohy, protože moduly nepředstavují nezávislé webové stránky. Budou implementovány do stávajícího prototypu aplikace a nasazeny již jako doplňková funkcionalita.

Klientská strana

Požadavky na klientskou část se týkají především uživatelského rozhraní, takže výběr technologií byl neomezený. Aby bylo možné aplikaci snáze integrovat, nebyly použity frameworky jako Angular nebo React. Místo toho byl zvolen standardní přístup využívající jazyk HTML pro definici obsahu stránky, CSS pro specifikaci stylů a JavaScript pro komunikaci se stranou serveru pomocí HTTP požadavků a vývoj logiky na klientské straně.

4.1.2 Funkcionalita

Nyní je jasné, jaká omezení se na implementaci vztahují, zbývá definovat požadavky na funkčnost vyvíjených modulů. V rámci rozšíření webové aplikace pro analýzu a vizualizaci dat bylo třeba vyvinout následující moduly:

- `Algoritmický modul:` představuje serverovou stranu s algoritmy pro analýzu jízdy na základě telemetrických dat, jako je výpočet úrovně zakřivení zatáček a rozdělení jízdy na autodromu na jednotlivá kola.
- `Modul pro vizualizaci telemetrických dat:` webová stránka, jejímž úkolem je vizualizace dat zpracovaných výše uvedeným modulem. K vizualizaci slouží mapa i grafy jednotlivých telemetrických hodnot. Pro analýzu jednotlivých úseků jízdy je možnost vizualizace dat v konkrétních časových intervalech.

Výsledkem je jednostránková webová aplikace, jejíž serverová část se skládá především z komponent prvního modulu a logika klientské části z komponent druhého modulu. Následující požadavky na funkčnost aplikace se týkají uživatelského rozhraní a z nich vyplývajících požadavků na vizualizační komponentu.

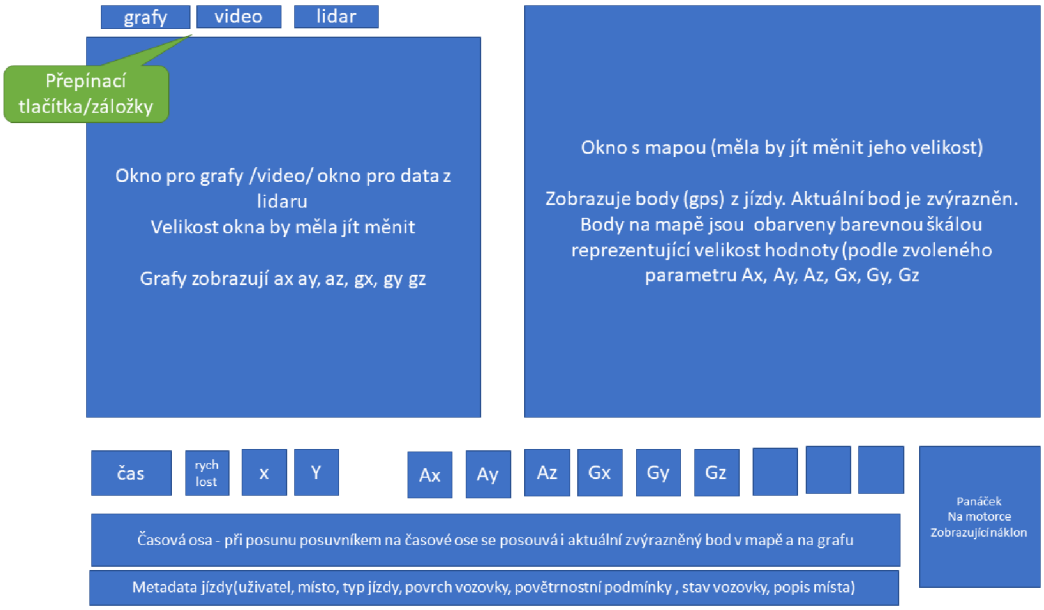
4.1.3 Uživatelské rozhraní

Uživatelské rozhraní slouží k analýze telemetrických dat přijatých ze senzorů senzorového zařízení a zpracovaných vybranými algoritmy. Aby je uživatel mohl analyzovat, je třeba je znázornit v grafech, které ukazují, jak se mění určité telemetrické hodnoty, a také na mapě, kde hlavní roli hrají spojené souřadnicové body podél trasy. Posuvník slouží k zobrazení hodnot ukazatelů a bodu na mapě ve zvoleném časovém okamžiku. Vizualizace všech původních požadavků na uživatelské rozhraní je znázorněna na obrázku 4.1.

Některé funkce uživatelského rozhraní, např. záložky s videem z jízdy a lidarem, byly během konzultací vynechané, protože požadovaná data nebyla v databázi k dispozici.

4.2 Implementace algoritmů

V této podkapitole se budeme zabývat implementací přístupu pro výpočet hodnoty zakřivení silnice a popíšeme vztah mezi zvoleným algoritmem a matematickým vzorcem pro výpočet křivosti křivky. Pak budeme pokračovat s popisem Haversinova vzorce a jeho použití při implementaci popisovaných algoritmů.



Obrázek 4.1: Vizualizace požadavku na UI

4.2.1 Výpočet zakřivení křivky

Popis algoritmu

Zakřivení křivky v daném bodě udává, jak prudce se křivka v daném bodě zakřivuje. Matematický vzorec pro křivost zahrnuje převrácenou hodnotu poloměru kružnice, která nejlépe aproximuje křivku v daném bodě. Vyjadřuje se jako $\kappa = 1/R$, kde κ je křivost a R je poloměr křivosti. Pro funkce popisující rovinnou křivku zadanou jako $f = f(x)$ platí následující vztah:

$$\kappa = \frac{\frac{d^2 f}{dx^2}}{\left(1 + \left(\frac{df}{dx}\right)^2\right)^{3/2}} \tag{4.1}$$

kde $\frac{df}{dx}$ a $\frac{d^2 f}{dx^2}$ jsou první a druhá derivace funkce f vzhledem k x.

V kontextu vyvíjené aplikace bude se jednat o výpočet hodnoty zakřivení křivky, reprezentující úsek cesty na mapě tvořený jednotlivými body s jejich souřadnicemi a hodnotami ze senzorů. Postup výpočtu vypadá následovně:

- Každé tři souřadnicové body tvoří trojúhelník. Pro každý trojúhelník existuje kružnice, která protíná všechny tři body. Poloměr této kružnice odpovídá poloměru křivky v tomto bodě. Tak je možné vypočítat poloměr křivosti pro každou trojici souřadnic.
- Poloměr křivosti každého segmentu(přímky mezi dvěma souřadnicovými body) je průměrem poloměrů trojúhelníků, jejichž je součástí.

- Zakřivení křivky jako celku se zjistí tak, že se sečtou délky jednotlivých segmentů, jejichž poloměr je menší než určitá hranice, nad kterou je křivka převážně rovná.

Popsaný přístup tedy v podstatě využívá geometrické vztahy zahrnující kružnice trojúhelníků k aproximaci zakřivení silnice na základě souřadnic jejích bodů. I když se nejedná o přímou aplikaci matematického vzorce pro křivost, využívá geometrické principy k dosažení podobného výsledku v kontextu odhadu křivosti silnice. Jak bylo uvedeno výše, je potřeba určit maximální hodnotu poloměru kružnice, která by popisovala „rovný“ segment silnice. Na základě provedených testů bylo rozhodnuto použít hodnotu 200 metrů. Další vstupní hodnotou je vzdálenost mezi souřadnicovými body, které budou použity k vytvoření jednotlivých segmentů. Tato hodnota se bude lišit v závislosti na typu jízdy. Pro běžnou jízdu na autodromu byla zvolena hodnota 2,5 metru mezi dvěma souřadnicovými body.

Pro výpočet vzdálenosti mezi dvěma souřadnicovými body byl použit Haversinův vzorec:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{4.2}$$

Kde:

- d - vzdálenost mezi dvěma body,
- r - poloměr Země,
- φ_1, φ_2 - zeměpisné šířky bodů v radiánech,
- λ_1, λ_2 - zeměpisné délky bodů v radiánech.

Pro malé vzdálenosti je Haversinův vzorec výpočetně efektivní a poskytuje rychlé výsledky. Ale při výpočtu velkých vzdáleností může docházet k nepřesnostem, zejména pokud jsou body umístěny na různých výškách nad úrovní moře. To je způsobeno tím, že Haversinův vzorec předpokládá, že Země má perfektní kulový tvar, což není úplně přesné. Skutečný tvar Země je elipsoidální, což může vést k chybám při výpočtu vzdáleností na velké vzdálenosti. Pro výpočet vzdálenosti mezi dvěma souřadnicovými body na autodromu je však více než vyhovující. Implementace Haversinova vzorce v jazyce C# je ilustrována na příkladě 4.1.

Kde:

- *lat1, lon1* - zeměpisné šířky a délky pro první bod,
- *lat2, lon2* - zeměpisné šířky a délky pro druhý bod,
- *eQuatorialEarthRadius* - poloměr Země u rovníku v kilometrech,
- *d2r* - konverzní faktor pro převod stupňů na radiány,
- *d* - výsledná vzdálenost mezi oběma body v metrech.

```
double Haversine(double lat1, double lon1, double lat2, double lon2)
  {
    double eQuatorialEarthRadius = 6378.1370D;
    double d2r = (Math.PI / 180D);
    double dlong = (lon2 - lon1) * _d2r;
    double dlat = (lat2 - lat1) * _d2r;
    double a = Math.Pow(Math.Sin(dlat / 2D), 2D) + Math.Cos(lat1 * _d2r
   ) * Math.Cos(lat2 * _d2r) * Math.Pow(Math.Sin(dlong / 2D), 2D);
    double c = 2D * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1D - a));
    double d = 1000D * _eQuatorialEarthRadius * c;

    return d;
  }
```

Ukázka zdrojového kódu 4.1: Implementace Haversinova vzorce v jazyce C#

Implementace algoritmu

Implementace tohoto algoritmu se skládá z několika klíčových tříd a komponent, které společně pracují na definování, měření a analýze křivosti křivek reprezentovaných sadou bodů:

- `Triangle.cs:` třída pro reprezentaci trojúhelníku s funkcí pro výpočet středu kružnice opsané a jejího poloměru. Tato třída je klíčová pro výpočet křivosti zatáček a změn směru mezi segmenty křivky.
- `Segment.cs:` reprezentuje část křivky mezi dvěma trojúhelníky, uchovává informace jako křivost, směr zatáčení, poloměr a délku segmentu. Segmenty se používají pro sestavení celých částí křivky s konzistentním směrem zatáčení.
- `CurvaturePart.cs:` skupina segmentů se stejným směrem zatáčení. Tato třída slouží k identifikaci a klasifikaci celkových částí křivky, poskytuje metody pro získání průměrného poloměru a celkové křivosti části.
- `Curvature.cs:` hlavní třída koordinující celý proces výpočtu křivosti. Obsahuje metody pro přidání bodů, výpočet segmentů a částí křivky, a poskytuje rozhraní pro získání informací o křivosti, směru zatáčení a úrovni křivosti u konkrétních bodů.
- `PointD.cs:` základní datová struktura reprezentující bod v prostoru s X, Y souřadnicemi a klíči pro interní a externí použití.
- `IDistance.cs:` rozhraní pro výpočet vzdálenosti mezi dvěma body v prostoru. Umožňuje flexibilitu v implementaci různých metod výpočtu vzdálenosti.
- `HaversineDistanceMeters.cs:` implementace IDistance používající Haversinovu formuli pro výpočet skutečné vzdálenosti mezi dvěma body na zemském povrchu ve metrech.

- `CurvatureEnvironment.cs:` třída, která obsahuje všechny potřebné komponenty pro výpočet křivosti, včetně implementace rozhraní IDistance pro výpočet vzdálenosti a definice úrovní křivosti pro klasifikaci segmentů.

Výpočetní proces je následující:

1. `Přidání bodů:` body jsou přidávány do systému, kde každé tři postupně přidané body jsou použity k vytvoření trojúhelníku.
2. `Výpočet trojúhelníků:` pro každou trojici bodů se vypočítá trojúhelník, určí se jeho opsaná kružnice, střed, poloměr a směr zatáčení.
3. `Formace segmentů:` na základě vypočítaných trojúhelníků se tvoří segmenty, které jsou analyzovány a klasifikovány podle směru zatáčení a poloměru.
4. `Konsolidace segmentů:` segmenty se stejným směrem zatáčení jsou spojeny do větších celků (CurvatureParts), které reprezentují konzistentní části křivky.
5. `Vyhodnocení a klasifikace:` celková struktura křivky je poté vyhodnocena a klasifikována na základě údajů o křivosti, poloměru a délce získaných z jednotlivých segmentů.

Nyní popíšeme koncový bod(endpoint), který je volán klientskou částí pro vizualizaci vystupních dat algoritmu. Metoda má 3 vstupní parametry: Id jízdy, minimální vzdálenost mezi dvěma body a práh pro přímé úseky.

- Metoda získává data o jízdě z databáze SQL pomocí *FromSql* metody Entity Frameworku, což umožňuje načíst data přímo pomocí SQL dotazu. Data jsou filtrována podle *RecordingId*, což je ID jízdy.
- Dále jsou odfiltrovány snímky, které jsou pro výpočet zakřivení irelevantní: jak je popsáno v případu použití, snímání se zapíná před samotnou jízdou kvůli kalibraci snímače. Filtrace probíhá na základě hodnoty rychlost, snímky vhodné pro další výpočty jsou snímky s hodnotou rychlosti vyšší než 8 km/h.
- Dalším krokem je filtrování snímků na základě vstupního parametru s minimální vzdáleností mezi dvěma doby. Pro výpočet vzdálenosti se používá Haversinův vzorec a pokud je tato vzdálenost větší než minimalní vzdálenost, provádí se průměrování souřadnic okolních bodů a přidává průměrovaný bod do výpočtu křivosti.
- Po přidání všech relevantních bodů se provede přepočet křivosti pro celou trasu. Následně se pro každý bod (a jeho segment) získávají různé metriky křivosti a směru zatáčení.
- Nakonec se vytváří seznam, jehož obsahem jsou data z původního seznamu, získaná z databáze a následně filtrovaná, doplněná o vypočtené hodnoty zakřivení a další potřebné informace, jako je směr otáčení. Tento seznam je odeslán jako odpověď na požadavek ze strany klienta ve formátu JSON.

Vizualizace je implementována pomocí knihovny Leaflet, která bude podrobněji popsána v podkapitole Vizualizace zpracovaných dat. Výsledek výstupu vyvinutého algoritmu je znázorněn na obrázku 4.2. Na obrázku vidíme jízdu na autodromu, kde jsou jednotlivé úseky různě barevně odlišeny v závislosti na stupni zakřivení a směru zatáčky. Zelená barva označuje minimální úroveň zakřivení a červená barva nejostřejší segment. Nepodbarvené segmenty jízdy znamenají, že jejich zakřivení je menší nebo rovno práhu pro přímé úseky. Směr zatáčky je označen dvěma barvami: modrou pro levotočivou a fialovou pro pravotočivou.

Obrázek 4.2: Vizualizace výsledku výpočtu zakřivení zatáček

4.2.2 Rozdělení jízdy na kola

Haversinův vzorec se používá také v následujícím implementovaném algoritmu pro rozdělení jízdy na závodní dráze do jednotlivých kol. V této podkapitole je uveden popis implementace tohoto algoritmu.

Často se jízda skládá z několika kol na autodromu a v případě vizualizace jízdy pomocí mapy se může křivka popisující jízdu protínat. Taková vizualizace může jen stěží poskytnout užitečné údaje pro vizuální analýzu jízdy. Příklad vizualizace jízdy složené ze tří kol je znázorněn na obrázku 4.3, který jasně ukazuje výše popsaný problém. Čáry znázorňující trasu řidiče se překrývají a neumožňují analyzovat celou jízdu. Popíšeme implementaci algoritmu, který tento problém řeší.

Popis a implementace algoritmu

- Vzhledem k tomu, že jízda může začít později než začátek snímání, je třeba nejprve určit skutečný okamžik zahájení jízdy. Způsob určení bodu začátku jízdy je totožný s implementací pro výpočet zakřivení křivky, tj. za počátek se považuje první snímek, jehož parametr rychlosti přesáhne 2,3 m/s. Index počátečního snímku je předán jako argument metodě, která rozdělí list snímků, rovněž předaný jako argument, do samostatných kol.

Obrázek 4.3: Vizualizace nerozdělené jízdy

- Metoda vstoupí do smyčky do-while, která zpracovává datové body ze vstupního seznamu snímku počínaje právě aktuálním indexem *currentStartIndex*. Pro každý datový bod vypočítá vzdálenost od aktuálního bodu záčatku kola pomocí Haversinova vzorce.
- Nové kolo se považuje za zahájené, pokud je vzdálenost od aktuálního bodu začátku kola menší než předem definované minimum *MinLapDistanceInMeters* a časová značka aktuálního datového bodu je alespoň o 10 sekund větší než časová značka v bodě s indexem *currentStartIndex*. Příznak *newLap* se nastaví na hodnotu **true**.
- Pokud je příznak *newLap* má hodnotu **true** a *currentLap* skládající z datových bodů obsahuje data, vytvoří se nový objekt *LapData*, který se přidá do seznamu kol, a aktualizují se různé proměnné pro zahájení sledování dalšího kola jako např. index aktuálního bodu záčatku kola.
- Každý datový bod se přidává do *currentLap*, dokud není spuštěna podmínka nového kola.

Seznam kol je list objektů *LapData*. Tento objekt obsahuje údaje o jednom kole: seznam jeho datových bodů, indexy startu a konce a dobu trvání v sekundách. Pořadí, v jakém jsou tyto objekty v seznamu uloženy, odpovídá jejich faktickému pořadí, takže v tomto případě nemá smysl ukládat číslo kola pro každý objekt.

Tento list je výstupní hodnotou metody, následně je list odeslán na stranu klienta pro následnou vizualizaci. Uživatelské rozhraní umožňuje uživateli rozdělit vybranou jízdu. Klientská část odešle odpovídající požadavek ovladači pomocí AJAX. Po obdržení odpovědi má uživatel možnost vybrat konkrétní kolo pro vizualizaci. Výsledek rozdělení tříkolové jízdy znázorněné na obrázku 4.3 je zobrazen na obrázku 4.4.

Obrázek 4.4: Vizualizace výsledku rozdělení jízdy na 3 kola

Snímek obrazovky vlevo, uprostřed a vpravo představují první, druhý a třetí kolo v uvedeném pořadí. Zelený kroužek na každém snímku představuje startovní bod kola, a červený kroužek reprezentuje koncový bod.

4.3 Vizualizace zpracovaných dat

Pro účely vizualizace získaných telemetrických dat zpracovaných popsanými algoritmy byla vytvořena webová stránka. Její uživatelské rozhraní vychází z požadavků popsaných v podkapitole 4.1.3.

4.3.1 Architektura aplikace

Aby bylo možné vytvořenou webovou stránku co nejsnáze integrovat do stávající aplikace, je její architektura založena na vzoru MVC a vypadá následovně:

- `Ovladač:` funkce ovladače zahrnují komunikaci s klientskou částí aplikace: zpracování požadavků a delegování provádění vnitřní logiky na služby. V tomto případě se jedná např. o odeslání dat vybrané uživatelem jízdy do klientské části.
- `Modely:` v kontextu aplikace slouží jednak jako reprezentace jednotlivých objektů z databáze, jednak pro přenos dat do klientské části. Příkladem může být model DriveData, který představuje jeden řádek z tabulky obsahující zachycená telemetrická data. Na druhé straně stejný model slouží jako objekt přenosu dat(DTO).
- `Zobrazení:` je HTML stránka s CSS styly a JavaScript kódem, jejímž úkolem je implementovat logiku vizualizace. Zobrazení je zodpovědné za zpracování interakce s uživatelem a za odesílání požadavků ovladači.

V případě rozsáhlejší aplikace by nebylo zbytečné vyčlenit obchodní logiku z ovladače do servisní vrstvy. V této aplikaci je však za provádění obchodní logiky zodpovědný ovladač.

4.3.2 Komunikace s klientskou stranou

Komunikace backendu s klientskou stranou je realizována pomocí ovladače. Ovladač je třída, jejíž jednotlivé metody představují jednotlivé koncové body. Inicializace konstruktoru ovladače je znázorněna na příkladu kódu 4.2. Konstruktor ovladače přijímá dva parametry:

- `ILogger<HomeController> logger:` tento parametr umožňuje zaznamenávání informací o provozu a chybách v controlleru. Interface *ILogger* je běžný vzor pro logging v aplikacích .NET, který podporuje různé úrovně logování (např. informace, varování, chyby).
- `SqlDbContext context:` parametr je instancí *SqlDbContext*, která slouží jako hlavní přístupový bod pro databázi v aplikaci pomocí Entity Framework. Pomocí tohoto kontextu lze provádět operace s databází jako jsou dotazy, vkládání dat, aktualizace a mazání.

Tento konstruktor je příkladem techniky vkládání závislostí. Vkládání závislostí používá konstruktor třídy k inicializaci objektů a poskytování požadovaných závislostí objektu, což znamená, že umožňuje „injektovat“ závislost mimo třídu [28]. V těle konstruktoru jsou tyto parametry přiřazeny k odpovídajícím privátním proměnným *currentLogger*, *currentContext*, a to umožňuje používat vložené závislosti v kontextu ovladače.

```
1 public HomeController(ILogger<HomeController> logger, SqlDbContext
      context)
2 {
3   currentLogger = logger;
4   currentContext = context;
5 }
```

Ukázka zdrojového kódu 4.2: Konstruktor ovladače

Nyní, když jsme inicializovali konstruktor kontroléru, přejděme k popisu koncových bodů. Koncový bod je funkce, která je volána požadavkem AJAX ze strany klienta pomocí JavaScript kódu. Příkladem takového koncového bodu může být koncový bod, který je vyvolán, když uživatel vybere konkrétní jízdu pomocí uživatelského rozhraní. Implementace tohoto koncového bodu je ilustrována příkladem 4.3. Metoda obdrží jako argument ID cesty vybrané uživatelem. Poté pomocí výše uvedeného vnořeného přístupového bodu pro databázi získá údaje o vybrané jízdě. Následně tato data odešle jako odpověď na požadavek klientské části ve formě listu jednotlivých bodů patřících k dané jízdě.

```
public List<object> FetchRecordingData(String selectedRecording)
{
  var driveData = _context.DriveData.FromSql(
    $"SELECT * FROM dbo.DriveData WHERE RecordingId = {Int32.Parse(
    selectedRecording)}")
    .ToList();
  List<object> data = new List<object>
    {
    driveData
  };
  return data;
}
```

Ukázka zdrojového kódu 4.3: Příklad koncového bodu

Nyní se podívejme, jak se odesílá požadavek na načtení dat vybrané jízdy pomocí AJAX. AJAX umožňuje uživateli webové aplikace komunikovat s webovou stránkou bez přerušení neustálým načítáním webové stránky. Interakce s webovou stránkou probíhá rychle, pouze části stránky se znovu načítají a obnovují [29]. Na ukázce kódu 4.4 je funkce, která se volá, když uživatel vybere nebo změní jízdu, kterou chce vizualizovat. Nejprve se pomocí jQuery získává ID vybrané jízdy z rozbalovacího menu a poté se odešle požadavek AJAX. Požadavek je konfigurován následovně:

- `contentType: "application/json"`: určuje, že typ obsahu posílaného na server je JSON.
- `dataType: "json"`: očekává, že server odpoví JSON formátem.
- `method: 'GET'`: typ požadavku HTTP, který je v tomto případě GET.
- `cache: false`: zabrání prohlížeči, aby si ukládal výsledky do cache a zajistí, že data budou vždy načítána přímo ze serveru.
- `url: '@Url.Action("FetchRecordingData", "Home")'`: URL, na kterou se požadavek odesílá. Tato URL je dynamicky generována pomocí funkce ASP.NET MVC Url.Action, který vytváří URL pro akci FetchRecordingData v ovladači Home.
- `data: { selectedRecording: _selectedRecording }`: data odeslaná na server, která obsahují identifikátor vybrané jízdy.

Když server úspěšně zpracuje požadavek a vrátí data, data z odpovědi jsou uložena do proměnné pro následné zpracování. V případě, že AJAX volání selže (např. kvůli síťové chybě nebo chybě na serveru), chybová zpráva se zaloguje do konzole prohlížeče. Zobrazená forma interakce klient-server platí i pro další funkce, které reagují na akce uživatele. Jediným rozdílem je typ odesílaných a přijímaných dat a samozřejmě také adresa URL.

```
function SelectedRecordingChanged() {
  _selectedRecording = $('#SelectedRecording').find(':selected').text()
    ;
  $.ajax({
    contentType: "application/json",
    dataType: "json",
    method: 'GET',
    cache: false,
    url: '@Url.Action("FetchRecordingData", "Home")',
    data: { selectedRecording: _selectedRecording },
    success: function (response) {
      data = response;
    },
    error: function (error) {
      console.log(error);
    }
  });
}
```

Ukázka zdrojového kódu 4.4: Příklad odeslání požadavku pro získaní dat vybrané jízdy

4.3.3 Implementace vizualizační logiky

Jak je uvedeno v podkapitole Stanovení požadavků, vizualizace telemetrických dat by měla být realizována pomocí grafů a mapy. K implementaci těchto grafických prvků na stránce bylo rozhodnuto použít knihovny třetích stran s volně přístupným zdrojovým kódem.

Implementace mapy

Leaflet je knihovna pro prezentaci mapových dat. Data spolu se základní mapovou vrstvou musí poskytnout vývojář. Mapy se skládají z dlaždicových vrstev spolu s podporou prohlížeče, výchozí interaktivitou, možností posouvání a zvětšování [30]. Tato funkcionalita, která je k dispozici bez rozšíření a pluginů, předurčila zvolení této knihovny. Inicializační kód mapy je zobrazen na ukázce 4.4.

Prvním krokem při implementaci mapy je její inicializace. Pro mapu se používá dlaždicový server Google. Dlaždice jsou načítány z poddomén mt0 až mt3. Za to je zodpovědná funkce *L.map()*, která vykreslí mapu do prvku HTML s ID *graph-gps*.

Poté se do mapy přidají souřadnicové body jízdy a vzájemně se propojí. Body, mezi nimiž je interval jedné sekundy, jsou zvýrazněny kroužkem. Pro každý kroužek přidaný do mapy je definována klikací událost, která jej zvýrazní a zobrazí o něm informace. Funkce aktualizuje prvky uživatelského rozhraní, jako je graf, posuvník a pole hodnot, v závislosti na vybraném bodu. Následně se na vykreslenou jízdu nanese čára znázorňující úroveň zakřivení s její barvou. Vedle ní se navrství širší čára znázorňující směr zakřivení.

```
function initMap() {
    map = L.map('graph-gps').setView([50, 15], 13);
    L.tileLayer('http://{s}.google.com/vt/lyrs=s&x={x}&y={y}&z={z}', {
        maxZoom: 21,
        subdomains: ['mt0', 'mt1', 'mt2', 'mt3'],
    }).addTo(map);
    addLegend();
    drawCompleteTrack();
    drawTrackPoints();
    drawCurvature();
}
```

Ukázka zdrojového kódu 4.5: Inicializace mapy

Posledním krokem je přidání legendy, která bude popisovat vizualizovanou jízdu nebo případně druhou jízdu vybranou jako referenční. Legenda se inicializuje pomocí příkazu *L.control();*, který jako argument přebírá její umístění v mapě. Poté se legenda, stejně jako všechny ostatní přidané prvky, jako jsou čáry znázorňující trasu, kružnice znázorňující jednotlivé body na mapě, přidá do vrstvy na mapě pomocí příkazu *element.addTo(map);*. Výsledek implementace mapy s vizualizovanou jízdou je zobrazen na obrázku 4.2.

Implementace grafů

K implementaci grafů byla použita knihovna Chart.js. Chart.js je bezplatná knihovna JavaScriptu pro tvorbu grafů v jazyce HTML. Je to jedna z nejjednodušších vizualizačních knihoven pro JavaScript a obsahuje vestavěné typy grafů jako např. rozptylový graf, čárový graf, sloupcový graf, koláčový graf a jiné [31]. V této aplikaci budou použity pouze čárové grafy, proto se zaměřme na jejich implementaci. Ukažme si přidání grafu na příkladu kódu z aplikace, který je zobrazen na ukázce 4.5.

Funkce přijímá jako argument objekt *graphData* obsahující všechny atributy pro inicializaci grafu:

- Název zobrazených telemetrických dat, který se zobrazuje jako štítek v legendě grafu.
- Pole číselného typu obsahující hodnoty telemetrických pro vykreslení v grafu.
- Pole časových značek pro osu X grafu.
- Grafické vlastnosti grafu: barvy, průhlednost, napětí čar a stylizace bodů (velikost a barva).

Graf obsahuje funkci pro přiblížení a oddálení, která je aktivována prostřednictvím kolečka myši nebo špetky na dotykovém zařízení, a také možnost posunutí. Při kliknutí na bod v grafu se spustí funkce *handleGraphPointClick*. Při detekci kliknutí na bod tato funkce vyčistí vrstvu markerů(velikosti a barvy bodů) a získá

informace o bodě, se kterým uživatel interagoval. Následně změní barvu a velikost zvýrazněného bodu a aktualizuje graf.

```
function initGraph(graphData) {
    var ctx = document.getElementById('myChart');
    myGraph = new Chart(ctx, {
      type: 'line',
      data: graphData,
      options: {
        plugins: {
          zoom: {
            zoom: {
              wheel: {enabled: true},
              pinch: {enabled: true},
              mode: 'xy',
            },
            pan: {enabled: true}
          }
        },
        maintainAspectRatio: false,
        onClick: handleGraphPointClick
      }
    });
}
```

Ukázka zdrojového kódu 4.6: Inicializace grafu

Důležitou funkcí je také zvýraznění ostatních hodnot telemetrických dat ve vybraném bodě. Stejně jako při kliknutí na bod na mapě se při kliknutí na bod v grafu zobrazí data ve vybraném bodě v celém rozhraní stránky.

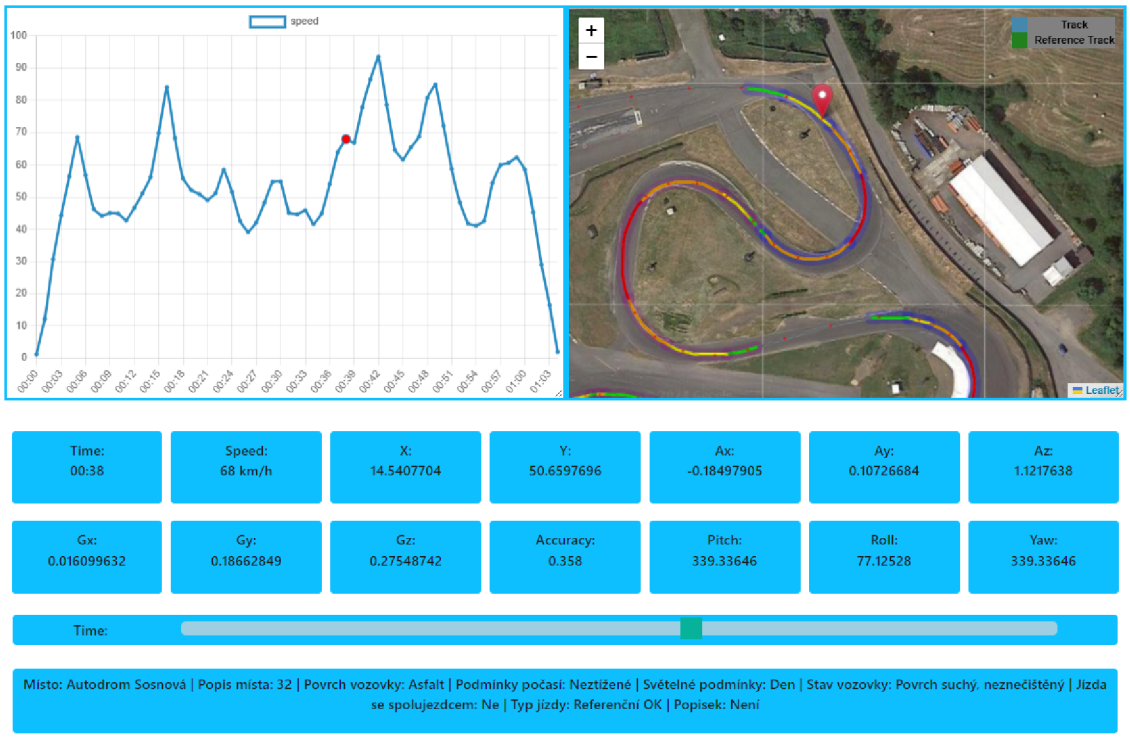
Implementace časového posuvníku

Posuvník času slouží k označení bodu v grafu, na mapě a k zobrazení hodnot telemetrických dat ve zvolené sekundě jízdy. Je implementován pomocí HTML prvku *input* s typem *range*. Tento vstupní typ vytváří prvek rozhraní posuvníku. Problém je, že při změně hodnoty posuvníku se nová hodnota nezobrazí. V současné době neexistuje žádný způsob, jak to provést pomocí HTML5 [32]. Této funkcionality však lze dosáhnout pomocí JavaScript kódu. Implementace posuvníku je znázorněná na ukázce 4.6. Důležitá je zde funkce, která je volána při změně hodnoty. Hodnoty posuvníku jsou pole čísel představující počet snímků s telemetrickými daty. Funkce přijímá novou hodnotu představující číslo snímku v poli, vyhledává tento snímek v poli všech snímků a poté formátuje jeho časovou značku do tvaru MM:SS.

```
<div class="inline-content rounded">
  <label class="time-label">Time: </label>
  <input type="range" min="1" max="100" value="50" class="timeline-
    slider" id="slider" oninput="updateDataFields()">
</div>
```

Ukázka zdrojového kódu 4.7: Implementace časového posuvníku

Výsledkem je uživatelsky přívětivý prvek uživatelského rozhraní, který umožňuje analyzovat údaje o jízdě a telemetrii v kterémkoli bodě jízdy. Realizací výše uvedených prvků vzniká stránka, na které se všechny grafické prvky vzájemně ovlivňují. Pokud je vybrán bod na mapě, zobrazí se odpovídající bod na grafu. Kromě toho se změní hodnota posuvníku a zobrazí se telemetrické údaje ve vybrané sekundě. Vzhled hotové stránky je znázorněn na obrázku 4.5.



Obrázek 4.5: Vzhled hotové stránky pro vizualizaci dat

Obrázek 4.5 zobrazuje vizualizaci jízdy, která se skládá z jednoho kola. Mapa zobrazuje její trasu na autodromu, kde jednotlivé úrovně zakřivení zatáček jsou zvýrazněny samostatnými barvami. Bod zvýrazněný značkou představuje sekundu jízdy zvolenou posuvníkem času. Stejný bod ve zvolené sekundě se zobrazuje také na grafu popisujícím rychlost, kde je zvýrazněn červenou barvou. Obdélníky níže zobrazují ostatní telemetrické data ve vybraném bodě. Po kliknutí na některý z nich se zobrazí graf odpovídajících telemetrických dat. Vybraný bod lze změnit nejen pomocí posuvníku času, ale také kliknutím na bod na mapě nebo v grafu. Pod posuvníkem času se zobrazují metadata o jízdě popisující místo a podmínky jízdy.

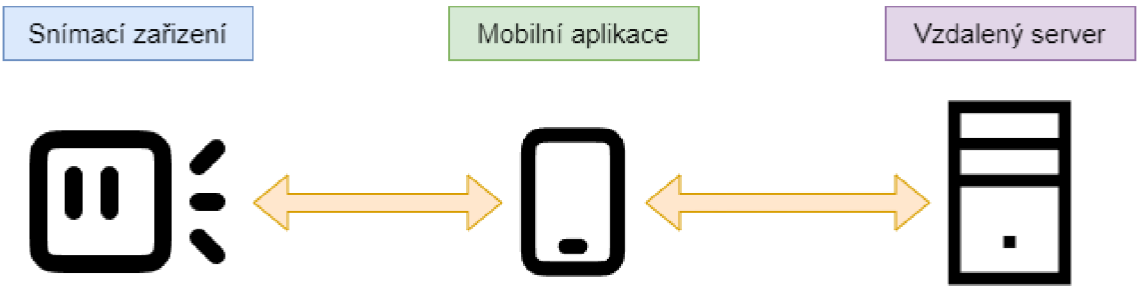
5 Mobilní aplikace

Tato kapitola bude věnována vývoji grafického rozhraní mobilní aplikace pro komunikaci se senzorovým zařízením. Nejprve se seznámíme se stávající aplikací na zastaralé platformě Xamarin, s její strukturou a funkcionalitou, poté stanovíme požadavky na uživatelské rozhraní a nakonec se vrhneme na implementaci nové aplikace napsané na platformě .NET MAUI.

5.1 Existující prototyp

Vzhledem k tomu, že nová aplikace je přepracovanou verzí staré aplikace implementované na nové platformě, je užitečné se nejprve seznámit s její předchůdkyní.

V telemetrickém systému popsaném v 2. kapitole hraje mobilní aplikace roli vzdáleného ovladače, jehož úkoly zahrnují komunikaci se senzorovým zařízením, jeho zapnutí/vypnutí a odesílání dat získaných ze senzorů na vzdálený server. Umístění mobilní aplikace v telemetrickém systému je znázorněno na obrázku 5.1.



Obrázek 5.1: Mobilní aplikace v telemetrickém systému

Aplikace byla vyvinuta na platformě Xamarin. Vzhledem k tomu, že v době psaní tohoto textu se blíží konec podpory této platformy ze strany vývojářů společnosti Microsoft [33], bylo rozhodnuto o migraci mobilní aplikace na platformu .NET MAUI.

5.1.1 Struktura rozhraní

Logika mobilní aplikace je rozdělena do několika částí. Každá část je v aplikaci reprezentována samostatnou stránkou. Přepínání mezi stránkami se provádí pomocí

navigačního menu v dolní části každé stránky. K popisu struktury celé aplikace stačí popsat součásti tohoto navigačního menu.

Nahrávání

Stránka má dva stavy. První obsahuje pole pro zadání metadat jízdy a zahájení snímání. Druhý stav je stránka po zahájení snímání, která zobrazuje data ze snímačů v reálném čase. Stránka je hlavní stránkou celé aplikace, její vzhled v nové verzi dozná mnoha změn.

Uživatel

Další složka v navigačním menu je stránka pro autorizaci uživatele. Pro zahájení snímání musí být definován uživatel, protože v databázi nemohou být žádné jízdy bez přiřazeného uživatele. Přihlášení se provádí zadáním e-mailové adresy. Autorizace pomocí hesla není vyžadována, protože aplikace bude distribuována pouze důvěrně známým uživatelům.

Zařízení

Tato složka slouží k výběru zařízení, které se má použít pro snímání. Lze vybrat interní nebo externí zařízení. Po výběru interního zařízení budou k snímání použity senzory telefonu. Pro výběr externího zařízení je potřeba nejprve zapnout Bluetooth a poté vybrat konkrétní zařízení z blízkého okolí.

Jízdy

Poslední složka slouží k zobrazení předchozích cest. Je to v podstatě seznam všech minulých cest a jejich metadat. Zde je také možné upravovat metadata jednotlivých jízd, protože uživatel nemusí být vždy schopen zadat správné údaje o jízdě před jejím zahájením.

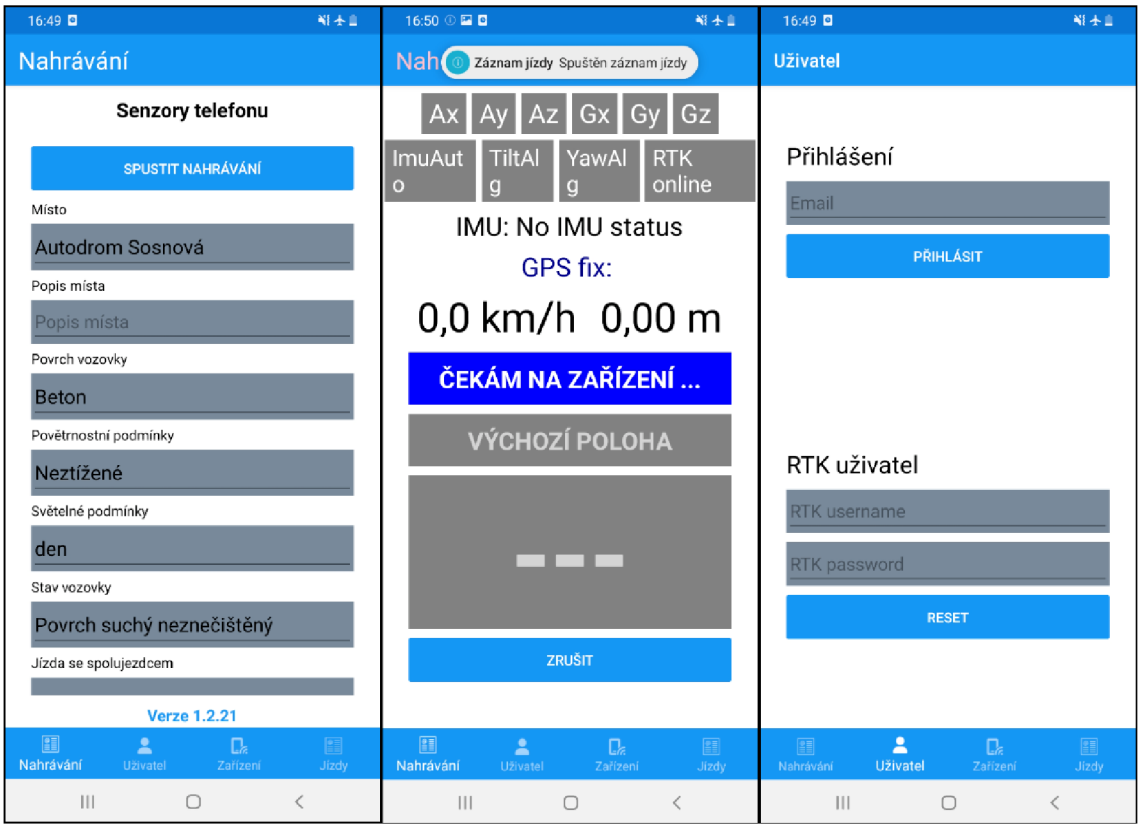
5.1.2 Funkcionalita

Když už známe komponenty aplikace, stojí za to popsat její typický případ použití. Je poměrně specifický, stejně jako požadavky na uživatelské rozhraní, které si vysvětlíme v příští podkapitole. Popíšeme počáteční podmínky tohoto use-case: uživatel (zároveň řidič) se nachází na závodní trati na startovní pozici za velmi slunečného počasí a musí vyplnit metadata jízdy, zapnout zařízení a počkat na jeho kalibraci. Celý proces zaznamenání jízdy rozdělit do několika kroků:

- `Přihlášení:` uživatel se přihlásí pomocí své e-mailové adresy registrované v databázi.
- `Nastavení snímacího zařízení:` uživatel vybere snímácí zařízení. Většinou samozřejmě se jedná o externí zařízení, a proto ten krok zahrnuje zapnutí Bluetooth.

- `Ručné vyplnění metadat:` řidič vyplní metadata jako např. místo jízdy, podmínky počasí, stav vozovky nebo číslo kola.
- `Zapnutí snímacího zařizení pomocí mobilní aplikace:` řidič následně zapne zařízení a počká, až se jednotlivé snímače zkalibrují.
- `Samotná jízda:` po kalibraci senzorů začne zařízení odesílat zaznamenané hodnoty na vzdálený server. Uživatel se začne pohybovat.
- `Ukončení jízdy:` uživatel ukončí jízdu a vypne zařízení v mobilní aplikaci. Tím se zastaví odesílání dat na server.

V rámci této práce se funkčnost aplikace nezmění, takže tento případ použití zůstane relevantní i pro novou aplikaci. Jednotlivé stránky původní aplikace jsou zobrazeny na obrázku 5.2. Můžeme pozorovat, že kvůli světlému pozadí je aplikace v rámci výše popsaného případu použití obtížně použitelná. Stránka Nahrávání navíc obsahuje nadměrný počet vstupních polí. Řešení těchto problémů je součástí vývoje nového rozhraní, protože tvoří některé z požadavků, o nichž bude řeč v další podkapitole.



Obrázek 5.2: Uživatelské rozhraní původní mobilní aplikace

5.2 Formulace požadavků

Při vytváření rozšíření pro mobilní aplikaci je třeba nejen proanalyzovat původní aplikaci, ale také určit konkrétní požadavky. V této podkapitole popíšeme požadavky, které byly původně zadány, a požadavky, které byly obdrženy v průběhu vývoje v rámci konzultací.

5.2.1 Uživatelské rozhraní

Základní požadavky na uživatelské rozhraní vycházejí z výše popsaného scénáře použití a ze zkušeností s původní mobilní aplikací. Lze je shrnout následovně:

Kontrast

Pro zajištění optimální viditelnosti a použitelnosti je nezbytné, aby bylo uživatelské rozhraní navrženo s vysokým kontrastem mezi textem a jeho pozadím. To je zvláště důležité v podmínkách silného osvětlení, jako je přímé sluneční světlo, kde nízký kontrast může značně snižovat čitelnost a tím pádem i uživatelský komfort a efektivitu interakce. To zahrnuje nejen text, ale i ikony, tlačítka a další interaktivní prvky, které musí být navrženy s ohledem na optimální viditelnost za všech okolností.

Optimalizace uživatelské zkušenosti

Uživatel by měl s aplikací pracovat co nejrychleji, což v případě výše uvedeného scénáře znamená, že všechny ovládací prvky, se kterými uživatel manipuluje, by měly být implementovány tak, aby jejich vyplnění trvalo co nejkratší dobu.

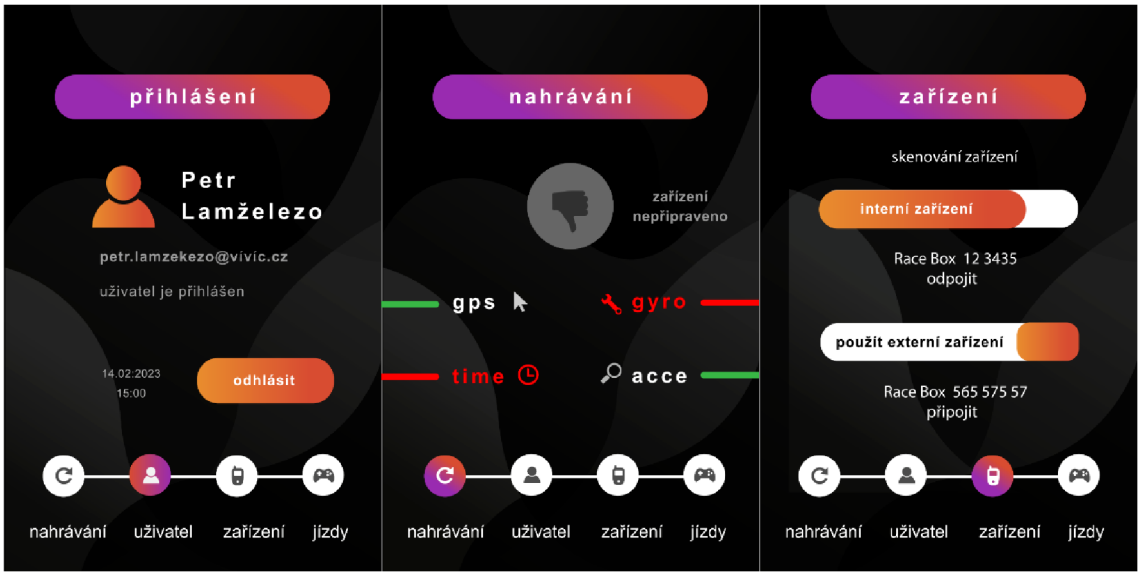
Použití navržených grafických prvků

Vzhled aplikace, tj. styl ikon, obrázků a pozadí by měl být implementován na základě návrhu poskytnutým grafickým designérem. Návrhy jsou poskytnuty jako SVG obrázky jednotlivých stránek aplikace.

Obrázek 5.3 ukazuje požadovaný vzhled aplikace podle návrhu. Vlevo je přepracovaná stránka obsahující údaje přihlášeného uživatele, jako je jméno uživatele a e-mail. Obrázek uprostřed ukazuje stránku, která se zobrazí po spuštění nahrávání. Zobrazují se na ní jednotlivé telemetrické údaje, kde barva čárky označuje stav kalibrace. Obrázek vpravo ukazuje stránku s nabídkou snímacích zařízení, které následně bude použito k záznamu telemetrických dat během jízdy.

5.2.2 Implementace

Jak bylo uvedeno výše, v rámci vývoje nové mobilní aplikace je cílem vytvořit nové uživatelské rozhraní na platformě .NET MAUI. Volba platformy vychází ze skutečnosti, že původní aplikace byla vyvinuta na platformě Xamarin, která je jejím předchůdcem. To umožní vývojáři logiky mobilní aplikace její snadnou migraci na nové rozhraní, protože .NET MAUI předpokládá možnou migraci starých aplikací



Obrázek 5.3: Požadovaný vzhled mobilní aplikace

a má k tomu odpovídající funkce. K tomuto účelu je k dispozici konzolový nástroj .NET Upgrade Assistant který si projekt nakonfiguruje samostatně [34].

Migrace na novou platformu s sebou nevyhnutelně nese refaktorizaci kódu, protože ne všechny funkce platformy Xamarin se uplatnily na platformě .NET MAUI. Aby se tomu předešlo, bylo rozhodnuto vytvořit zcela nový projekt. Pro tento projekt byly získány následující požadavky:

Konzistence s předchozí verzí

Aby byl přechod na novou mobilní aplikaci co nejjednodušší, určité prvky uživatelského rozhraní by měly být konzistentní s minulou verzí aplikace. To se částečně týká struktury rozhraní, včetně navigace mezi jednotlivými stránkami.

Ovládací prvky

Uživatelské rozhraní by se mělo skládat z nejnovějších ovládacích prvků, které se mapují na nativní ovládací prvky každé cílové platformy [35].

Rozšíření funkcionality

Pro zlepšení uživatelského zážitku při používání aplikace by měla být přidána možnost dodatečně měnit metadata jízdy. Při běžném používání aplikace bylo zjištěno, že je často nejen obtížné, ale i zbytečné vyplňovat všechna metadata jízdy před jejím zahájením, protože většinu těchto údajů lze vyplnit dodatečně po skončení jízdy.

5.3 Vývoj nového rozhraní

Nyní, když známe funkčnost původní aplikace a požadavky na nové rozhraní, přejděme k popisu implementace nové aplikace. Začněme popisem implementace komponent, které definují strukturu aplikace.

5.3.1 Navigace

Na obrázku 5.3 je vidět, že všechny stránky mají jeden společný prvek - navigační menu. Platforma .NET nabízí komponentu AppShell, která má řadu funkcí pro vytváření navigace v aplikaci.

Deklarace struktury navigačního menu, stejně jako všech ostatních prvků uživatelského rozhraní, je implementována v jazyce XAML. Výsledná navigace v souboru *AppShell.xaml* je zobrazena na příkladu kódu 5.1.

```
<TabBar>
    <Tab Icon="recording.svg" Title="Nahrávání">
        <ShellContent ContentTemplate="{DataTemplate local:MainPage}"/>
    </Tab>
    <Tab Icon="drives.svg" Title="Jízdy">
        <ShellContent ContentTemplate="{DataTemplate local:DrivesPage}"
    />
    </Tab>
    <Tab Icon="others.svg" Title="Ostatní">
        <ShellContent ContentTemplate="{DataTemplate local:OptionsPage}
    "/>
    </Tab>
</TabBar>
```

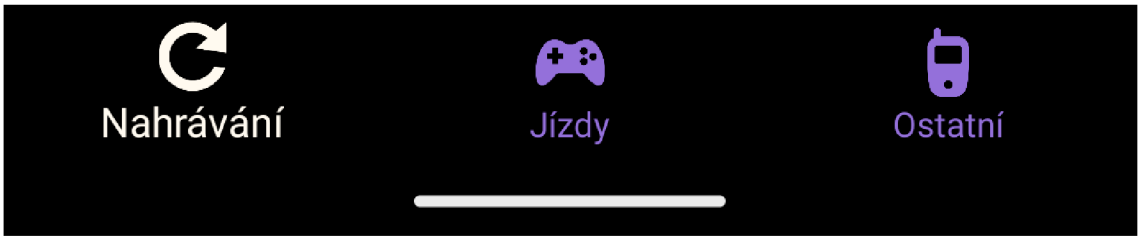
Ukázka zdrojového kódu 5.1: Struktura navigačního menu

Kde *<Tab/>* reprezentuje jednotlivé složky menu a *<ShellContent/>* určuje, jaký typ stránky má být instanciován a zobrazen.

AppShell.xaml.cs funguje jako kódové zázemí pro vizuální strukturu navigace. Kromě metody *InitializeComponent()*, která je volána v konstruktoru každé třídy pro inicializaci komponent definovaných v souboru XAML, zde jsou definovány vlastní trasy(route) pro navigaci. Tento mechanismus umožňuje definovat pojmenované trasy, které lze použít k navigaci mezi stránkami aplikace, např. kliknutím na tlačítko.

Routing.RegisterRoute("UserInfo", typeof(UserInfo)); je příkladem definování navigační trasy. Prvním parametrem je unikátní identifikátor trasy a druhý se používá pro specifikaci typu stránky, na kterou bude aplikace navigovat, když je trasa vyvolána. Důvodem pro definování tras je skutečnost, že v aplikaci existují podstránky, na které lze přejít bez použití menu.

Obrázek 5.4 ukazuje dokončené navigační menu. Můžeme si všimnout, že výsledek se liší od návrhu grafika. Důvodem je dodržení filozofie platformy .NET MAUI. Platforma nabízí použití hotových grafických a programových komponent, které po kompilaci do nativního kódu cílových platforem mapují jednotnou implementaci komponenty .NET MAUI na komponenty jednotlivých platforem. Vývoj vlastní grafické a kódové implementace komponenty proto přináší velké množství problémů s kompatibilitou.



Obrázek 5.4: Vypracované navigační menu

5.3.2 Implementace vzhledu jednotlivých stránek

Dále budeme pokračovat popisem vývoje jednotlivých stránek aplikace. Struktura aplikace je navržena tak, aby všechny stránky aplikace používaly společné styly a komponenty. Začněme proto popisem jednotlivých komponent.

Pozadí

Pozadí je jednotné pro všechny stránky aplikace a je vytvořeno přesně podle návrhu grafika. K přidání obrázku na pozadí byla použita komponenta *Grid*, která umožňuje organizovat obsah do řádků a sloupců. Ná ukázce 5.2 je zobrazena implementace pozadí.

```
1 <Grid>
2     <Image x:Name="backgroundImage" Source="background.png" Aspect="
    AspectFill"></Image>
3     <!-- Struktura specifická pro konkretní stránku -->
4 </Grid>
```

Ukázka zdrojového kódu 5.2: Implementace pozadí

V tomto případě žádné specifické definice řádků nebo sloupců nejsou nastaveny, takže *Grid* funguje jako jednoduchý kontejner pro umístění dalších prvků. Jedním z těchto prvků je obrázek, který je nastaven jako pozadí. Atribut *Source* určuje zdroj obrázku, který má být zobrazen a *Aspect* s hodnotou *AspectFill* znamená, že obrázek bude vyplňovat dostupný prostor tak, že zachová poměr stran obrázku, ale může některé části obrázku oříznout, pokud rozměry obrázku nesouhlasí s rozměry přiděleného prostoru.

Hlavičky

Hlavičky se používají k reprezentaci názvu stránky a jsou implementovány jako neklikatelná tlačítka. Důležitým vizuálním parametrem je gradient pozadí. Tlačítko používá pro pozadí *LinearGradientBrush*, což je lineární gradient, který přechází mezi dvěma barvami. První barva gradientu je světle fialová, aplikovaná na začátku gradientu, druhá barva gradientu je oranžová, aplikovaná na konci gradientu. Díky gradientu je tlačítko výrazným prvkem uživatelského rozhraní a při použití za slunečného počasí je dostatečně kontrastní. Implementace je zobrazena na ukázce 5.3.

```
<Button x:Name="SensorsBtn"
    Text="JÍZDY"
    HorizontalOptions="Center"
    WidthRequest="250"
    CornerRadius="30"
    Margin="0, 0, 0, 15">
    <Button.Background>
        <LinearGradientBrush StartPoint="0,0"
            EndPoint="1,0">
            <GradientStop Color="#8A26ED"
                Offset="0" />
            <GradientStop Color="#d74d35"
                Offset="1" />
        </LinearGradientBrush>
    </Button.Background>
</Button>
```

Ukázka zdrojového kódu 5.3: Implementace hlaviček

Tlačítka

V aplikaci jsou 2 typy tlačítek, které se liší důležitostí a vizuální prioritou na stránce. Pozadí nejdůležitějších tlačítek z hlediska funkčnosti je realizováno pomocí reverzního gradientu z hlaviček. Tlačítka, která hrají roli při navigaci mezi podstránkami, mají méně nápadný gradient dvou podobných barev, například tlačítka pro přechod z podstránky na nadřazenou stránku mají gradient z červené na oranžovou. Na ukázce 5.4 je příklad navigačního tlačítka v aplikaci. Důležitým atributem tlačítka je *Clicked*, který spojuje tlačítko jako prvek uživatelského rozhraní s kódem, který naviguje na stránku nastavení metadat pomocí jedné z výše popsaných navigačních tras.

Zadávací pole

V aplikaci je několik stránek, kde se provádí uživatelský vstup. Uživatel musí například zadat svůj e-mail pro autorizaci nebo zadat metadata při zahájení snímání.

Používají se dva typy vstupních polí: jedno využívá zadávání textu z klávesnice a druhé výběr ze seznamu nabízených hodnot. Pro urychlení práce uživatele s aplikací bylo rozhodnuto použít co nejvíce polí s možností výběru hodnoty. Výjimkou je již zmíněné zadání email adresy a také pole s popisem jízdy, který může být pro

```
<Button
    x:Name="RecordingSettingsBtn"
    Text="VLASTNOSTI JÍZDY"
    SemanticProperties.Hint="Stránka s nastavením metadat"
    Clicked="OnRecordingSettingsButtonClicked"
    HorizontalOptions="Center"
    WidthRequest="250"
    CornerRadius="30"
    Margin="0, 25, 0, 25">
   <Button.Background>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Color="#e78734" Offset="0"/>
            <GradientStop Color="#d74d35" Offset="1"/>
        </LinearGradientBrush>
    </Button.Background>
</Button>
```

Ukázka zdrojového kódu 5.4: Příklad tlačítka

každou jízdu specifický. Pole s výběrem hodnot je zobrazeno na ukázce 5.5.

Výsledek implementace výše popsaných grafických prvků je znázorněn na obrázku 5.5. Na levém snímku obrazovky si můžeme všimnout rozdělení stylů tlačítek podle jejich priority. Odráží realitu používání aplikace: uživatel stiskne tlačítko *Upravit vlastnosti* jen v některých případech, zatímco tlačítko *Spustit nahrávání* se bude používat ve všech případech a mělo by být jasně vizuálně odlišeno od méně používaných.

Na snímku obrazovky uprostřed můžeme pozorovat rozpracovanou implementaci stránky po spuštění nahrávání a identifikovat konkrétní rozdíly od návrhu grafika. Tyto rozdíly ale vycházejí z požadavků odvozených z konzultací s vývojáři původní aplikace, kteří jsou zároveň jejími uživateli. Při nahrávání bude uživatel potřebovat pouze přesnost určení polohy a rychlost jízdy. Kromě toho bude uživatel potřebovat vidět stav kalibrace GPS zařízení a RTK.

Snímek obrazovky vpravo ukazuje stránku složenou z polí s výběrem hodnoty z navrhovaného seznamu s výjimkou pole *Popis místa*. Rychlost vyplnění metadat jízdy je tímto přístupem znásobena, tím, že jim se jim ušetří čas při vymýšlení nebo psaní na klavesnici. Navíc se tím zkrácí čas strávený ověřováním dat na backendu při vyplňování formulářů [36]. Uživatelé totiž jednoduše vybírají ze seznamu předem vybraných hodnot. To je velmi užitečná výhoda i při filtrování metadat v databázi: pokud má uživatel k dispozici omezený seznam hodnot, nemůže dojít k překlepu, který by toto filtrování ovlivnil.

```
<Label Text="Weather Condition" Margin="0, 5, 0, 0"
    TextColor="LightGray"/>
<Border StrokeShape="RoundRectangle 10, 10, 10, 10"
    BackgroundColor="LightGray"
    Padding="0">
    <Picker x:Name="WeatherPicker"
        TextColor="Black"
        BackgroundColor="LightGray"
        Margin="10, 0, 10, 0">
        <Picker.ItemsSource>
            <x:Array Type="{x:Type x:String}">
                <x:String>Clear</x:String>
                <x:String>Rain</x:String>
                <x:String>Light rain</x:String>
                <x:String>Gusty wind</x:String>
                <x:String>Fog</x:String>
                <x:String>Other complicated</x:String>
            </x:Array>
        </Picker.ItemsSource>
        <Picker.SelectedIndex>0</Picker.SelectedIndex>
    </Picker>
</Border>
```

Ukázka zdrojového kódu 5.5: Příklad pole s výběrem hodnoty ze seznamu

5.3.3 Rozšíření funkcionality

Přestože v rámci vývoje mobilní aplikace bylo úkolem vytvořit design s využitím nových grafických prvků vycházejících z návrhu grafika, během konzultací byla navíc identifikována potřeba rozšířit funkční část aplikace. Jedná se o možnost rozšíření nebo opravy metadat již zaznamenané jízdy. Pro implementaci byl zvolen návrhový vzor MVVM popsaný v 2. kapitole. Probereme si jeho jednotlivé komponenty a jejich úlohu.

Model

Modelem je C# třída reprezentující objekt, který obsahuje jednotlivá metadata jízdy: unikátní identifikátor, typ jízdy, její popis a podmínky počasí, mezi dalšími.

Zobrazení

Zobrazení se skládá ze dvou tříd. Zobrazují data z modelů v grafickém formátu a přijímají uživatelské vstupy, které pak mohou být předány do modelů nebo view modelů. Třída *DrivesPage* je samostatná stránka aplikace, která je seznamem posledních zaznamenaných jízd. Na obrázku 5.6 na snímku vlevo je výsledný vzhled stránky. Zde můžeme vidět seznam minulých jízd s vizuálně odděleným číslem jízdy, jako tomu bylo v návrhu grafika. Třída *DriveDetailsPage* je také stránka, která se otevře po výběru konkrétní jízdy ze seznamu na *DrivesPage*. Zde se ve vstupních

Obrázek 5.5: Výsledek implementace nových grafických prvků

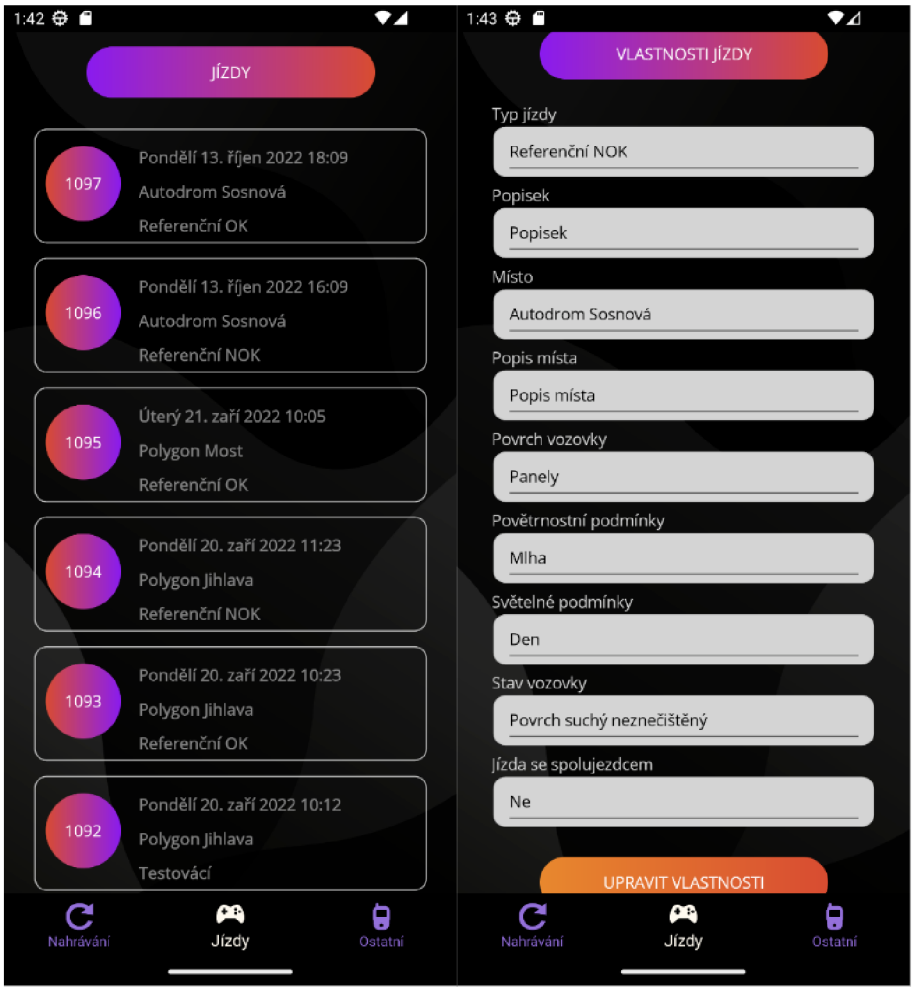
polích jednotek zobrazují původně zadané údaje o jízdě. Snímek vpravo na obrázku 5.6 ukazuje stránku po kliknutí na první cestu v seznamu na *DrivesPage*. V dolní části stránky je tlačítko, po jehož kliknutí se aplikují změny zadané uživatelem na příslušné instanci datového modelu *DriveData*.

ViewModel

Třída *DrivesViewModel* slouží jako prostředník mezi Zobrazením a datovým Modelem. Obsahuje kolekci objektů *DriveData*, která je zabalena v *ObservableCollection*. To umožňuje Zobrazení reagovat na změny v kolekci automaticky díky mechanizmu oznámení změn *INotifyPropertyChanged*.

Datové Vazby

V MVVM, zobrazení komunikuje s viewmodel prostřednictvím datových vazeb, které jsou zřízeny v zobrazení. Například, BindingContext *DrivesPage* je nastaven na instanci *DrivesViewModel*, což umožňuje vazbu mezi UI prvky a daty, které viewmodel poskytuje, konkretně se jedná o objekty *DriveData*.



Obrázek 5.6: Vzhled nových funkčních stránek

6 Závěr

Cílem této bakalářské práce bylo analyzovat problematiku tvorby mobilních a webových aplikací na platformě .NET a dále vyvinout rozšíření pro stávající prototyp telemetrického systému sestávajícího ze senzorového zařízení, mobilní a webové aplikace. Na základě rešerše byly vybrány vhodné komponenty platformy .NET, které byly následně použity při vývoji rozšíření. Výběr technologií byl omezen, protože rozšíření jsou vyvíjena pro existující aplikace, a byly zvoleny tak, aby vyvinuté moduly bylo možné snadno integrovat do již existujícího projektu.

Ve druhé části práce byly analyzovány pojmy telemetrie a telemetrický systém. Vysvětlena byla technologie kinematiky v reálném čase, která hraje důležitou roli při sběru přesných telemetrických dat. Dále byla popsána telemetrická data, která mají být následně zpracována webovou aplikací; byla rozdělena do samostatných skupin podle informací, které poskytují.

Ve třetí části byly vyvinuty vybrané algoritmy pro zpracování telemetrických dat pro jejich následnou vizualizaci pomocí webové stránky. Algoritmy byly otestovány na řadě jízd za různých podmínek. Dále byla vytvořena webová stránka pro vizualizaci dat zpracovaných algoritmy. Její funkčnost a design byly vyvinuty na základě požadavků. Webová aplikace umožňuje analyzovat jízdu v každé sekundě pomocí map a grafů zobrazujících změnu hodnot jednotlivých telemetrických údajů v průběhu celé jízdy.

V poslední části práce bylo vyvinuto nové uživatelské rozhraní pro mobilní aplikaci, která slouží jako prostředník mezi senzorovým zařízením a serverem s databází. Jeho návrh byl vypracován na základě návrhů získaných od grafika a upraven na základě konzultací s cílovými uživateli.

Všechny body zadání byly splněny. Požadavky na mobilní a webové aplikace byly rovněž plně realizovány. Jedinou výjimkou je návrh mobilní aplikace; některé funkční prvky navržené grafikem nebyly v nejnovější verzi platformy .NET MAUI realizovatelné a na základě konzultací byly implementovány jinak.

Použitá literatura

[1] O'GRADY, Brian. *What is a Framework? Why We Use Software Frameworks* [online]. [cit. 2024-05-04]. Dostupné z: https://codeinstitute.net/global/blog/what-is-a-framework/.

[2] JAIN, Ritika. *What is a Framework and Why use Frameworks in Software Development?* [online]. 2022. [cit. 2024-05-04]. Dostupné z: https://invedus.com/blog/what-is-a-framework-and-why-use-frameworks-in-software-development/.

[3] UXPIN. *Why Developers Use Frameworks?* [online]. 2023. [cit. 2024-05-04]. Dostupné z: https://www.uxpin.com/studio/blog/why-developers-use-frameworks/.

[4] AMAZON WEB SERVICES. *What is .NET?* [online]. Amazon Web Services, Inc., 2024 [cit. 2024-05-04]. Dostupné z: https://aws.amazon.com/what-is/net/.

[5] EPAM SYSTEMS. *What is .NET and what does a .NET developer do?* [online]. 2023. [cit. 2024-05-04]. Dostupné z: https://training.epam.com/en/blog/301.

[6] ROUSE, Margaret. *Framework Class Library* [online]. 2011. [cit. 2024-05-04]. Dostupné z: https://www.techopedia.com/definition/24212/framework-class-library-fcl-net.

[7] TECHTARGET. *Common Language Runtime (CLR)* [online]. [cit. 2024-05-04]. Dostupné z: https://www.techtarget.com/whatis/definition/Common-Language-Runtime-CLR.

[8] FREEMAN, Adam. *Pro ASP.NET Core 7*. 10. vyd. Manning, 2023. ISBN 978--1633437821.

[9] HUA, Chen. MVC Design Pattern. *Computer Knowledge and Technology*. 2007. Dostupné z DOI: 10.1007/978-1-4302-0129-8_5.

[10] MASOUD, Fawaz A.; HALABI, Dana H.; HALABI, Deema H. ASP.NET and JSP Frameworks in Model View Controller Implementation. In: *2006 2nd International Conference on Information Communication Technologies*. 2006, sv. 2. Dostupné z DOI: 10.1109/ICTTA.2006.1684998.

[11] JAPIKSE, Philip; GROSSNICKLAUS, Kevin; DEWEY, Ben. Introducing Entity Framework Core. *Pro C 9 with .NET 5*. 2019. Dostupné z DOI: 10.1007/978-1-4842-2478-6_1.

[12] GUTHRIE, Scott. *Microsoft to acquire Xamarin and empower more developers to build apps on any device* [online]. Microsoft Corporation, 2016 [cit. 2024-03-24]. Dostupné z: https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/.

[13] MICROSOFT LEARN. *What is Xamarin?* [online]. Microsoft Corporation, 2020 [cit. 2024-03-24]. Dostupné z: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

[14] PETZOLD, Charles. *Creating Mobile Apps with Xamarin.Forms Preview Edition 2*. Pearson Education, 2015. Developer Reference. ISBN 9780735697379. Dostupné také z: https://books.google.cz/books?id=mUMNCAAAQBAJ.

[15] MICROSOFT LEARN. *XAML overview (WPF .NET)* [online]. Microsoft Corporation, 2023 [cit. 2024-04-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/.

[16] GANAPATHY KATHIRESAN, Selva. *Xamarin Versus .NET MAUI* [online]. 2023. [cit. 2024-03-24]. Dostupné z: https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx.

[17] MICROSOFT LEARN. *Write and debug running code with Hot Reload in Visual Studio (C, Visual Basic, C++)* [online]. Microsoft Corporation, 2023 [cit. 2024-03-24]. Dostupné z: https://learn.microsoft.com/en-us/visualstudio/debugger/hot-reload.

[18] GOSSMAN, John. *Introduction to Model/View/ViewModel pattern for building WPF apps* [online]. 2005. [cit. 2024-04-29]. Dostupné z: https://shorturl.at/dirRV.

[19] ZYCH, Filip. *Getting to Grips with MVVM Architecture* [online]. 2023. [cit. 2024-04-30]. Dostupné z: https://www.netguru.com/blog/mvvm-architecture.

[20] GARCÍA GALLARDO, Estefanía. *What Is MVVM Architecture?* [online]. 2023. [cit. 2024-04-30]. Dostupné z: https://builtin.com/software-engineering-perspectives/mvvm-architecture.

[21] LINKEDIN. *How does MVVM data binding compare to other techniques in software design?* [online]. 2023. [cit. 2024-04-30]. Dostupné z: https://www.linkedin.com/advice/0/how-does-mvvm-data-binding-compare-other-techniques-c8v6e.

[22] RAMOTION. *Understanding MVVM: Model-View-ViewModel Architecture Explained* [online]. 2023. [cit. 2024-04-30]. Dostupné z: https://www.ramotion.com/blog/what-is-mvvm/.

[23] KREJCAR, Ondrej. *Modern Telemetry*. Rijeka: IntechOpen, 2011. ISBN 978--953-307-415-3. Dostupné z DOI: 10.5772/910.

[24] STACKIFY. *What Is Telemetry? How Telemetry Works, Benefits of Telemetry, Challenges, Tutorial, and More* [online]. 2024. [cit. 2024-03-25]. Dostupné z: https://stackify.com/telemetry-tutorial/.

[25] MICROSOFT LEARN. *Tables* [online]. Microsoft Corporation, 2024 [cit. 2024-03-25]. Dostupné z: https://learn.microsoft.com/en-us/sql/relational-databases/tables/tables?view=sql-server-ver16.

[26] GLOBAL GPS SYSTEMS. *RTK GPS: Understanding Real-Time Kinematic GPS Technology* [online]. [cit. 2024-05-03]. Dostupné z: https://shorturl.at/eiyX6.

[27] HEXAGON. *Real-Time Kinematic (RTK)* [online]. [cit. 2024-05-03]. Dostupné z: https://novatel.com/an-introduction-to-gnss/resolving-errors/rtk.

[28] CHAUHAN, Shailendra. *Implementation of Dependency Injection Pattern* [online]. 2024. [cit. 2024-05-01]. Dostupné z: https://www.scholarhat.com/tutorial/designpatterns/implementation-of-dependency-injection-pattern.

[29] IBM CORPORATION. *What is Ajax?* [online]. 2021. [cit. 2024-05-02]. Dostupné z: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=page-asynchronous-javascript-xml-ajax-overview.

[30] SWADIA, Shachee. *A Beginner's Guide to Creating a Map Using Leaflet.js* [online]. 2021. [cit. 2024-05-02]. Dostupné z: https://www.sitepoint.com/leaflet-create-map-beginner-guide/.

[31] W3SCHOOLS. *Chart.js* [online]. [cit. 2024-05-02]. Dostupné z: https://www.w3schools.com/ai/ai_chartjs.asp.

[32] FRAIN, Ben. *Responsive Web Design with HTML5 and CSS: Develop Future-Proof Responsive Websites Using the Latest HTML5 and CSS Techniques*. 3. vyd. Birmingham, England: Packt Publishing, 2020. ISBN 978-1-83921-156-0.

[33] MICROSOFT. *Xamarin Support Policy* [online]. Microsoft Corporation, 2024 [cit. 2024-04-20]. Dostupné z: https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin.

[34] MICROSOFT LEARN. *Upgrade from Xamarin to .NET* [online]. Microsoft Corporation, 2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/migration/.

[35] MICROSOFT LEARN. *Controls* [online]. Microsoft Corporation, 2024 [cit. 2024-04-01]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/.

[36] DOWNS, Joseph. *Drop down list design: the complete guide* [online]. 2020. [cit. 2024-04-27]. Dostupné z: https://www.justinmind.com/blog/drop-down-list-design.