

Návrh a vývoj webové aplikace pro evidenci firemních jízd

Michal Markus



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

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Michal Markus
Osobní číslo: A22386
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Kombinovaná
Téma práce: Návrh a vývoj webové aplikace pro evidenci firemních jízd
Téma práce anglicky: Design and Development of a Web Application for Corporate Vehicle Trip Records

Zásady pro vypracování

- Provedte rešerši stávajících řešení.
- Vypracujte stručný rozbor technologií, které budou použity k návrhu.
- Provedte rozbor a analýzu požadavků na zvolené řešení.
- Zpracujte aplikaci na základě výsledků analýzy.
- Věnujte pozornost škálovatelnosti, zabezpečení a uživatelské přívětivosti aplikace.

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

Seznam doporučené literatury:

1. FREEMAN, Adam, [2017]. *Pro ASP.NET Core MVC 2*. Seventh edition. London: Apress. ISBN 978-1484231494.
2. SMITH, Jon P., 2018. *Entity Framework Core in Action*. Manning. ISBN 978-1617294563.
3. PRICE, Mark J., 2020. *C# 9 and .NET 5 – modern cross-platform development: build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code*. Fifth edition. Birmingham: Packt. ISBN 978-1800568105.
4. KRUG, Steve, [2014]. *Don't make me think, revisited: a common sense approach to web usability*. Voices that matter. [Berkeley]: New Riders. ISBN 978-0-321-96551-6.
5. HOFFMAN, Andrew, 2024. *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. 2. O'Reilly Media. ISBN 9781098143893.

Vedoucí bakalářské práce:

doc. Ing. Radek Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **27. října 2024**

Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

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

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

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....

podpis autora

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a vývojem webové aplikace pro evidenci firemních jízd. Cílem bylo vytvořit systém vhodný pro malé firmy a podnikatele, kteří požadují jednoduché a mobilně dostupné řešení. Na základě rešerše stávajících softwarů byly definovány požadavky na funkčnost, škálovatelnost a bezpečnost systému. Aplikace byla navržena s využitím architektury ASP.NET Core MVC a principů doménově řízeného návrhu (DDD). Datová vrstva je realizována pomocí Entity Framework Core, uživatelské rozhraní využívá Bootstrap. Výsledná aplikace umožňuje evidenci jízd, vozidel a správu uživatelů s rozdílnými oprávněními. Zvláštní pozornost byla věnována validaci dat, ochraně proti běžným webovým útokům a připravenosti na horizontální škálování. Aplikace splňuje cíle stanovené v zadání a tvoří základ pro možné budoucí rozšíření.

Klíčová slova: ASP.NET Core, Entity Framework, evidence jízd, webová aplikace, bezpečnost, škálovatelnost

ABSTRACT

This bachelor's thesis focuses on the design and development of a web application for recording corporate vehicle trips. The goal was to create a system suitable for small businesses and entrepreneurs who require a simple and mobile-accessible solution. Based on a review of existing software, the functional, security, and scalability requirements were defined. The application was designed using ASP.NET Core MVC architecture and Domain-Driven Design (DDD) principles. The data layer is implemented with Entity Framework Core, and the user interface uses the Bootstrap framework. The resulting application supports trip logging, vehicle tracking and user management with role-based access. Special attention was given to data validation, protection against common web attacks, and readiness for horizontal scaling. The developed solution fulfills the defined objectives and provides a solid foundation for future enhancements.

Keywords: ASP.NET Core, Entity Framework, trip records, web application, security, scalability

Na tomto místě bych rád poděkoval vedoucímu práce doc. Ing. Radku Šilhavému, Ph.D. za cenné rady, odborné vedení a trpělivost během celé doby zpracování bakalářské práce. Dále děkuji své rodině za podporu a pochopení a obzvlášť bratrovi Tomáši Markusovi za pomoc s uživatelským testováním aplikace.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 REŠERŠE EXISTUJÍCÍCH SYSTÉMŮ.....	13
1.1 AUTOPLAN	13
1.1.1 Výhody	13
1.1.2 Nevýhody	14
1.2 GPS DOZOR	14
1.2.1 Výhody	14
1.2.2 Nevýhody	15
1.3 LOKÁTOR.CZ.....	15
1.3.1 Výhody	15
1.3.2 Nevýhody	16
1.4 MONEYS3 A JINÉ ÚČETNÍ SOFTWARE	16
1.5 SOUHRN	16
2 POUŽITÉ TECHNOLOGIE A NÁSTROJE	17
2.1 ASP.NET CORE.....	17
2.1.1 Architektura MVC	17
2.1.2 Razor Pages	18
2.1.3 Blazor	18
2.1.4 Validace dat.....	18
2.1.5 Ověřování	19
2.2 ENTITY FRAMEWORK CORE	20
2.3 TECHNOLOGIE FRONTEND	21
2.4 BĚHOVÉ PROSTŘEDÍ	22
3 ŠKÁLOVATELNOST	23
3.1 UKLÁDÁNÍ KRÁTKODOBÝCH STAVŮ A CACHE	23
3.2 ASYNCHRONNÍ ZPRACOVÁNÍ A NEBLOKUJÍCÍ I/O	23
3.3 OPTIMALIZACE DATABÁZOVÉHO PŘÍSTUPU	24
3.4 AUTOMATICKÉ ŠKÁLOVÁNÍ V CLOUDU A KONTEJNERY	24
4 UŽIVATELSKÁ PŘÍVĚTIVOST	25
II PRAKTICKÁ ČÁST	26
5 ANALÝZA POŽADAVKŮ	27
5.1 CÍLOVÁ SKUPINA	27
5.2 ODŮVODNĚNÍ VÝVOJE VLASTNÍHO SYSTÉMU	27
5.3 NAVRHOVANÉ VLASTNOSTI APLIKACE	27
5.3.1 Evidence jízd	27

5.3.2	Evidence vozidel	28
5.3.3	Správa uživatelů	28
6	NÁVRH APLIKACE	29
6.1	UŽIVATELSKÉ PŘÍBĚHY (USER STORIES)	29
6.2	MINIMÁLNÍ ŽIVOTASCHOPNÝ PRODUKT	38
6.3	MODEL PŘÍPADŮ UŽITÍ	39
6.3.1	Diagram případů užití	39
6.3.2	Popis případů užití	42
6.4	MODEL TŘÍD	50
6.5	DYNAMICKÁ ANALÝZA	52
6.5.1	Zahájení jízdy	53
6.5.2	Přidání bodu jízdy	53
6.5.3	Vytvoření uživatele	54
7	IMPLEMENTACE	56
7.1	ARCHITEKTURA APLIKACE	56
7.1.1	Doménová vrstva (projekt Domain)	56
7.1.2	Aplikační vrstva (projekt Application)	57
7.1.3	Prezentační vrstva (projekt Web)	58
7.1.4	Vrstva infrastruktury (projekt Infrastructure)	58
7.2	UŽIVATELSKÉ ROZHRANÍ	59
7.3	PRÁCE S DATY A KOMUNIKACE MEZI VRSTVAMI	66
8	ZABEZPEČENÍ	68
8.1	SQL INJECTION	68
8.2	CROSS-SITE SCRIPTING	68
8.3	CROSS-SITE REQUEST FORGERY (CSRF)	69
8.4	ÚTOKY NA OVĚŘOVÁNÍ A ZABEZPEČENÍ HESEL	70
	ZÁVĚR	71
	SEZNAM POUŽITÉ LITERATURY	73
	SEZNAM OBRÁZKŮ	75
	SEZNAM TABULEK	76
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	77
	SEZNAM PŘÍLOH	79

ÚVOD

Vedení knihy jízd je pro řadu podnikatelů a firem zákonnou povinností, ale zároveň i důležitým nástrojem pro sledování provozních nákladů a optimalizaci využití firemních vozidel. V praxi však řada dostupných softwarových řešení neodpovídá potřebám malých firem, protože buď vyžadují investici do specializovaného hardwaru, nebo naopak poskytují pouze omezenou funkcionalitu bez důrazu na přístupnost a pohodlí uživatele. Tato situace byla hlavním motivem pro volbu tématu této bakalářské práce.

Cílem práce je navrhnout a realizovat webovou aplikaci pro evidenci firemních jízd, která bude jednoduše dostupná i z mobilních zařízení, bude přehledná, uživatelsky přívětivá a zároveň dostatečně rozšiřitelná pro budoucí potřeby. Aplikace má vyplnit mezeru mezi složitými systémy s GPS sledováním a ruční evidencí např. v Excelu nebo účetních programech.

Pro dosažení tohoto cíle je nejprve provedena rešerše existujících řešení dostupných na českém trhu, následně jsou analyzovány požadavky na zamýšlenou aplikaci s ohledem na cílovou skupinu. Dále je zpracován návrh systému včetně modelu případů užití, návrhu tříd a sekvenčních diagramů. Aplikace je implementována pomocí frameworku ASP.NET Core MVC v kombinaci s Entity Framework Core a Bootstrap. V závěru je věnována pozornost zabezpečení a uživatelskému rozhraní.

Práce je rozdělena do teoretické a praktické části. Teoretická část shrnuje existující systémy, technologie, zásady škálovatelnosti a uživatelské přívětivosti. Praktická část se zaměřuje na vlastní návrh a implementaci aplikace včetně prezentace klíčových obrazovek a vyhodnocení dosažených cílů.

Při vypracování byly využívány nástroje umělé inteligence a to konkrétně GitHub Copilot jako doplněk vývojového prostředí Visual Studio Community 2022 k automatickému dokončování příkazů na základě kontextu. Veškerý takto dokončený kód jsem zkontroloval, upravil pro potřeby aplikace a přebírám za něj plnou zodpovědnost.

I TEORETICKÁ ČÁST

1 REŠERŠE EXISTUJÍCÍCH SYSTÉMŮ

Cílem této části je najít existující systémy s podobným účelem jako navrhovaná aplikace, porovnat jejich výhody a nevýhody a tím upřesnit vlastnosti vyvíjeného systému. Systémy do rešerše byly vybírány tak, aby pokryly co nejširší spektrum možností evidence jízd, které jsou dostupné na českém trhu a bylo tak možné porovnat co nejvíce dostupných možností od specializovaných systémů se sledováním polohy vozidel po ruční evidenci v ekonomických softwarech.

1.1 Autoplan

Jedná se o jeden z nejstarších softwarů pro evidenci firemních jízd a správu vozového parku vyvíjený společností KROB software s.r.o. [1] Celý systém se skládá ze tří hlavních aplikací, které pracují samostatně, ale mohou sdílet společná data a tím vytvořit jednotný funkční celek [1].

Pro tento dokument je nejvíce relevantní program Kniha jízd Autoplan [1], která umožňuje vyplňování knihy jízd ručně, ze seznamu obvyklých jízd nebo s využitím druhého programu Automapa [1]. Výrobce zmiňuje i možnost importu jízd z GPS zařízení, na webu výrobce ale nejsou uvedeny konkrétní požadavky, které musí toto zařízení splňovat, aby se systémem fungovalo [1].

Důležitou součástí tohoto systému je i evidence doplňování pohonných hmot, které je možné evidovat buď ručně, nebo importovat z tankovacích karet vybraných společností [1]. Údaje o tankování jsou pak využívány k výpočtu průměrné spotřeby vozidla a průměrných nákladů na zakázku [1].

Aplikace funguje pouze na operačním systému Microsoft Windows [1], což umožňuje její provoz pouze na počítači nebo notebooku. Pro sdílení aplikace mezi více uživateli je nutná instalace SQL serveru [1], což přináší další náklady. Tento systém je tedy vhodný spíše na zpětné doplňování jízd než průběžnou evidenci přímo ve vozidle. To naznačuje i možnost zpětného importu jízd.

1.1.1 Výhody

- Komplexní funkce: kniha jízd, cestovní příkazy, náhrady, evidence pohonných hmot
- Jednorázová platba, dlouhodobá podpora
- Import GPS dat z externích systémů

1.1.2 Nevýhody

- Omezená mobilní dostupnost
- Převážně ruční zadávání jíst zpětně

1.2 GPS Dozor

Tento systém se zaměřuje především na sledování polohy vozidel v reálném čase. K tomu využívá zařízení, která jsou instalována do vozidel [2], což může představovat nevýhodu, protože zprovoznění systému v některých případech vyžaduje servisní zásah [2]. Výrobce nabízí několik variant zařízení pro sledování, která se mezi sebou liší mj. četností odesílání polohy na server, způsobem instalace a schopností číst data o vozidle [2]. Výhodou je, že díky sledování pomocí instalovaného zařízení systém nevyžaduje manuální vyplňování a snižuje tak administrativní zátěž pro řidiče.

Hned na úvodní stránce je na rozdělené obrazovce seznam všech vozidel a mapa, na které je vyznačena jejich aktuální poloha [2]. Ze seznamu vozidel se uživatel může dostat do samotné knihy jízd, kde jsou zobrazeny jednotlivé cesty včetně přesné trasy na mapě [2]. Krom informací o počátečním a koncovém místě a času cesty je zobrazena i spotřeba paliva a typ jízdy. V aplikaci je uložen kompletní záznam trasy, který se dá zpětně přehrát [2]. Jednotlivé jízdy se dají filtrovat pomocí kalendáře zobrazeného v horní části stránky. Systém umožňuje také evidenci tankování, a to buď ručně, nebo importem dat z tankovacích karet [2].

Součástí aplikace jsou také exporty jízd ve formátu, který uznává finanční správa [2]. Výhodou tohoto softwaru je dostupnost na většině zařízení. Hlavní rozhraní je webová aplikace, díky čemuž není závislá na operačním systému počítače. Pro mobilní telefony je pak k dispozici samostatná aplikace [2].

1.2.1 Výhody

- Automatický záznam jízd
- Přesné trasy a detaily jízd
- Sledování polohy v reálném čase
- Dostupnost mobilní aplikace
- Možnost exportu pro finanční správu

1.2.2 Nevýhody

- Měsíční poplatky za službu, a tedy dlouhodobé náklady
- Nutnost instalace sledovacího zařízení do vozidla

1.3 Lokátory.cz

Jde o webovou aplikaci, která se podobně jako GPS Dozor soustředí na sledování polohy vozidel, nákladu nebo osob v reálném čase a ukládání záznamů o jejich pohybu [3]. Ke sledování polohy se používá zařízení FleetLock Profi, která je vestavěná přímo do vozidla a vyžaduje instalaci servisním technikem [3]. Z těchto záznamů pak tvoří elektronickou knihu jízd. Stejně jako u předchozího systému tedy není nutné manuální vyplňování. Navíc tento systém disponuje rozšířenými možnostmi vyhledávání a statistikami. Například je možné zobrazit nejčastěji navštěvovaná místa, nebo nastavit hlídání opuštění definované zóny nebo nedodržení naplánované trasy [3].

Na úvodní obrazovce je seznam sledovaných vozidel s možností řazení a vyhledávání a také jejich poloha na mapě [3]. V horní části se pak nachází nabídka funkcí. Samotná kniha jízd pak umožňuje vybrat konkrétní vozidlo a rozsah datumů v kalendáři a na základě toho zobrazit jízdy za zvolené období [3]. V tabulce se zobrazuje datum a typ jízdy, čas příjezdu a odjezdu, ujetá vzdálenost, spotřebované palivo, čas jízdy a stání, průměrná a maximální rychlost a počáteční a cílová adresa. Jednotlivé jízdy lze zpětně upravit nebo vytvořit sestavu pro tisk nebo export [3].

Aplikace také umožňuje evidenci tankování včetně importu dat z tankovacích karet, díky čemuž umí vypočítat náklady na každou jízdu [3]. Dále je možné jízdy odlišit nejen na soukromé a služební, ale také např. na jízdu s majáky nebo s nákladem. Možná je i identifikace řidiče před jízdou pomocí čipové karty a následně kontrola jeho stylu jízdy. Výhodou je také hlídání termínů servisních prohlídek [3].

1.3.1 Výhody

- Komplexní automatická evidence
- Evidence spotřeby, tankování PHM a servisů, statistiky jízd
- Podpora více uživatelů s různými úrovněmi přístupu

1.3.2 Nevýhody

- Nutnost instalace záznamového zařízení
- Nedostupnost trvalé licence
- Pro malou firmu zbytečně moc funkcí

1.4 MoneyS3 a jiné účetní softwary

Jednoduchou evidenci jízd umožňují i některé účetní programy. Např. v MoneyS3 je tato funkce dostupná jako jeden z modulů [4]. V programu se ukládají jednotlivá vozidla, ke kterým jsou přiřazeni řidiči. U vozidel jsou evidovány základní vlastnosti, jako SPZ, typ, palivo, průměrná spotřeba PHM aj. [4] V samotné knize jízd se pak zvolí řidič, vozidlo, účel jízdy, datum, čas a místo odjezdu a příjezdu a stav tachometru před a po jízdě [4]. Podobně jako u programu Autoplan se zde jedná o zpětné doplňování knihy jízd.

1.5 Souhrn

Systémy pro evidenci firemních jízd je možné rozdělit podle způsobu vyplňování dat. Automatické systémy vyplňují data o jízdách pomocí technického zařízení instalovaného ve vozidle, což většinou znamená nějakou vstupní investici. Díky tomu ale krom základních údajů o jízdě poskytují i detailnější informace o průběhu jízdy, jako je přesná trasa a průměrná a maximální rychlost. Výhodou mohou být také dostupné informace o aktuální poloze vozidla a snížená administrativní zátěž řidičů. Tyto systémy najdou uplatnění především ve větších firmách, kde by náročná administrativa manuálního vyplňování převážila cenu systému a instalaci zařízení pro sledování.

Manuální systémy naopak najdou využití u menších firem, kde jde primárně o splnění zákonné povinnosti vedení knihy jízd a není důležité evidovat přesné informace o průběhu jízd. Z provedení rešerše také vyplývá, že důležitou součástí všech systémů je i evidence tankování pohonných hmot [1; 2; 3]. Často se také objevuje hlídání termínů servisních prohlídek, technické kontroly a případně i školení řidičů.

2 POUŽITÉ TECHNOLOGIE A NÁSTROJE

2.1 ASP.NET Core

Základem aplikace bude vývojová platforma webu od společnosti Microsoft ASP.NET Core. Tato platforma pod sebou sdružuje aplikační frameworky MVC, Razor Pages a Blazor, podpůrné frameworky Identity a Entity Framework a funkce samotné platformy, které mají za úkol např. zpracování http požadavků a vykreslování Razor Pages [5].

2.1.1 Architektura MVC

Tato architektura rozděluje aplikaci do 3 částí, přičemž každá část má vlastní oblast zodpovědnosti. Model obsahuje objekty, které reprezentují data, se kterými aplikace pracuje a metody, které tato data upravují podle aplikační logiky [5]. Často se používá tzv. ViewModel, který obsahuje pouze data specifická pro daný View a metody pro jejich validaci [5].

View slouží jako uživatelské rozhraní. Jeho účelem je zobrazovat data, která přebírá z modelu, a naopak předávat do modelu data, která zadá uživatel [5]. V architektuře MVC se používá tzv. Razor View, což je textový soubor s příponou cshtml, jehož základem jsou elementy z HTML, které definují rozložení stránky. Do tohoto HTML kódu se vkládá kód C# uvozený zavináčem, který slouží k doplnění dat z modelu na stránku nebo také k podmíněnému zobrazení či opakování stejného vzoru, např. řádků tabulky, pomocí cyklu foreach. Tento kód se spouští na straně serveru a převádí se na statické HTML, které je pak předáno webovému prohlížeči [5].

Centrálním prvkem aplikace je pak kontrolér. Každý kontrolér v sobě definuje metody s návratovým typem IActionResult, kterým se říká akce [5]. Tyto metody přijímají data z http požadavků a ty předávají do modelu. Model pak používají při volání metod aplikační logiky tak, aby získali požadovaný výsledek. Tento pak předávají zpět uživateli ve formě vygenerovaného View. Kontroléry mohou vracet také přesměrování, čímž řídí průchod uživatele systémem [5].

Směrování požadavků probíhá na základě tzv. conventional routingu, který páruje části URL k názvům tříd kontrolérů a jejich akcí [5]. Přesná struktura URL je nastavena v souboru Program.cs v metodě MapControllerRoute(). Zde je definováno, která část URL určuje kontrolér a která akci. Třída kontroléru musí mít název, který bude použit v URL

následovaný slovem „Controller“. Akce tohoto kontroléru jsou pak pojmenovány tak, jako v URL [5].

2.1.2 Razor Pages

Razor pages představují jednodušší alternativu k MVC architektuře. Směrování požadavku vychází z umístění stránky v souborovém systému, čímž se podobá statickým html stránkám nebo jednoduchým PHP aplikacím. [5]. Kontrolér je tedy úplně vynechán a místo něj je spuštěn přímo soubor Razor View, který je ve struktuře projektu umístěn ve složce Pages v podsložkách podle cesty z URL. Toto zobrazení na rozdíl od architektury MVC obsahuje i aplikační logiku [5].

Razor pages najdou využití v jednodušších scénářích, kde se pro zpracování požadavku používá jedna akční metoda a jeden view. V této situaci je zápis rychlejší než v MVC, protože celá funkcionality může být sdružena do jednoho souboru stránky. Je ale nutné si dávat pozor na rozšiřování funkcionality, protože by se kód mohl stát nepřehledným.

2.1.3 Blazor

Blazor je moderní komponentový framework pro vývoj webových aplikací, který je součástí platformy ASP.NET Core. Na rozdíl od Razor Pages nebo MVC umožňuje vytvářet interaktivní uživatelské rozhraní výhradně v jazyce C# bez použití JavaScriptu [6]. K vykreslování rozhraní využívá Blazor komponenty, což jsou malé znovupoužitelné části UI, které obsahují jak značkovací jazyk, tak aplikační logiku [6].

Existují dvě hlavní varianty Blazoru, kterými jsou Blazor Server a Blazor WebAssembly [6]. Blazor Server vykonává veškerou logiku na serveru a aktualizace UI posílá do klienta pomocí technologie SignalR. WebAssembly varianta naopak umožňuje běh celé aplikace přímo v prohlížeči uživatele, a to bez potřeby serverové komunikace během interakcí [6]. Blazor nachází využití jak v rozsáhlých podnikových aplikacích, tak v jednodušších klientských rozhraních, kde je cílem sdílení kódu mezi frontendem a backendem [6].

2.1.4 Validace dat

Webová aplikace musí počítat s tím, že uživatelé mohou zadat neplatná či nekompletní data. ASP.NET Core proto obsahuje mechanismus pro validaci dat na úrovni modelů. Nejčastěji se využívají tzv. Data Annotations, což jsou předdefinované atributy, kterými se přímo

u vlastností tříd ViewModelu deklarují validační pravidla [7]. Typickým příkladem je atribut [Required], který označuje povinné pole formuláře, [StringLength] omezující délku textu, [Range] definující číselný rozsah apod. V attributech je možné také nastavit chybovou zprávu, která bude zobrazena při zadání nevalidních dat [7]. Taková validace je výhodná, protože pravidla jsou součástí definice datového modelu, ke kterému se vztahují a framework je při zpracování formulářů automaticky aplikuje.

Pokud uživatel zadá data, která poruší zadaná omezení, framework to po odeslání formuláře rozpozná a označí stav modelu jako neplatný. Kontrolér pak může tento stav detekovat pomocí atributu ModelState.IsValid a reagovat např. znovu zobrazením formuláře s chybovými hláškami [7].

Kromě serverové validace podporuje ASP.NET Core i validaci na straně klienta, která je realizována JavaScript knihovnou jQuery Validation [6]. Pokud jsou skripty z této knihovny vloženy do zobrazení a vstupní pole formuláře jsou definovány pomocí ASP.NET Tag Helperů, generují se do HTML atributy, které umožňují prohlížeči provést validaci ještě před odesláním formuláře na server [6]. Klientská validace zlepšuje uživatelský komfort, protože ihned upozorní na chybný vstup a uživatel tak nemusí čekat na zpracování dat serverem. Kontrola na straně serveru je ale stále nezbytná, protože skripty pro klientskou validaci mohou být ze stránky klientem odebrány nebo upraveny, což by umožnilo odeslat nevalidní data [8].

ASP.NET Core umožňuje i vytvoření vlastních validačních atributů nebo implementaci rozhraní IValidatableObject ve ViewModelu pro složitější scénáře validace, kdy si nevystačíme s předdefinovanými atributy [5].

2.1.5 Ověřování

Ověřování uživatelů (autentizace) a řízení jejich oprávnění (autorizace) je v ASP.NET Core dostupné v rámci integrovaného systému ASP.NET Core Identity. Ten zajišťuje správu uživatelských účtů, hesel, rolí a přihlášení a umožňuje určit, ke kterým akcím má uživatel přístup na základě jeho role nebo přiřazených oprávnění [7].

Framework poskytuje metody pro vytvoření uživatelů a jejich údaje pak ukládá do nastaveného úložiště, typicky do databáze pomocí Entity Framework Core [7]. Při přihlášení pak ověří přihlašovací údaje a v případě shody s uživatelským účtem vytvoří autentizační cookie a odešle ji prohlížeči. Prohlížeč pak tuto cookie připojuje ke všem dalším požadavkům. Díky této cookies aplikace pozná, že je uživatel přihlášený a při každém

požadavku vytvoří objekt `HttpContext.User`, který obsahuje informace o tom, že je uživatel přihlášený, ale také jeho přiřazené role [5]. Do tohoto objektu je možné přidávat i vlastní data pomocí tzv. claimů. Claim je vlastnost uživatele ve formátu klíč-hodnota a ukládá se do databáze k danému uživateli [7]. Při přihlášení jsou claimy automaticky vloženy do autentizační cookie, a díky tomu jsou dostupné během celého přihlášení bez nutnosti dalšího dotazování databáze. Claimy je možné použít např. k zobrazování specifických dat uživateli [7].

Přístupová práva se určují pomocí atributu `[Authorize]`, který se přidává buď k celému controlleru, nebo k jednotlivým akcím [5]. Samotný atribut říká, že akci mohou používat jen přihlášení uživatelé. Oprávnění je pak možné upřesnit parametrem u atributu `Authorize`, který umožní přístup buď na základě role, kterou má uživatel přiřazenou, nebo na základě předem definovaných pravidel, tzv. Policy, které mohou například vyžadovat, aby měl uživatel přiřazené claimy s nějakou hodnotou, aby mu byl udělen přístup [5]. Pokud požadavek na akci označenou atributem `Auhorize` nemá platnou autorizační cookies, vrátí server odpověď s http kódem 401 – Unauthorized, nebo uživatele přesměruje na nastavené stránku [5].

2.2 Entity Framework Core

Pro přístup k databázi je použit framework od společnosti Microsoft Entity Framework Core. Tento framework poskytuje vrstvu mezi aplikací a databází, která umožňuje vývojáři pracovat s databází jako s objekty C# kódu na místo ručního psaní SQL dotazů [9]. Díky tomu je práce s databází intuitivní a zároveň méně náchylná k chybám, protože vývojář pracuje přímo s typově bezpečnými třídami namísto textových SQL příkazů [10].

Jednou z klíčových výhod EF Core je možnost použití přístupu Code First. V tomto režimu vývojář nejprve definuje doménový model jako C# třídy s požadovanými vlastnostmi a relacemi mezi entitami [10]. Relace jsou definovány pomocí tzv. navigačních vlastností, které pak EF Core přeloží na cizí klíče v databázi. Navigační vlastnost vznikne, když entita obsahuje jako vlastnost jinou entitu [9]. Z takto definovaný tříd se poté pomocí EF Core nástrojů vygeneruje migraci. Migrace je skript, který vytvoří odpovídající schéma v databázi [9]. Tento přístup umožňuje vývojářům plně kontrolovat strukturu databáze z prostředí C#, snadno provádět změny a verzovat vývoj databáze v čase. Alternativně je možné využít i přístup Database First, ve kterém se naopak generují třídy z existující struktury databáze [9].

K datům se pak přistupuje přes instanci třídy DbContext, která by měla existovat po dobu jedné operace, např. jednoho požadavku na webovou aplikaci [10]. DbContext obsahuje reprezentaci jednotlivých tabulek databáze ve vlastnostech DbSet, se kterými se dá pracovat pomocí tzv. LINQ dotazů. Ty umožňují např. filtrovat nebo seřazovat data a vybírat z nich jen potřebná pole [10]. LINQ dotazy jsou následně pomocí EF Core přeloženy na optimalizované SQL dotazy a vykonány na databázi, ze které se vrátí objekt doménové entity. S tímto objektem se pak v rámci požadavku dále pracuje a jeho data se dají upravovat podle potřeby. Entity framework sleduje změny těchto objektů a následně je schopen uložit tyto změny do databáze [10]. Aby toto sledování fungovalo, je důležité dodržovat životní cyklus DbContextu [9; 10].

2.3 Technologie frontend

Pro realizaci uživatelského rozhraní webové aplikace byly využity standardní frontendové webové technologie HTML5, CSS3 a JavaScript. HTML5 slouží ke strukturování obsahu stránky. Definuje elementy jako nadpisy, odstavce, formuláře, tlačítka apod. CSS3 určuje vizuální styl a rozvržení těchto elementů a JavaScript zajišťuje interaktivitu na straně klienta, umožňuje reagovat na události, jako je kliknutí na tlačítko nebo odeslání formuláře a dynamicky měnit obsah stránky bez nutnosti opětovného načtení ze serveru [11].

Pro vzhled a responzivitu aplikace byl použit CSS framework Bootstrap ve verzi 5.3.6. Bootstrap poskytuje sadu předdefinovaných stylů a komponent v podobě tříd, které jsou aplikovány na html elementy [11]. Bootstrap počítá s rozložením obsahu primárně pro malé displeje, který se pak přizpůsobuje pro větší displeje. To je ve frameworku realizováno pomocí tzv. breakpointů, které slouží k rozdělení velikostí displejů do skupin [11]. Vybrané třídy, které určují rozložení prvků pak zpravidla existují v několika variantách pro různé breakpointy a díky tomu je možné aplikovat pro každou skupinu velikostí displeje jiné rozložení elementů [11].

Kromě responzivního rozložení prvků na stránce obsahuje bootstrap také různé hotové komponenty, jako navigační lišty, modální okna, tlačítka, formulářové prvky apod., které stačí vložit do stránky a případně upravit [11]. Díky Bootstrapu bylo dosaženo konzistentního a esteticky jednotného vzhledu aplikace bez nutnosti psát rozsáhlé CSS.

2.4 Běhové prostředí

Aplikace je provozována na platformě .NET ve verzi 8, což je multiplatformní běhové prostředí od společnosti Microsoft pro spouštění aplikací vytvořených v .NET, včetně ASP.NET Core. Tato platforma umožňuje aplikaci běžet na různých operačních systémech se stejným výsledkem za předpokladu, že je na cílovém systému nainstalován odpovídající .NET runtime [6]. To dává velkou flexibilitu při volbě serveru pro hostování aplikace. Běhové prostředí se stará o správu operační paměti, kompilaci kódu do strojových instrukcí za běhu a další nízko úroňové aspekty provozu aplikace [6].

ASP.NET Core používá vestavěný webový server Kestrel, který zpracovává příchozí HTTP požadavky a předává je aplikaci, a následně odesílá HTTP odpovědi klientům [7]. Na rozdíl od staršího ASP.NET založeného na platformě .NET Framework, který byl vázaný na internetovou informační službu v systému Windows [12], je Kestrel plně samostatný a funguje napříč platformami. Díky tomu lze aplikaci spustit jako samostatný proces na libovolném serveru a Kestrel zajistí naslouchání na zvoleném TCP portu pro HTTP komunikaci [7; 5].

V produkčním prostředí se Kestrel často nasazuje za reverzní proxy server, který přijímá HTTP požadavky z internetu, zajišťuje šifrování pomocí TLS, load balancing mezi více instancemi a případně cache a následně předává požadavky aplikaci běžící na Kestrelu. Tento design umožňuje aplikaci škálovat na více replik a zvyšuje bezpečnost a stabilitu [7]. Samotný Kestrel je ale dostatečně výkonný, aby obsloužil i velký provoz, a v jednodušších scénářích může běžet i bez proxy [7].

Běhové prostředí umožňuje také rozlišení režimů běhu aplikace prostředí tzv. environments. Ty se nastavují pomocí proměnné prostředí ASPNETCORE_ENVIRONMENT a díky nim je možné konfigurovat aplikaci podle aktuálních potřeb [7]. Například v prostředí Development jsou povoleny detailní chybové zprávy a dynamické načítání úprav za běhu aplikace, zatímco v prostředí Production jsou zapnuty optimalizace výkonu a potlačeny interní informace o chybách [7]. Podle běhového prostředí je také možné načítat různou konfiguraci např. připojení k databázi pro ladění a pro ostrý provoz [7].

3 ŠKÁLOVATELNOST

Škálovatelnost označuje schopnost systému přizpůsobit se rostoucí zátěži, typicky počtu uživatelů nebo objemu dat, přidáním dalších prostředků. Rozlišujeme dva základní způsoby škálování, a to vertikální a horizontální [13].

Vertikální škálování znamená navyšování kapacity jednoho serveru, např. přidáním výkonnějšího hardware, aby tentýž server zvládl obsloužit více požadavků. Naproti tomu horizontální škálování spočívá v nasazení více paralelně běžících instancí aplikace na více serverů, mezi nimiž je zátěž rozkládána pomocí load balanceru, který příchozí požadavky rozděluje mezi dostupné instance aplikace [13]. Horizontální přístup tak umožňuje prakticky neomezený růst výkonu přidáváním dalších uzlů, zatímco vertikální škálování je limitováno fyzickými možnostmi jednoho stroje. V praxi se většinou kombinují oba přístupy [13]. Aby aplikace umožňovala horizontální škálování, musí splňovat některé předpoklady rozebrané dále.

3.1 Ukládání krátkodobých stavů a cache

Pro efektivní horizontální škálování je zásadní navrhovat aplikaci jako bezstavovou, tedy tak, aby aplikace neukládala krátkodobý uživatelský stav lokálně na serverech, ale využívala distribuované úložiště [13]. Použití distribuované mezipaměti, např. Redis, umožňuje sdílení stavu mezi všemi instancemi aplikace. Tento přístup garantuje konzistentní uživatelskou zkušenost i při dynamických změnách počtu běžících instancí [13].

Pro zvýšení výkonu a zajištění rychlé odezvy je doporučeno implementovat víceúrovňové cachování. ASP.NET Core podporuje kromě distribuované cache také in-memory cache pro jednotlivé instance aplikace, což umožňuje rychlé načítání často používaných dat přímo z operační paměti [14]. Pro statický obsah, jako jsou JavaScriptové soubory, obrázky a CSS styly, je vhodné využít tzv. Content Delivery Network (CDN), což je síť serverů geograficky rozložená po světě, aby si uživatelé statické zdroje stahovali z jim nejbližšího serveru, což snižuje zátěž sítě a backendu [13].

3.2 Asynchronní zpracování a neblokuující I/O

Jedním z hlavních faktorů umožňujících škálování je plná podpora asynchronního zpracování v ASP.NET Core. Asynchronní přístup znamená, že vlákna serveru nejsou blokována během čekání na odpovědi databáze nebo externích služeb. Server tak může efektivněji obsloužit vyšší počet souběžných požadavků [14]. Doporučuje se důsledně

implementovat asynchronní databázové operace, např. využívat metod jako `ToListAsync()` v EF Core, a delegovat dlouhodobé či výpočetně náročné operace na pozadí mimo vlákno, které se stará o zpracování http požadavků [13].

3.3 Optimalizace databázového přístupu

Databázová vrstva často tvoří úzké hrdlo výkonnosti aplikace. Použití EF Core zjednodušuje optimalizovaný a efektivní přístup k databázi, například využitím connection pooling, který minimalizuje nároky na otevírání a zavírání databázových spojení [10]. Dále umožňuje asynchronní databázové operace, což pomáhá efektivněji využít prostředky serveru. Framework automaticky generuje optimalizované SQL dotazy a podporuje jejich kompilaci a cachování výsledků. Pro další optimalizaci je vhodné zajistit správné indexování databázových tabulek a případně využít oddělení čtecích a zapisovacích operací na repliky databáze [10].

3.4 Automatické škálování v cloudu a kontejnery

Cloudová prostředí, jako je Azure nebo AWS, umožňují automatické horizontální škálování aplikací dle aktuálního zatížení. ASP.NET Core je pro tato prostředí dobře připravený díky svému bez stavovému návrhu a multiplatformnosti [13]. Nasazení aplikace v Docker kontejnerech nebo cloudových službách je přímočaré a umožňuje flexibilní přidávání či odebírání instancí aplikace dle potřeby. Pravidla automatického škálování mohou být nastavena na základě různých metrik, jako je například zatížení CPU nebo počet aktivních požadavků [13].

4 UŽIVATELSKÁ PŘÍVĚTIVOST

Cílem návrhu přívětivého rozhraní je, aby se nový uživatel rychle zorientoval, tedy aby bylo rozhraní intuitivní a dobře zapamatovatelné, aby uživatel po čase nemusel znovu studovat ovládací prvky [15]. Zároveň je třeba, aby zkušený uživatel mohl se aplikací pracovat rychle a aby systém minimalizoval uživatelské chyby či je snadno napravoval. Takové chování pak vede k celkové spokojenosti uživatelů [15].

Základem pro uživatelsky přívětivou aplikaci je intuitivní UI. Prvky by v něm měli být rozloženy tak, jak uživatel očekává na základě předchozí zkušenosti [16]. To zahrnuje například umístění hlavního menu nahoře, nebo tlačítka pro uložení formuláře dole pod vstupy [16]. Aplikace by také měla dodržovat známé ikony, tj. tužka pro editaci, koš pro smazání záznamu atp. Velmi důležité je také dodržovat konzistentní design napříč celou aplikací a dobře obsah uspořádávat i na zařízeních s malým displejem [16].

Dalším klíčovým prvkem je předávání informací o stavu systému uživateli. Při provádění delší operace by mělo být zobrazeno načítání a pokud je to možné včetně procentuálního postupu operace, aby uživatel věděl, že systém nezamrzl a pracuje na pozadí [15]. Po kliknutí na akční tlačítko je dobré zobrazit zprávu, že akce byla provedena nebo chybovou zprávu. S tím souvisí i validace formulářů, kde by se měli chybové zprávy zobrazovat nejlépe ihned po zadání neplatné hodnoty, aby uživatel měl možnost chybu napravit a nemusel např. vyplňovat ostatní údaje zbytečně [16]. Když už je formulář odeslán s chybou, je velmi důležité ve vráceném formuláři zachovat vyplněné hodnoty [16].

Podstatný je také obsah chybové zprávy. Ta by měla být vždy formulována tak, aby uživateli jasně řekla, co se pokazilo, a jak to může napravit, a to co nejpřívětivějším způsobem [15]. Měli bychom se řídit zásadou nikdy nevinit uživatele a chybové zprávy formulovat spíše pozitivně, např.: „Vyplňte prosím uživatelské jméno“ místo „Nevyplnil jste uživatelské jméno!“. Pro chyby na straně aplikace by měli být připraveny přívětivé chybové stránky, které uživateli řeknou, jak může problém řešit, například zobrazí kontakt na technickou podporu [16].

II PRAKTICKÁ ČÁST

5 ANALÝZA POŽADAVKŮ

5.1 Cílová skupina

Tento software se bude soustředit primárně na podnikatele a menší firmy, kde není potřeba detailního sledování polohy vozidel a jde především o evidenci jízd pro úřední účely nebo i jakožto podklad pro fakturaci nákladů na cestu klientům.

5.2 Odůvodnění vývoje vlastního systému

Hlavním důvodem pro vývoj vlastní aplikace pro evidenci firemních jízd je vyplnění mezery mezi komplexními systémy s podporou automatického sledování a jednoduššími nástroji pro zpětné vyplňování, ať už se jedná o rozšiřující moduly účetních systému, tabulku v excelu nebo papírovou knihu jízd ve vozidle.

Automatické systémy často přináší vyšší pořizovací náklady kvůli nutnosti instalace zařízení pro sledování do vozidla. Navíc často obsahují pokročilé funkce, které malá firma nebo podnikatel nejspíš nevyužije. S tím pak souvisí i průběžné náklady na systém, které mohou být pro zmíněnou cílovou skupinu neodůvodněné a příliš vysoké.

Jednoduché systémy s ruční evidencí naopak mívají omezenou dostupnost v terénu, jak vyplývá z provedené rešerše, a tak je pravděpodobnější, že v evidenci vzniknou nepřesnosti kvůli zpětnému doplňování. Tento problém by měl odstranit vývoj webové aplikace, která bude jednoduše dostupná v terénu na mobilních zařízeních a bude klást důraz na jednoduchost a dílčí automatizaci, aby administrativní zátěž pro řidiče zůstala co nejnížší.

5.3 Navrhované vlastnosti aplikace

5.3.1 Evidence jízd

Hlavním cílem je splnění zákonné povinnosti pro evidenci jízd, takže systém musí umožnit v první řadě odlišit účel jízdy, tedy jestli se jedná o soukromou nebo služební jízdu. U soukromých jízd stačí evidovat pouze datum a stav tachometru před a po jízdě. U služebních jízd je navíc nutné zadat výchozí a cílové místo a nepovinně také čas odjezdu a příjezdu. Všechny záznamy navíc musí obsahovat informace o řidiči, který jízdu provedl.

S ohledem na předpokládané požadavky cílové skupiny by měla aplikace umožňovat manuální vyplňování jízd pomocí jednoduchého formuláře, který bude responzivní a dobře použitelný i na mobilním telefonu. V rámci úspory času se nabízí například využití aktuální

polohy zařízení jako koncového bodu jízdy, případně možnost vybrat začátek a konec jízdy z mapy. Vhodné by bylo také automatické dopočítání ujetých kilometrů na základě počátečního a koncového bodu trasy. Řidič bude automaticky doplněn podle aktuálně přihlášeného uživatele. Díky této automatizaci by pak řidič musel doplnit pouze počáteční a koncový stav tachometru.

5.3.2 Evidence vozidel

Samotná kniha jízd bude vázaná na konkrétní vozidlo, tudíž další důležitou součástí aplikace bude evidence vozidel. Ta musí obsahovat jednoznačný identifikátor vozidla, kterým bude SPZ, případně VIN kód. Dále se nabízí možnost evidovat důležité termíny, jako jsou servisní prohlídky, ať už na základě ujetých kilometrů načtených z knihy jízd nebo nastaveného časového intervalu a termíny povinných technických kontrol. Pro zjednodušení vyplňování jízd by také byla užitečná možnost přiřadit vozidlo konkrétnímu řidiči, který by pak mohl mít toto vozidlo předvyplněné při vytváření záznamu o jízdě.

Mimo samotné jízdy by bylo vhodné evidovat také doplňování pohonných hmot. Zde se nabízí možnost uložit do systému fotografii účtenky za palivo a vyplnit načerpané množství a celkovou částku. Výhodou by bylo automatické načtení těchto hodnot z účtenky. Je rovněž vhodné zvážit možnost importu dat z tankovacích karet, které mohou být využívány i v malých firmách nebo podnikatelích.

5.3.3 Správa uživatelů

Jelikož aplikace cílí krom jednotlivců také na malé firmy, bude potřebná podpora více uživatelů s nastavitelnými oprávněními. V rámci zachování jednoduchosti budou oprávnění přidělena na základě rolí. Tyto role budou primárně tři. První z nich bude administrátor, který bude mít přístup do všech částí aplikace a bude se starat o vytváření ostatních uživatelů a přiřazování vozidel. Řidič bude mít přístup pouze k vyplňování jízd, eventuálně k informacím o jemu přiřazených vozidlech. Poslední rolí pak bude účetní, která bude mít přístup k zobrazení evidence jízd pro jednotlivá vozidla a k jejich exportu pro další zpracování.

Toto rozdělení rolí podporuje provoz aplikace v menší firmě, kde už je nastavení přístupu a s tím spojené zodpovědnosti důležité. Zároveň bude ale zachována možnost aplikaci pohodlně provozovat pro samostatného podnikatele s jedním vozidlem, který bude jako administrátor mít přístup k evidenci jízd i přehledům a exportům dat.

6 NÁVRH APLIKACE

Tato kapitola se věnuje návrhu systému podle analýzy z teoretické části. Z popisu aplikace budou vytvořeny uživatelské příběhy, následně budou identifikovány klíčové součásti a vytvořen model případů užití. Dále pak model klíčových tříd a sekvenční diagramy.

6.1 Uživatelské příběhy (User Stories)

První část návrhu představuje jednoduchý způsob, jak vyjádřit požadavky na funkcionalitu systému z pohledu koncového uživatele. Slouží jako základ pro návrh i plánování vývoje a pomáhá udržet zaměření na potřeby uživatele. Každý příběh popisuje, kdo bude funkci využívat, co chce provést a proč považuje tuto funkcionalitu za potřebnou [17].

US01: Jako řidič chci ručně zadat novou jízdu a vyplnit povinné údaje, tj. datum, typ jízdy (soukromá nebo služební), stav tachometru před a po jízdě, u služebních jízd navíc počáteční a koncové místo a účel cesty, abych splnil zákonnou povinnost evidence.

Akceptační kritéria:

- Po kliknutí na tlačítko přidat jízdu se zobrazí formulář vytvoření jízdy
- Řidič může zadat všechny povinné údaje
- Formulář zobrazí jen vyžadovaná pole na základě zvoleného typu jízdy
- Formulář validuje vstupy a upozorňuje na chyby
- Po uložení se jízda zobrazí v přehledu

Možné úkoly:

- Vytvoření databázového modelu pro jízdu
- Implementace frontend a backend formuláře pro zadávání jízd
- Implementace validace vstupů na FE i BE

Časový odhad: 20 MH

US02: Jako řidič chci použít aktuální polohu zařízení jako počáteční a koncový bod jízdy, abych ušetřil čas při vyplňování.

Akceptační kritéria:

- Pole pro zadání cílové i výchozí adresy obsahuje tlačítko pro načtení aktuální polohy
- Po stisknutí tlačítka se do pole vyplní aktuální poloha zařízení

Možné úkoly:

- Implementace zjištění polohy zařízení
- Rozšíření polí pro zadání adresy o tlačítko aktuální polohy

Časový odhad: 8 MH

US03: Jako řidič chci zvolit body jízdy z mapy, abych přesně zaznamenal trasu.

Akceptační kritéria:

- Ve formuláři je možnost otevření mapy, kde lze vybrat počáteční, cílový i průjezdní body trasy
- Vybrané body se uloží k jízdě v zadaném pořadí

Možné úkoly:

- Implementovat zobrazení mapy
- Implementovat načítání adres z mapy
- Rozšířit databázový model jízdy o průjezdní body

Časový odhad: 24 MH

US04: Jako řidič chci, aby systém automaticky vypočítal ujetou vzdálenost podle mapy a předvyplnil ji do formuláře, abych ušetřil čas.

Akceptační kritéria:

- Systém najde trasu mezi zadanými body a vypočítá její délku
- Konečný stav tachometru se automaticky vyplní podle počátečního stavu a vzdálenosti trasy

Možné úkoly:

- Implementovat plánování trasy
- Přidat do formuláře vytvoření jízdy automatické vyplnění koncového stavu tachometru

Časový odhad: 8 MH

US05: Jako řidič chci vidět přehled jízd v tomto měsíci s možností úprav jízd, abych mohl zkontrolovat a případně opravit evidenci

Akceptační kritéria:

- Pro řidiče je dostupná tabulka všech zadaných jízd v tomto měsíci, která obsahuje základní údaje o jízdě
- Z tabulky lze otevřít detail jízdy, kde jsou zobrazeny všechny údaje o jízdě
- V detailu jízdy je možné upravovat údaje o jízdě

Možné úkoly:

- Implementovat front end tabulku přehledu jízd a odpovídající back end
- Vytvořit formulář pro úpravu nebo smazání jízdy

Časový odhad: 12 MH

US06: Jako řidič chci mít přístup jen k jízdám, které jsem vytvořil, aby nebylo zobrazení nepřehledné kvůli informacím, které jsou pro mě nepodstatné

Akceptační kritéria:

- V přehledu jízd se zobrazují jen jízdy přihlášeného řidiče
- Místo jízd ostatních řidičů se zobrazuje jen souhrnný řádek se stavu tachometru pro kontext

Možné úkoly:

- Implementovat filtrování jízd podle řidiče

Časový odhad: 4 MH

US07: Jako řidič chci evidovat stav tachometru při tankování, množství a cenu paliva natankovaného do vozidla, datum a čas tankování a nahrát fotografii účtenky, aby měl administrátor přehled o spotřebě a nákladech

Akceptační kritéria:

- Úvodní stránka obsahuje tlačítko, které zobrazí evidenci tankování
- Do formuláře tankování je možné vyplnit všechny požadované údaje a nahrát fotografii účtenky
- Systém validuje vstupy a zobrazuje relevantní chybové hlášky

Možné úkoly:

- Navrhnout databázový model pro tankování PHM
- Vytvořit formuláře pro vytvoření záznamu o tankování
- Implementovat ukládání fotografií

Časový odhad: 16 MH

US08: Jako řidič chci, aby aplikace načetla údaje z účtenky automaticky, abych ušetřil čas.

Akceptační kritéria:

- Aplikace po nahrání účtenky do formuláře tankování předvyplní množství a cenu paliva

Možné úkoly:

- Implementovat rozpoznávání textu z obrázků

Časový odhad: 8 MH

US09: Jako účetní chci importovat záznamy z tankovacích karet, abych je nemusel zadávat ručně a ušetřil čas.

Akceptační kritéria:

- Aplikace umožní nahrát soubor ve formátu csv se záznamy o tankování
- Z datového souboru lze do systému importovat datum tankování, množství paliva a cenu
- Systém umožní nastavit mapování sloupce z csv na položku v aplikaci

Možné úkoly:

- Přidat do formuláře tankování tlačítko importu csv
- Implementovat zpracování nahraného csv souboru
- Vytvořit stránku s nastavením mapování sloupců

Časový odhad: 8 MH

US10: Jako administrátor chci přidat nové vozidlo a uvést k němu značku, model, rok výroby, barvu, SPZ a VIN, abych mohl sledovat jízdy konkrétního vozu.

Akceptační kritéria:

- Je k dispozici formulář pro přidání nového vozidla se všemi uvedenými údaji
- Po potvrzení formuláře je vozidlo uloženo do evidence
- Systém validuje vstupy a zobrazuje relevantní chybové hlášky

Možné úkoly:

- Navrhnout databázový model pro vozidlo
- Vytvořit formulář pro přidání vozidla

Časový odhad: 8 MH

US11: Jako administrátor chci u vozidel evidovat termíny STK a servisních prohlídek podle času nebo ujetých kilometrů, abych je viděl přehledně na jednom místě.

Akceptační kritéria:

- Aplikace umožní u vozidla nastavit termín příští technické kontroly
- Aplikace umožní u vozidla nastavit interval servisních prohlídek podle času a ujetých kilometrů
- Je možné zaevidovat stav tachometru a datum poslední servisní prohlídky
- Systém vypočítá podle zadaných intervalů datum příští prohlídky a zbývající kilometry

Možné úkoly:

- Vytvořit formulář pro nastavení servisních intervalů a termínů
- Implementovat výpočet příští servisní prohlídky
- Zobrazit údaje v přehledu vozidel

Časový odhad: 4 MH

US12: Jako administrátor chci vidět přehled všech vozidel, který bude obsahovat jejich značku, model, SPZ, rok výroby, termíny servisních prohlídek a počet najetých km, abych měl přehled o stavu vozového parku

Akceptační kritéria:

- Je dostupný přehled vozidel se všemi uvedenými údaji
- Z přehledu je možné se prokliknout na detail vozidla

Možné úkoly:

- Vytvořit stránku přehledu vozidel

Časový odhad: 2 MH

US13: Jako administrátor chci zobrazit detail vozidla se všemi evidovanými údaji a možnostmi úprav vozidla, abych měl přehled o stavu konkrétního vozidla a mohl upravit údaje, které se mohou změnit

Akceptační kritéria:

- Je k dispozici stránka se všemi informacemi o vozidle
- Na stránce jde upravit barvu, SPZ, výchozího řidiče, poslední servisní prohlídka a příští STK

Možné úkoly:

- Vytvořit stránku s detaily o vozidle s možností editace a uložení údajů

Časový odhad: 2 MH

US14: Jako administrátor chci vytvářet nové uživatele, nastavit jim jméno, příjmení, uživatelské jméno a heslo a přiřazovat jim role řidič, administrátor a účetní, abych mohl řídit přístup do aplikace.

Akceptační kritéria:

- Existuje formulář pro vytvoření nového uživatele
- Do formuláře je možné vyplnit všechny uvedené údaje
- Systém validuje vstupy a zobrazuje relevantní chybové hlášky
- Po vytvoření se uživatel uloží do systému a může se přihlásit

Možné úkoly:

- Navrhnout databázový model uživatele
- Definovat systémové role a nastavit jejich přístup
- Vytvořit formulář pro založení uživatele

Časový odhad: 4 MH

US15: Jako uživatel chci zadat svoje uživatelské jméno a heslo a tím se přihlásit do systému, abych získal přístup k jeho používání

Akceptační kritéria:

- Po otevření systému se zobrazí přihlašovací obrazovka
- Uživatel je po vyplnění přihlašovacích údajů přihlášen a přesměrován na svou výchozí stránku podle role
- Pokud je přihlášení neúspěšné, zobrazí se chybová zpráva

Možné úkoly:

- Implementovat mechanismus ověření přihlašovacích údajů
- Nastavit přesměrování na správnou výchozí stránku podle role

Časový odhad: 4 MH

US16: Jako administrátor chci vidět přehled všech uživatelů, který bude obsahovat jejich jméno, příjmení, nastavené role a výchozí vozidlo s možností prokliku na úpravu uživatele, abych měl přehled o aktuálních zaměstnancích a mohl měnit jejich údaje

Akceptační kritéria:

- Je dostupný přehled uživatelů se všemi uvedenými údaji
- Z přehledu je možné se prokliknout na úpravu uživatele

Možné úkoly:

- Vytvořit stránku přehledu uživatelů s proklikem na úpravu

Časový odhad: 2 MH

US17: Jako administrátor chci přiřadit vozidlo konkrétnímu řidiči, aby bylo při zadávání jízdy automaticky předvyplněno a řidič šetřil čas.

Akceptační kritéria:

- Na kartě uživatele s rolí řidič je možné vybrat výchozí vozidlo
- Toto vozidlo má řidič implicitně vybrané při zadávání jízdy
- Řidič stále může vybrat pro jízdu jiné vozidlo

Možné úkoly:

- Rozšířit kartu uživatele o výběr výchozího vozidla
- Přidat do formuláře vytvoření jízdy načtení výchozího vozidla

Časový odhad: 2 MH

US18: Jako účetní chci zobrazit záznamy jízd jednotlivých vozidel a exportovat je ve formátu uznávaném finanční správou, abych mohl vyhovět kontrolám.

Akceptační kritéria:

- Účetní může zobrazit přehled všech jízd vybraného vozidla za zvolené období
- Přehled obsahuje všechny povinné informace o jízdě a jméno a příjmení řidiče
- Přehled je možné exportovat do souboru

Možné úkoly:

- Vytvořit přehled jízd filtrovaný podle vozidla a časového období
- Implementovat export záznamů do souboru

Časový odhad: 8 MH

6.2 Minimální životaschopný produkt

Pro vyvíjenou aplikaci je klíčová evidence jízd, proto je pro MVP nejdůležitější funkcionalita vytvoření jízdy. Tu je nutné evidovat ke konkrétnímu vozidlu, takže systém musí umožnit i jejich vytváření. Jak jízdy, tak vozidla je třeba zobrazit na přehledu. Aby mohli uživatelé v systému pracovat podle svých oprávnění, je nutné také vytváření uživatelů, nastavování jejich rolí a přihlášení. Všechny user stories jsou v Tabulce 1 rozděleny do následujících kategorií:

- MVP – nutné pro dosažení úrovně minimálního životaschopného produktu
- QoL – přináší zlepšení uživatelského zážitku při používání aplikace a často také šetří uživatelův čas
- Rozšíření – přidávají do aplikace užitečné doplňkové funkce

V rámci této práce budou vyvíjeny jen funkce patřící do MVP s důrazem na škálovatelnost, aby mohly být ostatní funkce do aplikace snadno přidány při dalším vývoji.

Tabulka 1 Kategorie User Stories

User Story	Kategorie
US01: Vytvoření jízdy	MVP
US02: Použití aktuální polohy zařízení jako start nebo cíl jízdy	QoL
US03: Výběr bodů jízdy z mapy	Rozšíření
US04: Výpočet ujeté vzdálenosti z mapy	QoL
US05: Přehled jízd v měsíci s možností úprav	MVP
US06: Přístup řidiče jen k vlastním jízdám	QoL
US07: Evidence tankování pohonných hmot a fotografie účtenky	Rozšíření
US08: Načtení údajů o tankování z účtenky	QoL
US09: Import z tankovacích karet	Rozšíření
US10: Vytvoření vozidla	MVP
US11: Evidence servisních termínů vozidla	Rozšíření
US12: Přehled všech vozidel	MVP
US13: Detail a úpravy vozidla	MVP
US14: Vytvoření uživatele a přiřazení role	MVP
US15: Přihlášení	MVP
US16: Přehled uživatelů s možností editace	MVP
US17: Přiřazení vozidla řidiči	QoL
US18: Zobrazení přehledu jízd vozidla a export	MVP

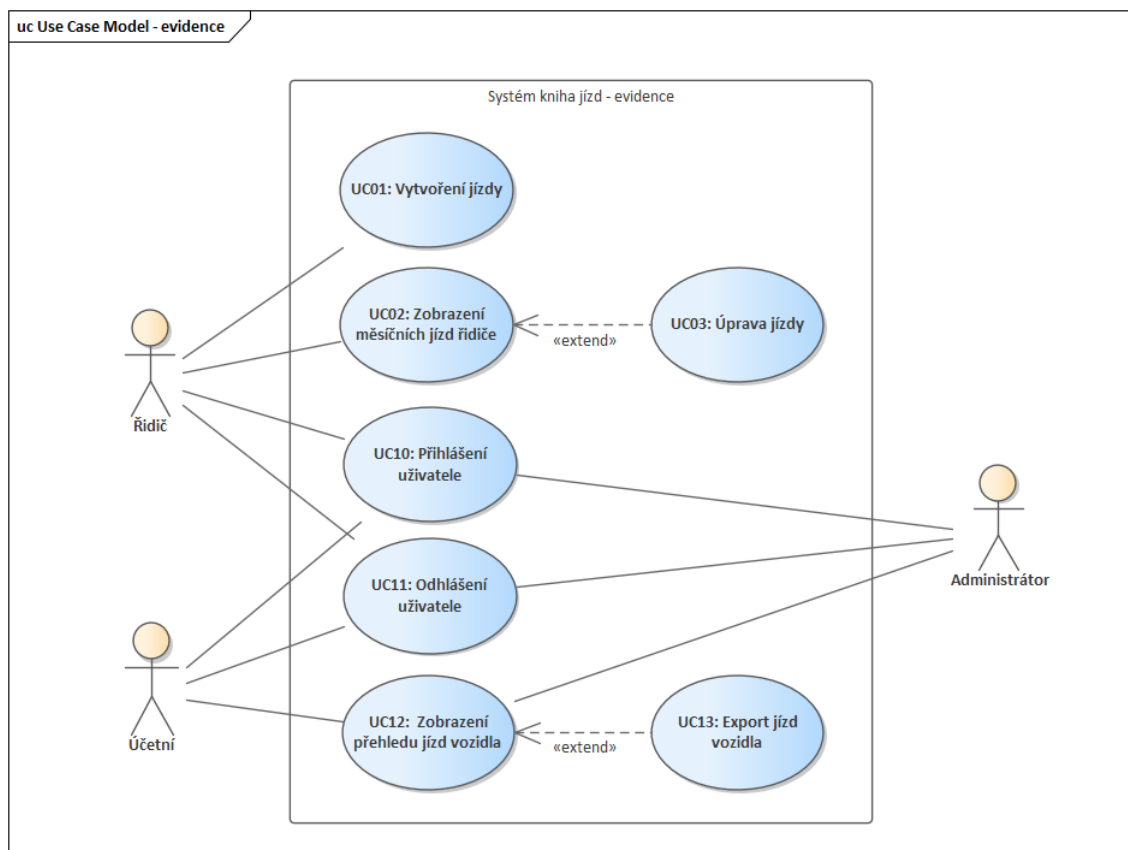
(zdroj: vlastní)

6.3 Model případů užití

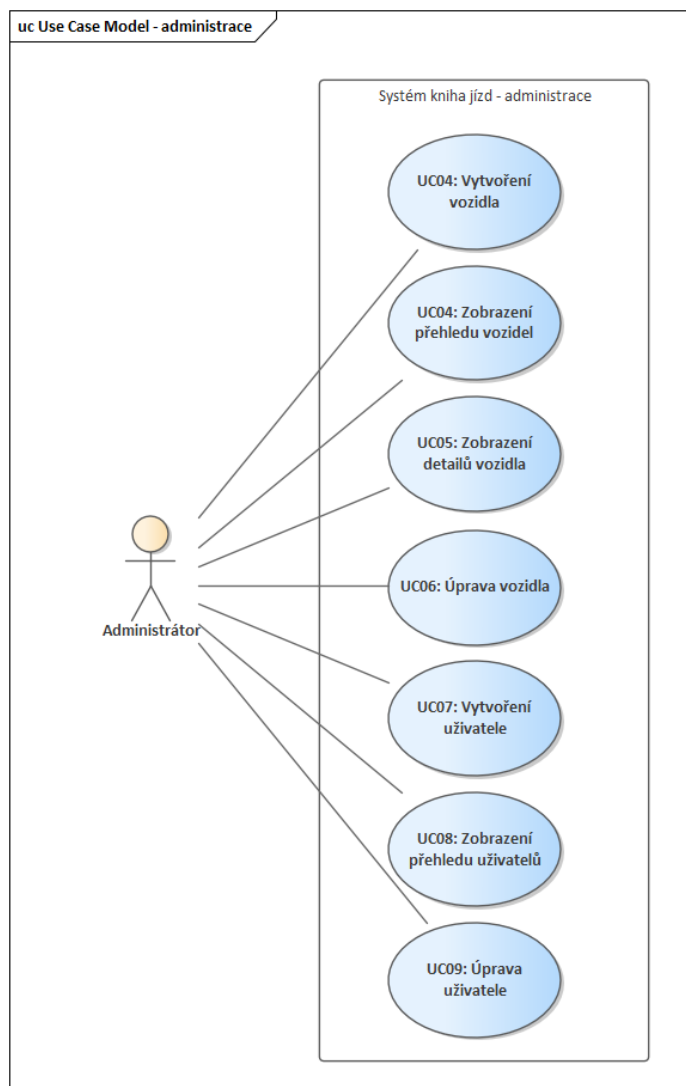
Tento model slouží k zachycení interakcí mezi uživatelem aplikací. Popisuje, jak uživatelé v jednotlivých rolích využívají systém k dosažení svých cílů. Pomocí případů užití můžeme formálně definovat hlavní funkce systému. Diagram případů užití pak poskytuje přehledné grafické znázornění těchto interakcí [18].

6.3.1 Diagram případů užití

Pro přehlednost byl diagram rozdělen na dvě části podle modulů systému. Na obrázku 1 je znázorněn diagram modulu evidence a na obrázku 2 pak diagram modulu administrace



Obrázek 1 Model případů užití modulu evidence



Obrázek 2 Model případů užití modulu administrace

6.3.2 Popis případů užití

V následující části jsou popsány jednotlivé případy užití.

UC01: Vytvoření jízdy

Aktéři: Řidič

Hlavní scénář:

1. Řidič otevře formulář vytvoření jízdy
2. Řidič vybere typ jízdy služební
3. Řidič vyplní všechny údaje specifikované v US01 a potvrdí formulář
4. Systém úspěšně validuje zadané údaje a uloží jízdu

Alternativní scénář 1:

2. Řidič vybere typ jízdy soukromá
3. Systém skryje nepovinná pole pro soukromou jízdu
4. Případ užití pokračuje bodem 3. hlavního scénáře

Alternativní scénář 2:

4. Systém zjistí chybu při validaci zadaných údajů
5. Systém zobrazí na formuláři chybovou zprávu
6. Případ užití pokračuje od bodu 3. hlavního scénáře

UC02: Zobrazení měsíčních jízd řidiče

Aktéři: Řidič

Hlavní scénář:

1. Řidič otevře přehled jízd
2. Systém vyhledá jízdy daného řidiče v aktuálním měsíci a zobrazí je v tabulce

Alternativní scénář 1:

3. Uživatel klikne v řádku tabulky na tlačítko detail jízdy
4. Spustí se UC03

UC03: Úprava jízdy

Aktéři: Řidič

Hlavní scénář:

1. Uživatel otevře detail jízdy
2. Systém zobrazí všechny údaje o jízdě s možností úprav
3. Uživatel upraví požadované údaje a klikne na tlačítko Uložit změny
4. Systém úspěšně validuje nová data
5. Systém uloží změny a přesměruje uživatele na přehled jízd

Alternativní scénář 1:

4. Systém zjistí při validaci dat chybu
5. Systém vrátí uživatele zpět na úpravu jízdy a zobrazí chybovou zprávu
6. Příklad užití pokračuje od bodu 3. hlavního scénáře

Alternativní scénář 2:

3. Uživatel klikne na tlačítko Zavřít
4. Systém přesměruje uživatele na přehled jízd

UC04: Vytvoření vozidla

Aktéři: Administrátor

Hlavní scénář:

1. Administrátor otevře formulář pro vytvoření vozidla
2. Administrátor vyplní všechny údaje specifikované v US10 a potvrdí formulář
3. Systém úspěšně validuje zadané hodnoty
4. Systém uloží vozidlo a přesměruje administrátora na přehled vozidel

Alternativní scénář 1:

3. Systém zjistí chybu při validaci zadaných údajů
4. Systém zobrazí na formuláři chybovou zprávu
5. Příklad užití pokračuje od bodu 2. hlavního scénáře

UC05: Zobrazení přehledu vozidel

Aktéři: Administrátor

Hlavní scénář:

1. Administrátor otevře přehled vozidel
2. Systém zobrazí v tabulce základní informace o vozidlech specifikované v US12

Alternativní scénář 1:

3. Uživatel klikne v řádku vozidla na tlačítko detaily vozidla
4. Spustí se UC06

UC06: Úprava vozidla

Aktéři: Administrátor

Hlavní scénář:

1. Administrátor otevře detaily vozidla
2. Systém zobrazí všechny údaje o vozidle a umožní úpravu údajů specifikovaných v US13
3. Uživatel upraví požadované údaje a klikne na tlačítko Uložit změny
4. Systém úspěšně validuje zadané údaje
5. Systém uloží nové údaje o vozidle a přesměruje uživatele na přehled vozidel

Alternativní scénář 1:

3. Uživatel klikne na tlačítko Zavřít
4. Systém přesměruje uživatele na přehled vozidel

Alternativní scénář 2:

4. Systém zjistí chybu při validaci údajů
5. Systém vrátí uživatele na detail vozidla s upravenými údaji a zobrazí chybovou zprávu
6. Příklad užití pokračuje od bodu 3. hlavního scénáře

UC07: Vytvoření uživatele

Aktéři: Administrátor

Hlavní scénář:

1. Administrátor otevře formulář vytvoření uživatele
2. Administrátor vyplní všechny údaje specifikované v US14, nastaví uživateli role a uloží formulář
3. Systém úspěšně validuje zadané údaje
4. Systém uloží nového uživatele

Alternativní scénář 1:

4. Systém zjistí chybu při validaci zadaných dat
5. Systém vrátí uživatele na vytvoření uživatele a zobrazí chybovou zprávu
6. Příklad užití pokračuje bodem 2. hlavního scénáře

UC08: Zobrazení přehledu uživatelů

Aktéři: Administrátor

Hlavní scénář:

1. Administrátor otevře přehled uživatelů
2. Systém zobrazí v tabulce informace o uživateli specifikované v US16

Alternativní scénář 1:

3. Uživatel klikne v řádku uživatele na tlačítko detaily uživatele
4. Spustí se UC09

UC09: Úprava uživatele

Aktéři: Administrátor

Hlavní scénář:

1. Administrátor otevře detaily uživatele
2. Systém zobrazí všechny údaje o uživateli a umožní úpravu údajů
3. Uživatel upraví požadované údaje a klikne na tlačítko Uložit změny
4. Systém úspěšně validuje zadané údaje
5. Systém uloží nové údaje o uživateli a přesměruje na přehled uživatelů

Alternativní scénář 1:

5. Uživatel klikne na tlačítko Zavřít
6. Systém přesměruje uživatele na přehled uživatelů

Alternativní scénář 2:

7. Systém zjistí chybu při validaci údajů
8. Systém vrátí uživatele na detail uživatele s upravenými údaji a zobrazí chybovou zprávu
9. Příklad užití pokračuje od bodu 3. hlavního scénáře

UC10: Přihlášení uživatele

Aktéři: Řidič, Administrátor, Účetní

Hlavní scénář:

1. Nepřihlášený uživatel otevře systém
2. Systém zobrazí přihlašovací formulář
3. Uživatel zadá uživatelské jméno a heslo a potvrdí formulář
4. Systém validuje zadané údaje a ověří existenci účtu
5. Ověření proběhlo úspěšně a uživatel je přihlášen
6. Systém přesměruje na výchozí stránku podle role

Alternativní scénář 1:

5. Ověření selhalo
6. Systém zobrazí přihlašovací formulář s vyplněným uživatelským jménem a zobrazí chybovou zprávu
7. Příklad užití pokračuje od bodu 3. hlavního scénáře

UC11: Odhlášení uživatele

Aktéři: Řidič, Administrátor, Účetní

Hlavní scénář:

1. Uživatel klikne na tlačítko odhlášení
2. Systém uživatele odhlásí a přesměruje na přihlašovací formulář

UC12: Zobrazení přehledu jízd vozidla

Aktéři: Účetní, Administrátor

Hlavní scénář:

1. Uživatel otevře stránku přehledu jízd vozidla
2. Uživatel vybere vozidlo a období, za které chce zobrazit jízdy
3. Systém vyhledá všechny jízdy, které odpovídají zadaným parametrům a zobrazí informace o těchto jízdách specifikované v US18 v tabulce

Alternativní scénář 1:

4. Uživatel klikne na tlačítko exportovat jízdy
5. Spustí se UC13

UC13: Export jízd vozidla

Aktéři: Účetní, Administrátor

Hlavní scénář:

1. Uživatel spustí export jízd do souboru
2. Systém vytvoří soubor PDF s údaji o vozidle (značka, model, SPZ), exportovaném období a tabulkou jízd za toto období
3. Systém nabídne tento soubor PDF uživateli ke stažení
4. Uživatel uloží soubor do svého zařízení
5. Systém přesměruje uživatele na přehled jízd vozidla

UC14: Zadání tankování

Aktéři: Řidič

Hlavní scénář:

1. Uživatel otevře formulář pro zadání tankování
2. Uživatel vyplní všechny údaje specifikované v US07
3. Uživatel nahraje fotografii účtenky a potvrdí formulář
4. Systém úspěšně validuje zadané hodnoty
5. Systém uloží fotografii účtenky jako soubor
6. Systém uloží tankování a přesměruje uživatele na přehled jízd

Alternativní scénář 1:

3. Systém zjistí chybu při validaci zadaných údajů
4. Systém zobrazí na formuláři chybovou zprávu
5. Případ užití pokračuje od bodu 2. hlavního scénáře

6.4 Model tříd

Model tříd zachycuje strukturu systému. Vizualizuje vztahy mezi jednotlivými třídami a obsahuje jejich vlastnosti a metody. Návrh tříd vychází z modelu případů užití a dále se používá pro implementaci doménové vrstvy systému, která představuje strukturu dat a business logiku aplikace [18].

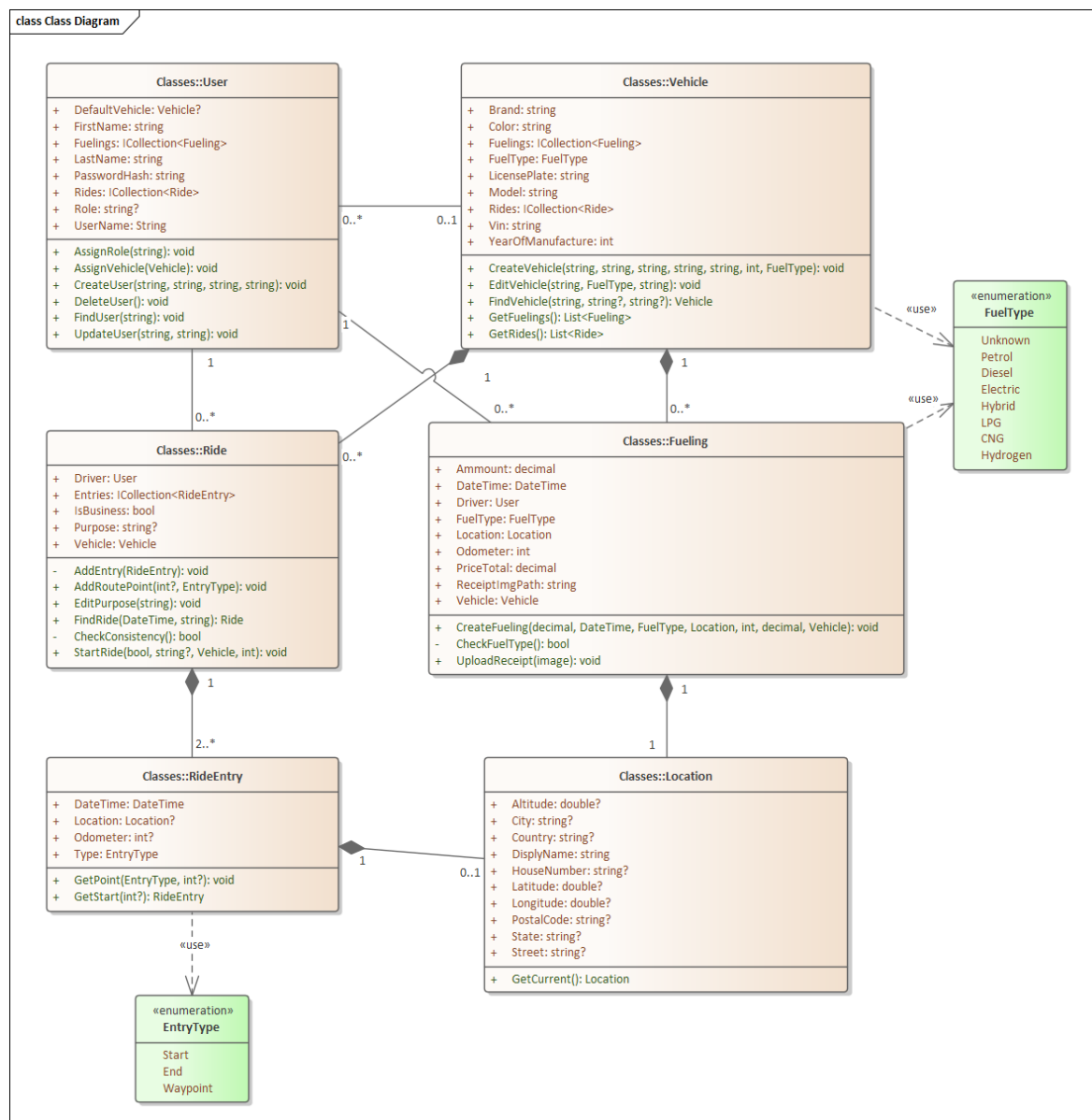
Byly identifikovány čtyři hlavní třídy, které budou reprezentovat data aplikace. První z nich je třída **Ride**, která vychází z UC1, UC2 a UC3 a bude použita také v UC12 a UC13. Dále třída **Vehicle** podle UC4, UC5 a UC6, třída **User**, se kterou pracují UC7 – UC11 a poslední třída **Fueling** pro UC14.

Aby se zjednodušila třída **Ride** a aby bylo možné aplikaci v budoucnu snáze rozšiřovat, je navržena také pomocná třída **RideEntry**, která bude součástí **Ride** a bude představovat jednotlivé body jízdy, především její počátek, případné průjezdní body a konec. Tato třída je také nositelem některých údajů o jízdě samotné v těchto bodech a obsahuje i druhou pomocnou třídu **Location**, která určuje polohu vozidla v bodě jízdy buď podle adresy, nebo podle souřadnic GPS.

Mezi třídami byly navrženy následující vztahy:

- Kompozice 1:N mezi Vehicle a Ride – vozidlo může obsahovat několik jízd, které bez vozidla nedávají smysl
- Kompozice 1:N mezi Vehicle a Fueling – vozidlo může obsahovat několik záznamů o tankování, které bez vozidla nedává smysl
- Asociace 1:N mezi User a Ride – uživatel může mít několik jízd a každá jízda má uživatele. Po jeho odstranění ale jízda zůstane v evidenci, proto se nejedná o kompozici
- Asociace 1:N mezi User a Fueling – uživatel eviduje několik tankování, tankování existuje i po odstranění uživatele.
- Asociace 0..1:N mezi Vehicle a User – uživatel může, ale nemusí mít přiřazené výchozí vozidlo. Vozidlo může mít více řidičů.
- Kompozice 1:2..N mezi Ride a RideEntry – jízda má nejméně 2 záznamy – začátek a konec a volitelně průjezdní body. Záznam patří k jedné konkrétní jízdě a bez ní nedává smysl.
- Kompozice 1:0..1 mezi RideEntry a Location – pozice patří ke konkrétnímu záznamu o jízdě a bez něj ztrácí význam. Záznam o jízdě nemusí mít pozici, pokud se nejedná o služební jízdu.
- Kompozice 1:1 mezi Fueling a Location – tankování probíhá v konkrétní lokalitě

Tyto vazby a vlastnosti jednotlivých tříd jsou znázorněny na obrázku 3



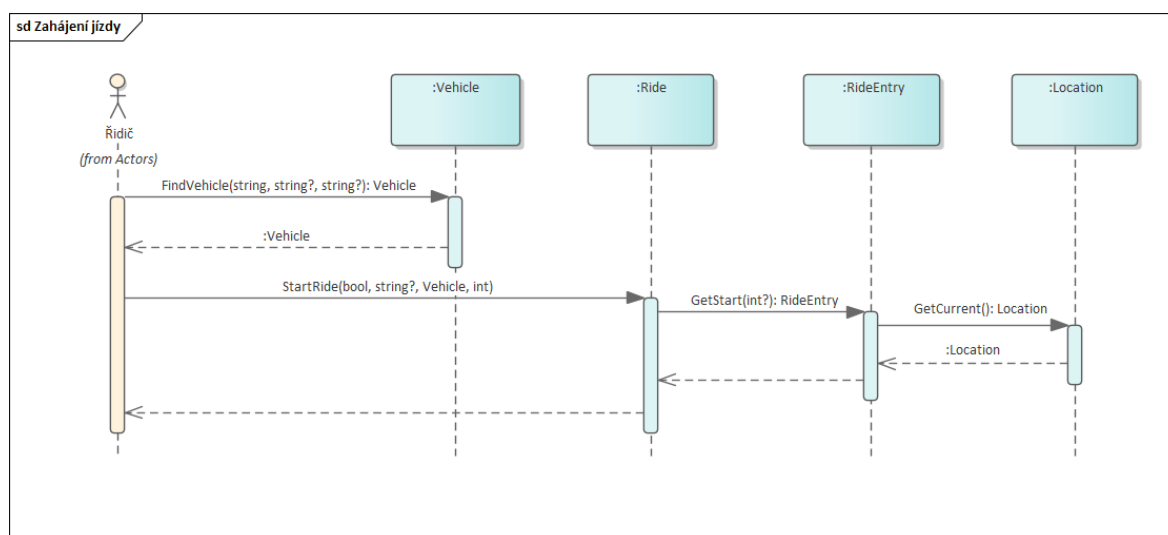
Obrázek 3 Model tříd

6.5 Dynamická analýza

V této kapitole je popsáno předávání informací mezi objekty během důležitých scénářů práce se systémem. Slovní popis je doplněn sekvenčními diagramy, které znázorňují volání metod mezi jednotlivými instancemi tříd v pořadí, v jakém během scénáře probíhají. Sekvenční diagramy jsou užitečné zejména při analýze a návrhu složitější logiky, protože pomáhají vizualizovat, jak si objekty předávají informace a jak spolu spolupracují při řešení konkrétní úlohy [18].

6.5.1 Zahájení jízdy

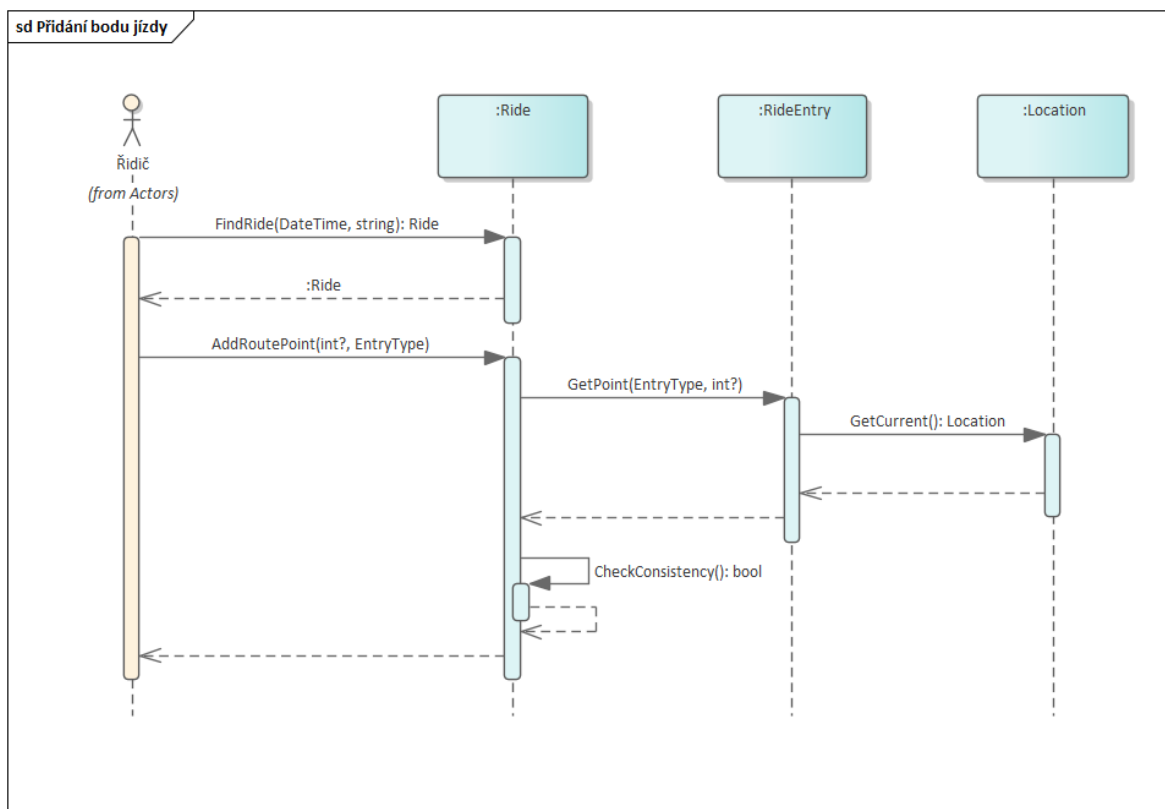
Prvním krokem pro zahájení jízdy je vyhledání vozidla, se kterým řidič jízdu uskutečnil. Následně se zavolá metoda `StartRide` na jízdě, které se informace, jestli se jedná o služební jízdu a pokud ano, tak i účel jízdy, dále vybrané vozidlo a aktuální stav tachometru. Jako řidič se vyplní aktuální uživatel, který metodu spustil. Poté se zavolá metoda `GetStart`, které se předá stav tachometru, tato zavolá metodu `GetCurrent` na `Location` pro získání aktuální polohy a uloží si ji spolu s aktuálním datem a časem. Jako typ se vybere automaticky vybere `Start` a `RideEntry` se vrátí do jízdy, která si ho uloží do seznamu svých záznamů. Tento postup je znázorněn v diagramu na obrázku 4.



Obrázek 4 Sekvenční diagram zahájení jízdy

6.5.2 Přidání bodu jízdy

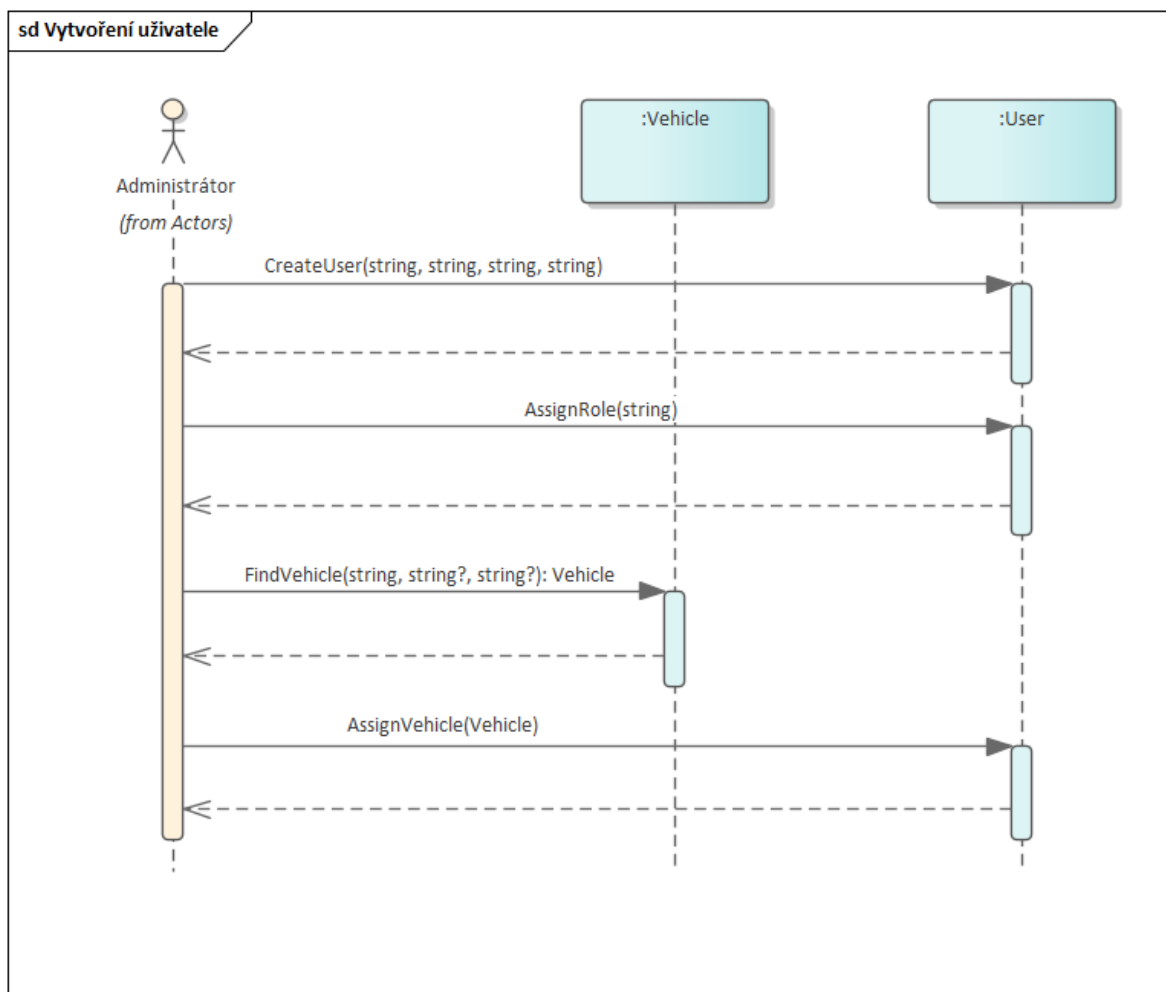
Tento scénář popisuje přidání bodu do existující jízdy, proto musí být jízda nejdříve vyhledána. Pak je dané jízdě předán stav tachometru a typ záznamu, aby bylo možné odlišit průjezdní bod od konce jízdy. Vytvoření `RideEntry` a `Location` proběhne stejně jako u zahájení jízdy, jen s rozdílem specifikace typu záznamu. Následně je nutné zkontrolovat konzistenci jízdních záznamů, především navazující datum s ohledem na typ záznamu (konec nesmí být před začátkem atp.) a rostoucí stav tachometru v čase. Tento scénář je na obrázku 5.



Obrázek 5 Sekvenční diagram přidání bodu jízdy

6.5.3 Vytvoření uživatele

Když administrátor vytváří nového uživatele, nejdříve se zavolá metoda `CreateUser`, do které se předají informace o novém uživateli. Pak se uživateli přiřadí role, aby mohl v systému pracovat, a nakonec i výchozí vozidlo, které je potřeba nejdříve vyhledat. Tento proces je vidět na obrázku 6.



Obrázek 6 Sekvenční diagram vytvoření uživatele

7 IMPLEMENTACE

7.1 Architektura aplikace

Aplikace je navržena podle principů Domain-Driven Design (DDD) v kombinaci s architekturou ASP.NET Core MVC. Cílem je jasně oddělit doménovou logiku od prezentační vrstvy a infrastruktury, aby bylo možné doménu udržovat nezávisle na ostatních technologiích. Řešení je složeno z několika projektů, které představují jednotlivé vrstvy aplikace.

7.1.1 Doménová vrstva (projekt Domain)

Obsahuje základní entity, se kterými aplikace pracuje. Definuje jejich vlastnosti a aplikační logiku. Nejdetailněji je zpracována entita Ride a jí podřízené, protože je pro funkcionalitu systému klíčová. Entity mají u datových polí zakázané úpravy zvenčí, aby nebylo možné obejít validace, které obsahují. S daty je tak možné manipulovat jen pomocí definovaných metod. Příkladem je Entita RideEntry:

```
[Owned]
public class RideEntry
{
    public DateTime DateTime { get; private set; }
    public int? Odometer { get; private set; } = null;
    public EntryType Type { get; private set; } = null!;
    public Location? Location { get; private set; } = null!;

    private RideEntry() { } // Parameterless constructor for EF Core
    and AutoMapper

    public RideEntry(DateTime dateTime, EntryType type, Location?
    location, int? odometer = null)
    {
        DateTime = dateTime;
        Type = type ?? throw new ArgumentNullException(nameof(type));
        Location = location;
        Odometer = odometer;
        Validate();
    }

    private void Validate()
    {
```

```
        if (DateTime == default)
            throw new ArgumentException("DateTime must be provided.");

        if (Type.IsStart || Type.IsEnd)
        {
            if (!Odometer.HasValue || Odometer.Value < 0)
                throw new ArgumentException("Odometer must be non-negative for start and end entries.");
        }
    }
}
```

Doménová vrstva dále obsahuje rozhraní repozitářů, které specifikují požadavky na přístup k datům a Lookup třídy, které slouží jako číselníky. Jejich implementace pomocí tříd místo enum umožňuje jejich správu z aplikace, která může být později implementována.

7.1.2 Aplikační vrstva (projekt Application)

Propojuje ostatní vrstvy aplikace pomocí služeb, které obsahují část aplikační logiky, provádí orchestraci entit a volají jejich metody. Služby jsou volány z prezentační vrstvy a po provedení požadované akce vrací zpět výsledek operace v návratovém typu `ServiceResult`, který indikuje úspěšnou nebo neúspěšnou operaci a případně obsahuje i `ErrorCode`, který upřesňuje důvod neúspěchu. Rovněž může v případě úspěšné operace přenášet data. Jako příklad je uvedena metoda `GetByIdAsync` ze služby `VehicleService`, která vyhledá v repozitáři vozidlo podle zadaného id a pokud je nalezeno, vrátí ho jako součást úspěšného `ServiceResult`. V opačném případě vrací chybný `ServiceResult`

```
public async Task<ServiceResult<VehicleDto>> GetByIdAsync(int id)
{
    if (id <= 0)
        return ServiceResult<VehicleDto>.Fail(ErrorCode.DataInvalid);

    var vehicle = await _vehicleRepository.GetByIdAsync(id);

    if (vehicle == null)
        return ServiceResult<VehicleDto>.Fail(ErrorCode.VehicleNotFound);

    return ServiceResult<VehicleDto>.Ok(_mapper.Map<VehicleDto>(vehicle));
}
```

Aplikační vrstva používá k předávání dat tzv. Data Transfer Object (DTO), který umožňuje řídit, která data z domény se dostanou do prezentační vrstvy a která zůstanou skryty.

7.1.3 Prezentační vrstva (projekt Web)

Představuje uživatelské rozhraní aplikace, jehož základem je ASP.NET Core MVC. Data jsou reprezentována ViewModely, které mají kromě aplikačních dat pole specifická pro zobrazení, např. položky výběru pro formulář na příkladu z CreateRideViewModel níže

```
[Required(ErrorMessage = "Vyberte prosím typ jízdy")]
[Display(Name = "Typ jízdy")]
public string SelectedRideType { get; set; } = "true"; // Default to
    business ride

public List<SelectListItem> RideTypes { get; set; } = [
    new("Služební", "true", true),
    new("Soukromá", "false")];
```

Data jsou zobrazeny pomocí Razor View šablon. Celá aplikace používá společné rozložení `_Layout.cshtml`, které obsahuje navigační lištu a patičku stránky. Interakce s uživatelským rozhraním jsou pak řízeny kontroléry, které zajišťují směrování http požadavků.

Prezentační vrstva také obsahuje soubor `Program.cs`, který slouží jako vstupní bod aplikace. V tomto souboru se načítá a zpracovává veškerá konfigurace systému, registrace služeb pro dependency injection a následně se spouští samotná aplikace.

7.1.4 Vrstva infrastruktury (projekt Infrastructure)

Implementuje rozhraní pro práci s daty (repozitáře) pomocí EF Core. Jejím jádrem je třída `AppDbContext`, která definuje strukturu databáze pomocí `DbSet`ů pro jednotlivé entity. Ty představují tabulky v databázi.

```
// Entities
public DbSet<Vehicle> Vehicles { get; set; }
public DbSet<Ride> Rides { get; set; }
public DbSet<Fueling> Fuelings { get; set; }

// Lookups
public DbSet<FuelType> FuelTypes { get; set; }
public DbSet<EntryType> EntryTypes { get; set; }
```

Dále pak v metodě `OnModelCreating` definuje vztahy mezi tabulkami a plní číselníky daty. Např.:

```
modelBuilder.Entity<EntryType>(
    et =>
    {
        et.Property("_id").HasColumnName("Id");
        et.HasKey("_id");

        et.HasIndex(ft => ft.Name)
            .IsUnique();
    }
);

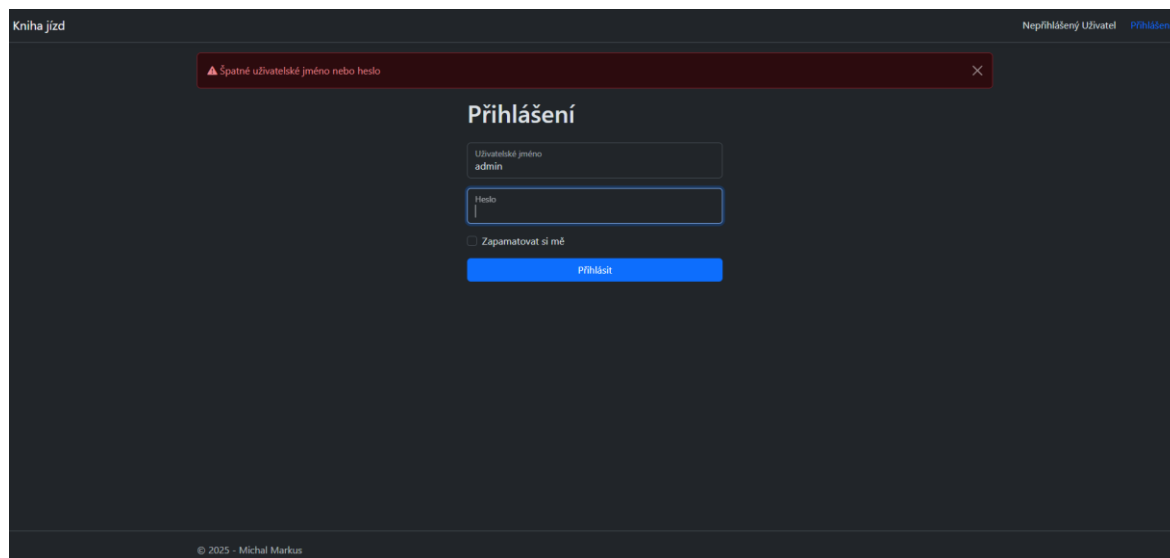
modelBuilder.Entity<EntryType>().HasData(
    new EntryType(id: 1, name: "Start", displayName: "Začátek"),
    new EntryType(id: 2, name: "End", displayName: "Konec"),
    new EntryType(id: 3, name: "Waipoint", displayName: "Průjezdni
bod")
);
```

Na základě této třídy se pak pomocí nástrojů dotnet generují skripty pro změnu struktury databáze, tak zvané migrace. Migrace umožňují i rollback změn do původního stavu.

Aplikace nemá žádné repozitáře specifické pro Entity. Vzhledem k rozsahu si vystačila s obecným `GenericRepository`, který poskytuje základní CRUD operace (Create, Read, Update, Delete) a díky šablonování ho lze použít pro různé entity. Pro složitější operace čtení se pak používá `QueryRepository`, který umožňuje vložení v podstatě libovolného LINQ dotazu, ten převede na SQL dotaz a vrátí požadovaný objekt. Výhodou tohoto přístupu je, že lze z databáze vybrat jen data, která jsou opravdu potřeba a není nutné načítat celé entity, což přispívá k optimalizaci výkonu.

7.2 Uživatelské rozhraní

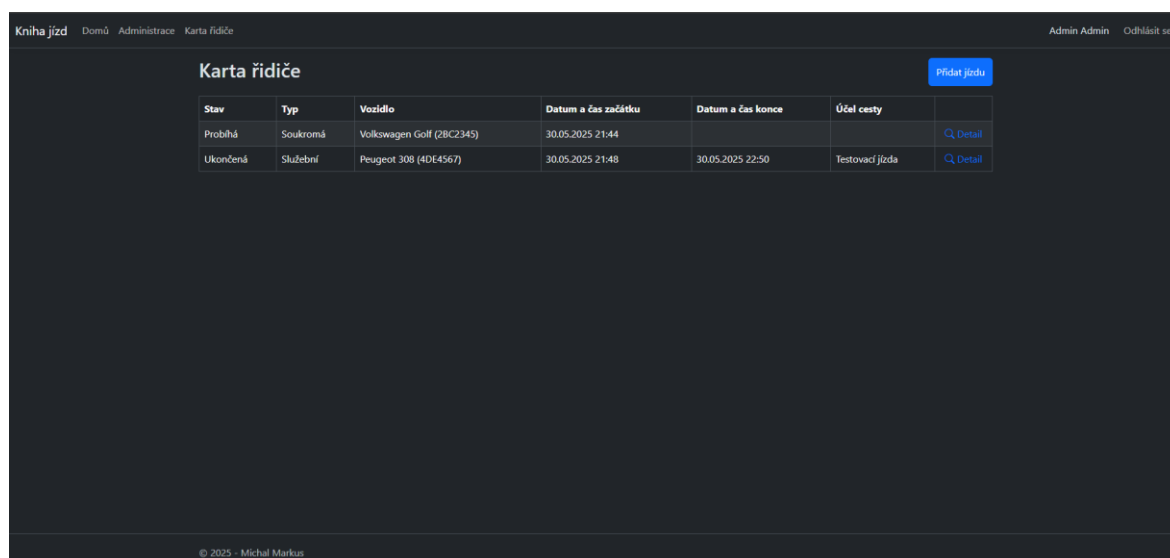
První stránka, která se uživateli zobrazí je přihlášení (Obrázek 7). Zde uživatel zadá své přihlašovací údaje a pokud jsou správné je přesměrován na domovskou obrazovku podle role. Pokud přihlášení selže, zobrazí se chybová zpráva, jak je vidět na obrázku 7.



Obrázek 7 Stránka přihlášení

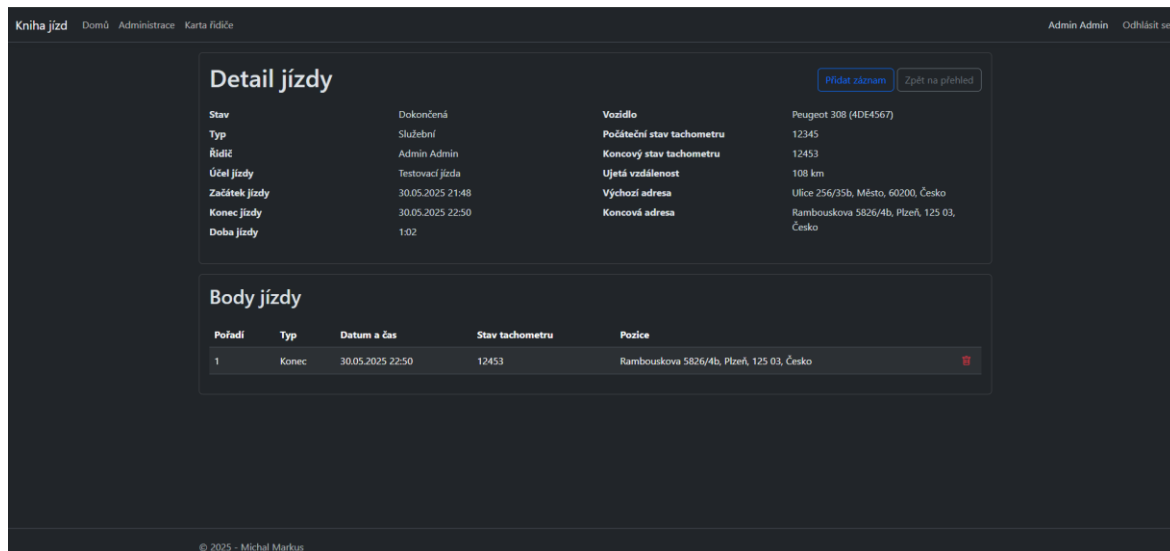
Chybová zpráva je realizována pomocí tzv. partial view `_Message.cshtml`, který slouží k zobrazení uvnitř jiného View, nebo např. v modálním okně. V tomto případě je zpráva vložena do `_Layout.cshtml` těsně před hlavní obsah. Tím je zajištěno, že každý View obsahuje zobrazení zprávy, které je navíc konzistentní napříč celou aplikací.

Protože uživatel může mít více rolí zároveň, bylo nutné stanovit prioritu rolí pro přesměrování. Nejvyšší prioritu má role Řidič, protože bude aplikaci používat nejčastěji. Následuje účetní a až na posledním místě je admin. Domovská stránka řidiče (Obrázek 8) obsahuje přehled jízd daného řidiče s možností zobrazit detail jednotlivých jízd nebo vytvořit novou jízdu.

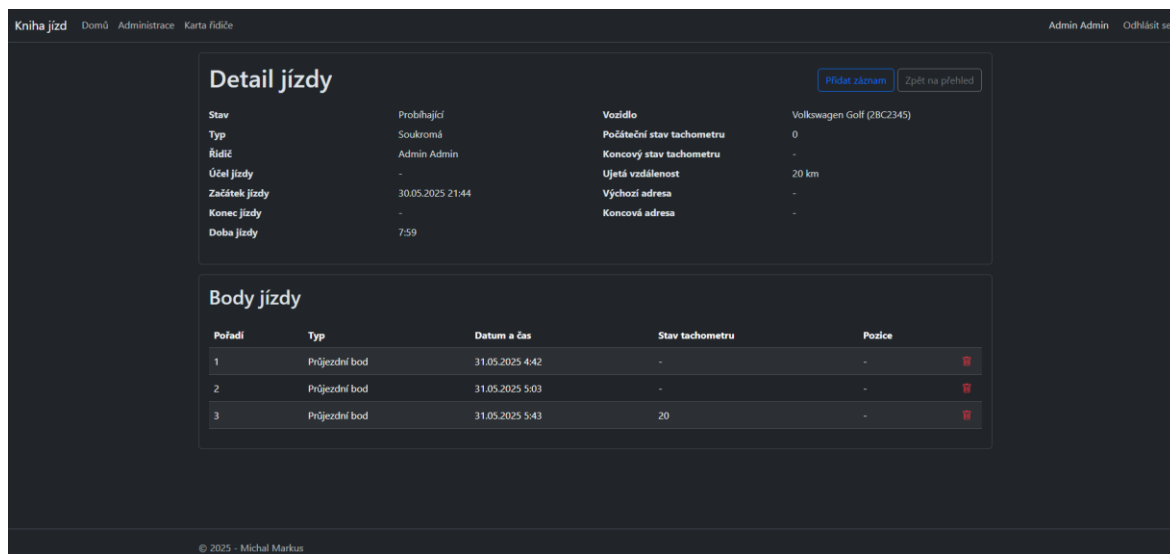


Obrázek 8 Domovská stránka řidiče

Po kliknutí na lupu se zobrazí detailní stránka, kde jsou vidět veškeré informace o jízdě včetně všech průjezdních bodů (Obrázek 9). Pokud nějaké údaje chybí, jsou vyplněny pomlčkou, jak je vidět na obrázku 10, aby bylo zřejmé, že se nejedná o chybu.



Obrázek 9 Detail služební jízdy



Obrázek 10 Detail soukromé jízdy

Po kliknutí na tlačítko přidat jízdu se zobrazí formulář na vytvoření nového záznamu o jízdě (Obrázek 11), ve kterém jsou podle typu jízdy skryta nepotřebná pole. Typicky adresa u soukromé jízdy. Validace se pak řídí jednak typem jízdy a druhá i vybraným typem záznamu. Pro průjezdní bod není povinný stav tachometru. Záznam je z jízdy možné odstranit kliknutím na ikonu koše v příslušném řádku. Poté se zobrazí dotaz na potvrzení operace (Obrázek 12) a když uživatel potvrdí, záznam je vymazán.

Kniha jízd Domů Administrace Karta řidiče Admin Admin Odhlásit se

Nový záznam

Zpět na detail

Typ záznamu Průjezdní bod

Datum a čas 30.05.2025 22:00

Stav tachometru 12350

Ulice Třída Tomáše Bati

Číslo popisné 268/42

Město Zlín

PSČ 325 02

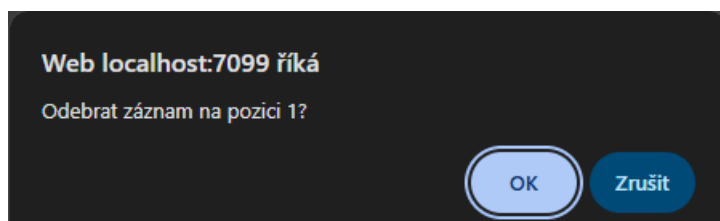
Země Česko

Stát, provincie nebo kraj Zlínský

Uložit

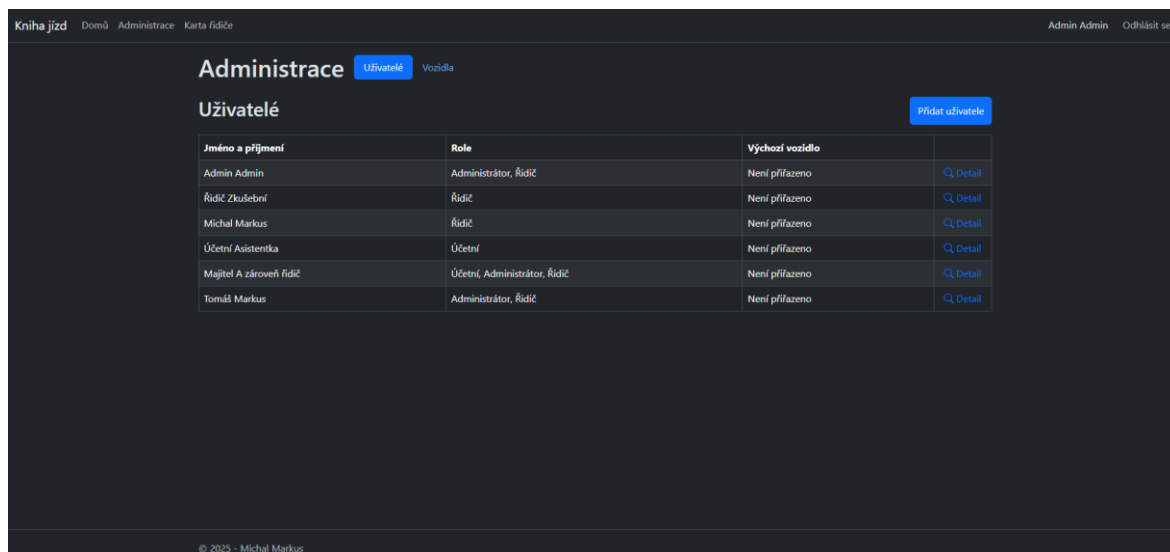
© 2025 - Michal Markus

Obrázek 11 Nový záznam jízdy



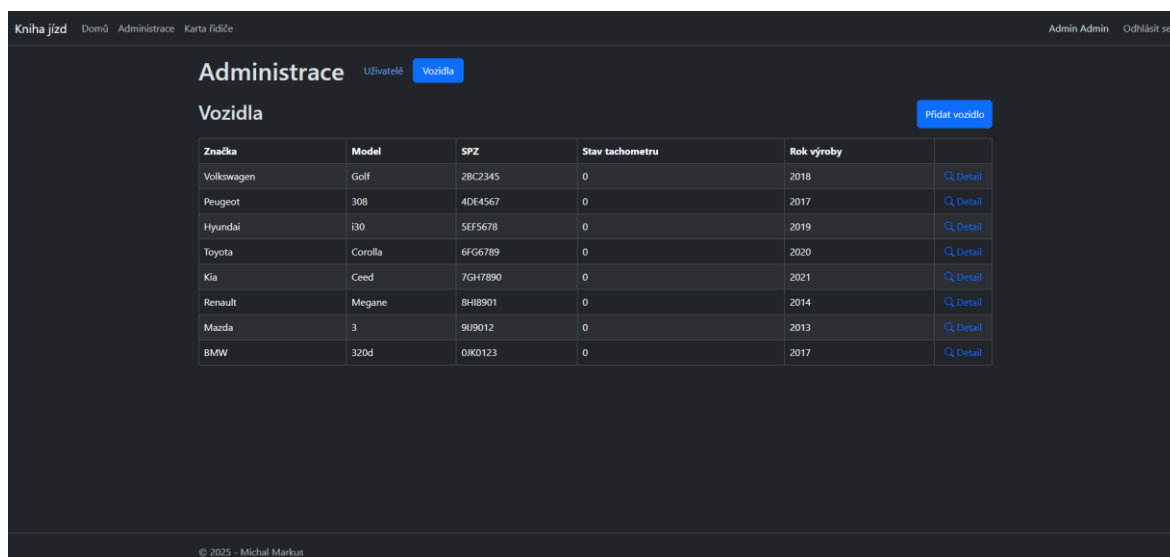
Obrázek 12 Potvrzení destruktivní akce

Samostatnou sekci uživatelského rozhraní je oblast administrace. Její hlavní stránka slouží jako domovská obrazovka pro roli Administrátor a dá se na ní přepínat mezi přehledem uživatelů (Obrázek 13) a přehledem vozidel (Obrázek 14). V obou zobrazeních se dá zobrazit detail uživatele nebo vozidla kliknutím na ikonu v příslušném řádku tabulky nebo vytvořit nový uživatel / vozidlo tlačítkem vpravo nad tabulkou.



Jméno a příjmení	Role	Výchozí vozidlo	
Admin Admin	Administrátor, Řidič	Není přiřazeno	Detail
Řidič Zkušební	Řidič	Není přiřazeno	Detail
Michal Markus	Řidič	Není přiřazeno	Detail
Účetní Asistentka	Účetní	Není přiřazeno	Detail
Majitel A zároveň řidič	Účetní, Administrátor, Řidič	Není přiřazeno	Detail
Tomáš Markus	Administrátor, Řidič	Není přiřazeno	Detail

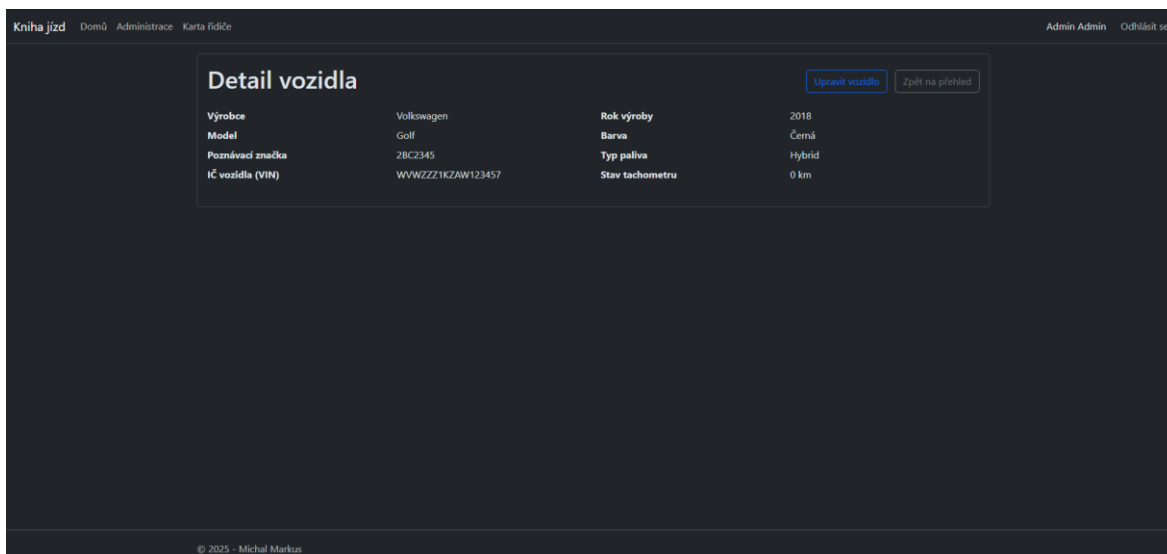
Obrázek 13 Přehled uživatelů



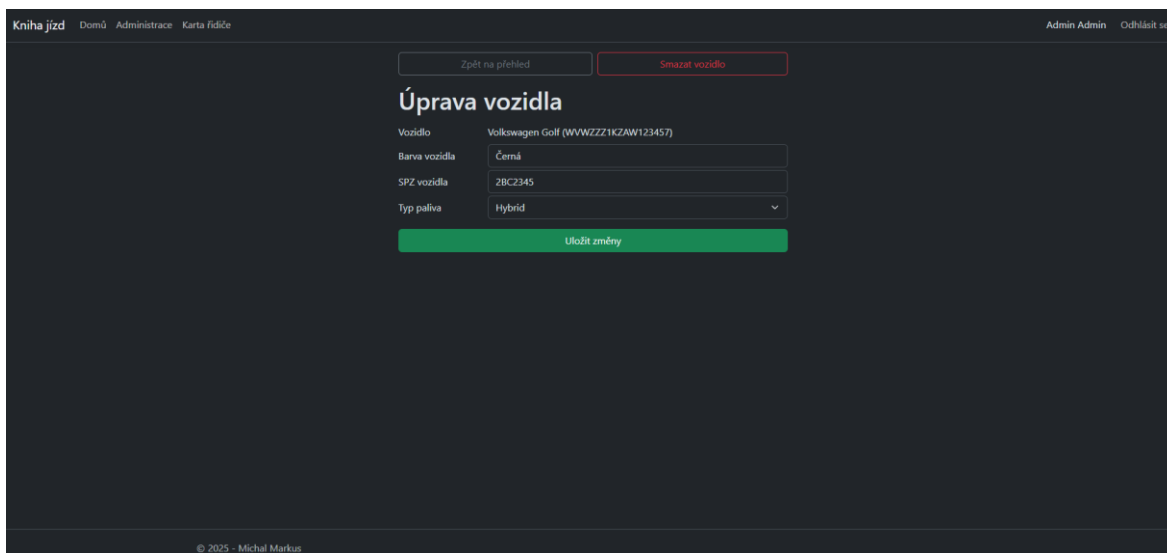
Značka	Model	SPZ	Stav tachometru	Rok výroby	
Volkswagen	Golf	2BC2345	0	2018	Detail
Peugeot	308	4DE4567	0	2017	Detail
Hyundai	i30	5EF5678	0	2019	Detail
Toyota	Corolla	6FG6789	0	2020	Detail
Kia	Ceed	7GH7890	0	2021	Detail
Renault	Megane	8HI8901	0	2014	Detail
Mazda	3	9IJ9012	0	2013	Detail
BMW	320d	0IK0123	0	2017	Detail

Obrázek 14 Přehled vozidel

Stránka detailu vozidla (Obrázek 15) zobrazuje všechny informace o vozidle a umožňuje rozkliknout formulář úpravy vozidla (Obrázek 16), kde je možné změnit některé údaje o vozidle nebo vozidlo po dvojitém potvrzení smazat. Pokud má vozidlo evidované nějaké jízdy, smazání nebude umožněno a zobrazí se chybová zpráva.

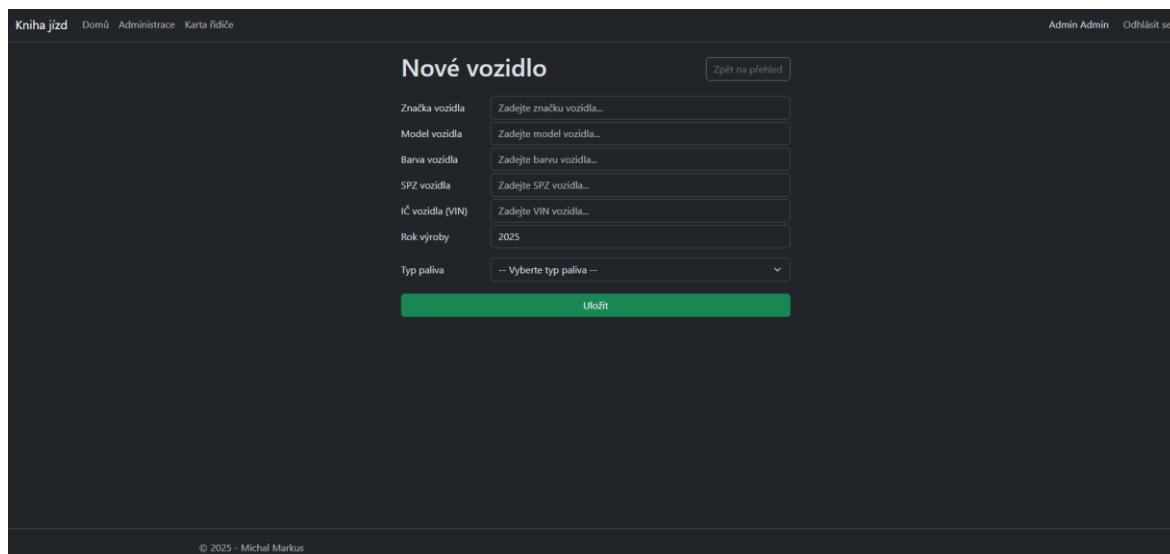


Obrázek 15 Detail vozidla



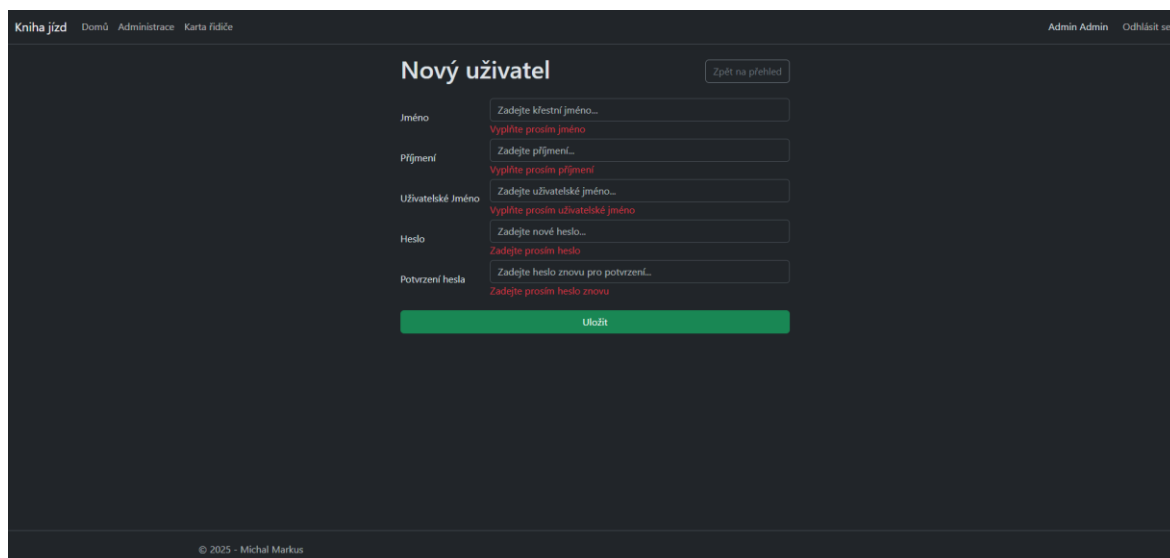
Obrázek 16 Úprava vozidla

Kliknutím na tlačítko Přidat vozidlo na přehledu vozidel se uživatel dostane do formuláře vytvoření nového vozidla, který je zobrazen na obrázku 17.



Obrázek 17 Formulář vytvoření vozidla

Po kliknutí na tlačítko přidat uživatele v přehledu uživatelů se zobrazí formulář pro vytvoření uživatele (Obrázek 18)



Obrázek 18 Formulář vytvoření uživatele

Po úspěšném uložení tohoto formuláře je administrátor přesměrován na stránku detailu uživatele (Obrázek 19), kterou může později otevřít kliknutím na tlačítko lupy v přehled uživatelů. Na této stránce je možné uživateli přiřadit roli výběrem z rozevíracího seznamu a potvrzením tlačítkem Přidat roli. Nová role se následně objeví v seznamu rolí nad výběrem a je možné ji uživateli opět odebrat pomocí tlačítka s ikonou koše.

Je také možné odemknout úpravu údajů stiskem tlačítka Upravit údaje. To spustí JavaScript, který odemkne pole formuláře a zobrazí tlačítka Uložit a Zrušit (Obrázek 20).

Tlačítkem Uložit se formulář odešle, zatímco tlačítkem zrušit se vymažou všechny provedené změny a formulář se vrátí do stavu, v jakém byl při načtení stránky. Tlačítkem smazat uživatele je pak možné po dvojitém potvrzení vymazat celý uživatelský účet.

Obrázek 19 Detail uživatele

Obrázek 20 Úprava uživatele

7.3 Práce s daty a komunikace mezi vrstvami

Během předávání dat mezi vrstvami aplikace dochází k opakovanému přepisování dat z jednoho objektu do druhého. Nejdříve z ViewModelu do DTO v kontroléru, pak v DTO na entitu ve službě a stejným procesem zpět při návratu. Abych zamezil opakování kódu, používám k tomuto přepisování knihovnu AutoMapper. Tato knihovna umožňuje, jak název napovídá, automatické mapování dat z jednoho datového typu na jiný. Stačí vytvořit

v projektu třídu, která dědí z Profile a v ní nadefinovat mapy. Minimální konfigurace Mapy zahrnuje jen zdrojový a cílový typ, na kterých má mapování probíhat a mapper automaticky spáruje vlastnosti, které mají stejný název a datový typ. AutoMapper je pak zaregistrován jako služba v Dependency Injection pomocí následujícího kódu:

```
// Add automapper
builder.Services.AddAutoMapper(
    typeof(RideRecords.Application.Common.ServiceResult).Assembly,
    typeof(RideRecords.Web.Common.MappingProfile).Assembly);
```

Pak je možné v jakémkoliv konstruktoru získat instanci přes rozhraní IMapper a na ní zavolat metodu Map, která provede převod.

8 ZABEZPEČENÍ

Webové aplikace čelí mnoha typům bezpečnostních hrozeb, kterým je potřeba se bránit. Mezi nejčastější hrozby patří zejména útoky SQL Injection, Cross-Site Scripting a Cross-Site Request Forgery a útoky na proces ověření a zabezpečení hesel [8]. Na tyto typy útoků jsem se proto zaměřil nejvíce.

8.1 SQL Injection

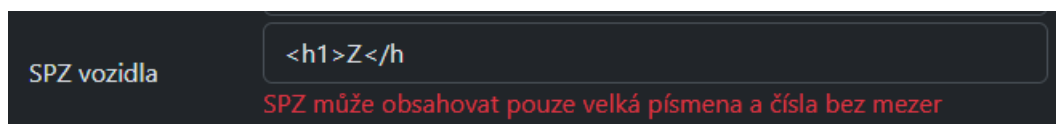
Tento útok spočívá v tom, že se útočník pokusí vložit škodlivý SQL kód do vstupního pole aplikace tak, aby byl začleněn do databázového dotazu. Takto modifikovaný dotaz pak může v databázi provést neočekávané změny nebo zobrazit data, ke kterým by uživatel jinak neměl přístup [19].

Účinnou obranou proti tomuto typu útoku je používání parametrizovaných dotazů. Ty fungují tak, že místo vkládání vstupního řetězce od uživatele přímo do SQL příkazu, se do databáze odešle samostatně šablona dotazu se zástupnými znaky (parametry) a následně samotná data v binární podobě. Díky tomu databázový server ví, že se případný kód v parametru nemá vykonávat a použije ho pouze jako hodnotu do připraveného dotazu [19; 8]. Vyvíjená aplikace je vůči tomuto typu útoku odolná díky přístupu k databázi výhradně přes EF Core, který na pozadí parametrizované dotazy používá [10].

8.2 Cross-Site Scripting

Při cross-site scriptingu útočník vloží do webové stránky škodlivý skript (typicky JavaScript), který se pak spustí v prohlížeči jiných uživatelů [19]. V kontextu evidence jízdy by mohl být tento útok proveden například tak, že by řidič do popisu jízdy vložil skript, který by se uložil do databáze třeba jako účel jízdy a později by se načetl do přehledu jízd pro účetní, kde by se vykonal.

Aplikace se tomuto útoku brání důslednou validací vstupů ve ViewModelu, které brání v uložení neočekávaných hodnot. Důležitější částí obrany je ale integrována přímo do ASP.NET Core a spočívá v escapování nebezpečných znaků před vykreslením proměnný v Razor View [7]. Při tomto procesu jsou např. znaky < a >, které uvozují html tagy včetně <script>, nahrazeny řetězcem, které slouží k zobrazení těchto znaků v HTML (např. < pro znak <). Escapování se provádí automaticky při vyhodnocení proměnných do View uvozených znakem @ [7]. Příklad obrany pomocí validace vstupů je vidět na obrázku 21 níže.



Obrázek 21 Selhání validace při pokusu vložit html tagy

Zabudovaná ochrana escapováním znaků je patrná na obrázku 22, kde se HTML zobrazené pomocí následujícího kódu zobrazí jako ostatní hodnoty, přestože je uzavřen v HTML tazích pro nadpis:

```
<div class="d-flex">
    <dt class="w-50">@Html.DisplayNameFor(m => m.Brand)</dt>
    <dd class="w-50">@Model.Brand</dd>
</div>
```



Obrázek 22 Escapování HTML tagů

8.3 Cross-Site Request Forgery (CSRF)

Principem tohoto útoku je, že útočník oklame legitimního uživatele, aby nevědomky poslal aplikaci požadavek, který ve skutečnosti pochází od útočníka [19]. K tomu může dojít například když uživatel klikne na podvržený odkaz. Když je uživatel k aplikaci přihlášen, může takto podvržený odkaz spustit nechtěnou akci, jako je třeba změna hesla [19].

ASP.NET Core obsahuje vestavěnou ochranu proti CSRF útokům pomocí anti-forgery tokenů. Framework generuje unikátní token (náhodný řetězec) pro každou relaci uživatele a vkládá jej jako skryté pole do každého formuláře odesílaného metodou POST [5]. Při obdržení požadavku pak server ověřuje, zda nechybí platný token [5]. Token nelze útočníkem jednoduše uhodnout a je vázán na uživatelskou relaci, takže podvržený požadavek z jiného webu nebude obsahovat platný token a aplikace jej rozpozná jako neoprávněný a odmítne zpracovat [5].

Toto ověřování je ve vyvíjené aplikaci povoleno globálně pro všechny metody spouštěné http pžadavkem POST, PUT nebo DELETE pomocí následujícího kódu ve vstupním souboru Program.cs:

```
builder.Services.AddControllersWithViews(options =>
{
    options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute());
});
```

Tokeny mohou být do formulářů vkládány buď manuálně pomocí tagu `@Html.AntiForgeryToken()` uvnitř formuláře, nebo automaticky při použití tzv. Tag Helperů, které se dají importovat na úrovni celé aplikace pomocí řádku `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers` v souboru `Views/_ViewImports.cshtml` [5].

8.4 Útoky na ověřování a zabezpečení hesel

Do této kategorie spadá využití slabín při zabezpečení hesel. Může jít o ukládání hesel jako čistý text, příliš slabá hesla, která jdou uhodnout hrubou silou v kombinaci s neomezenými pokusy na přihlášení, nebo také zneužití hesel uniklých z jiných služeb, pokud má uživatel stejné heslo nastavené i do této aplikace [19; 8].

Vyvíjený systém používá k ověřování uživatelů modul ASP.NET Identity, který ukládá hesla v zašifrované podobě [7], což je chrání v případě úniku databáze. Dále jsou v Identity nastaveny zásady bezpečnosti hesla a politika uzamčení účtu pomocí následujícího kódu v `Program.cs`:

```
builder.Services.AddIdentity<AppUser, AppRole>(options =>
{
    options.Password.RequiredLength = 8;
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireUppercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;
})
```

Uzamčení účtu je také třeba povolit při volání metody přihlášení:

```
var signInResult = await _signInManager.PasswordSignInAsync(
    vm.Username,
    vm.Password,
    vm.RememberMe,
    lockoutOnFailure: true);
```

Díky této konfiguraci je případné hádání hesla hrubou silou výrazně zpomaleno a zároveň požadavky na heslo zajišťují jeho dostatečnou komplexitu.

ZÁVĚR

Tato bakalářská práce se zabývala návrhem a vývojem webové aplikace pro evidenci firemních jízd se zaměřením na potřeby malých firem a podnikatelů. Úvodní rešerše ukázala, že na trhu existuje řada systémů, ale většina z nich je buď finančně a technicky náročná, nebo příliš jednoduchá a nepřístupná v terénu. Navrhovaná aplikace se snaží tento prostor vyplnit.

Při návrhu byl kladen důraz na jednoduchost použití, responzivní uživatelské rozhraní a možnost pozdějšího rozšíření o další funkce. Aplikace je postavena na architektuře ASP.NET Core MVC a využívá principy doménově řízeného návrhu (DDD), díky čemuž je logika systému oddělena od prezentační a datové vrstvy. Pro správu dat je použit Entity Framework Core, mapování zajišťuje AutoMapper a frontend je postaven na Bootstrapu. Funkcionalita je rozčleněna podle uživatelských rolí s využitím ASP.NET Identity.

Výsledná aplikace splňuje všechny cíle definované v zadání – umožňuje evidenci jízd, vozidel a uživatelů, zajišťuje přístupová práva, validaci data je připravena pro škálování i další rozšíření. Při vývoji byly uplatněny zásady bezpečného programování a aplikace odolává běžným typům webových útoků.

Do budoucna lze aplikaci rozšířit například o automatické zpracování fotografií účtenek, import z tankovacích karet nebo detailnější analytické přehledy. Z pohledu vývoje byla práce přínosná zejména v oblasti návrhu vícevrstvé architektury a integrace moderních technologií do uceleného systému.

SEZNAM POUŽITÉ LITERATURY

- [1] *Kniha jízd, Cestovní příkazy, Automapa - AUTOPLAN, KROB software s.r.o.* [online]. [2009] [cit. 2025-03-02]. Dostupné z: <https://www.autoplan.cz/>
- [2] *GPS Dozor - Satelitní sledování vozidel* [online]. c2025 [cit. 2025-03-02]. Dostupné z: <https://www.gpsdozor.cz/>
- [3] *GPS lokátory a sledování vozidel | Lokátory.cz* [online]. 2007-2025 [cit. 2025-03-02]. Dostupné z: <https://lokatory.cz/>
- [4] *Elektronická kniha jízd v Money S3* [online]. [2025] [cit. 2025-03-02]. Dostupné z: <https://money.cz/vlastnosti/kniha-jizd-s3/>
- [5] FREEMAN, Adam. *Pro ASP.NET Core MVC 2*. Seventh edition. London: Apress, [2017]. ISBN 978-1-4842-3149-4.
- [6] PRICE, Mark J. *C# 9 and .NET 5 - modern cross-platform development: build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code*. Fifth edition. Birmingham: Packt, 2020. ISBN 978-1-80056-810-5.
- [7] FREEMAN, Adam. *Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages*. 8th ed. 2020. Berkeley, CA: APress, 2020. ISBN 9781484254400. Dostupné také z: <http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=2494147&authtype=ip,shib&custid=s3936755>
- [8] HOFFMAN, Andrew. *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. 2. O'Reilly Media, 2024. ISBN 9781098143893.
- [9] SCHWICHTENBERG, Holger. *Modern Data Access with Entity Framework Core: Database Programming Techniques for .NET, .NET Core, UWP, and Xamarin with C#*. [Berkeley, CA]: Apress L.P., 2018, 1 online zdroj (665 stran). Dostupné z: doi:9781484235522
- [10] SMITH, Jon P. *Entity Framework Core in Action*. Manning, 2018. ISBN 9781617299525.
- [11] CHIARETTA, Simone. *Front-end Development with ASP.NET Core, Angular, and Bootstrap*. John Wiley & Sons, [2018]. ISBN 9781119181316.
- [12] NAGEL, Christian. *Professional C# 7 and .NET Core 2.0*. Indianapolis: Wrox, John Wiley & Sons, [2018], xviii, 1368 s. ISBN 9781119449270.
- [13] DANYLKO, Jonathan R. *ASP.NET 8 Best Practices: Explore techniques, patterns, and practices to develop effective large-scale .NET web apps*. Packt Publishing, [2023]. ISBN 9781837632121.
- [14] ANDREW, Lock. *ASP.NET Core in Action*. Third Edition. Manning, [2023]. ISBN 9781633438620.
- [15] KRUG, Steve. *Don't make me think, revisited: a common sense approach to web usability*. [Berkeley]: New Riders, [2014]. Voices that matter. ISBN 978-0-321-96551-6.
- [16] HARTSON, H. Rex a Pardha S. PYLA. *The UX book: agile UX design for a quality user experience*. Second edition. Cambridge, MA: Morgan Kaufmann, Elsevier, [2019]. ISBN 978-0-12-805342-3.

- [17] NEO, Giseldo, MOURA, José, Hyggo ALMEIDA, Alana NEO a Olival FREITAS JÚNIOR. User Story Tutor (UST) to Support Agile Software Developers. In: *Proceedings of the 16th International Conference on Computer Supported Education* [online]. SCITEPRESS - Science and Technology Publications, 2024, s. 51-62. ISBN 978-989-758-697-2. Dostupné z: doi:10.5220/0012619200003693
- [18] UNHELKAR, Bhuvan. *Software engineering with UML*. Boca Raton: Boca Raton : Auerbach Publications, [2018]. ISBN 978-1-351-23516-7. Dostupné také z: <https://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=5188116>
- [19] MCDONALD, Malcolm. *Grokking web application security*. Shelter Island: Manning, [2024]. ISBN 978-1-63343-826-2.

SEZNAM OBRÁZKŮ

Obrázek 1 Model případů užití modulu evidence.....	40
Obrázek 2 Model případů užití modulu administrace	41
Obrázek 3 Model tříd	52
Obrázek 4 Sekvenční diagram zahájení jízdy	53
Obrázek 5 Sekvenční diagram přidání bodu jízdy	54
Obrázek 6 Sekvenční diagram vytvoření uživatele	55
Obrázek 7 Stránka přihlášení	60
Obrázek 8 Domovská stránka řidiče	60
Obrázek 9 Detail služební jízdy	61
Obrázek 10 Detail soukromé jízdy	61
Obrázek 11 Nový záznam jízdy	62
Obrázek 12 Potvrzení destruktivní akce	62
Obrázek 13 Přehled uživatelů	63
Obrázek 14 Přehled vozidel	63
Obrázek 15 Detail vozidla	64
Obrázek 16 Úprava vozidla	64
Obrázek 17 Formulář vytvoření vozidla	65
Obrázek 18 Formulář vytvoření uživatele	65
Obrázek 19 Detail uživatele.....	66
Obrázek 20 Úprava uživatele.....	66
Obrázek 21 Selhání validace při pokusu vložit html tagy	69
Obrázek 22 Escapování HTML tagů	69

SEZNAM TABULEK

Tabulka 1 Kategorie User Stories	39
--	----

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ASP.NET	Active Server Pages .NET – webový framework od Microsoftu
BE	Backend (serverová část aplikace)
CDN	Content Delivery Network – síť pro distribuci statických souborů
CRUD	Create, Read, Update, Delete – základní operace nad daty
CSS	Cascading Style Sheets – kaskádové styly
CSRF	Cross-Site Request Forgery – podvržení požadavku napříč stránkami
DDD	Domain-Driven Design – doménově řízený návrh
DI	Dependency Injection – injekce závislostí
DTO	Data Transfer Object – přenosový objekt dat
EF	Entity Framework – ORM nástroj do Microsoftu
FE	Frontend (klientská část aplikace)
GPS	Global Positioning System – systém pro určování polohy
HTML	HyperText Markup Language – značkovací jazyk pro tvorbu webových stránek
HTTP	Hypertext Transfer Protocol – protokol pro komunikaci klient–server
HTTPS	Hypertext Transfer Protocol Secured – šifrovaný protokol pro komunikaci klient–server
LINQ	Language Integrated Query – jazyk pro dotazování nad objekty
MVC	Model–View–Controller – architektonický návrhový vzor
MVP	Minimum Viable Product – minimální životaschopný produkt
ORM	Object-Relational Mapping – mapování objektů na databázi
PDF	Portable Document Format – formát dokumentů
PHM	Pohonné hmoty
SQL	Structured Query Language – jazyk pro práci s databázemi
UI	User Interface – uživatelské rozhraní
URL	Uniform Resource Locator – jednotný identifikátor adresy

UX User Experience – uživatelský zážitek

XSS Cross-Site Scripting – útok vložení škodlivého skriptu do stránky

SEZNAM PŘÍLOH

Příloha P I: Kompletní zdrojový kód aplikace RideRecords

PŘÍLOHA P I: KOMPLETNÍ ZDROJOVÝ KÓD APLIKACE RIDEREORDS

Zdrojový kód aplikace RideRecords zabalený do zip archivu.

Pro sestavení a spuštění aplikace jsou dostupné 2 možnosti:

- Otevřít soubor řešení RideRecords.sln ve vývojovém prostředí Visual Studio Community a spustit projekt Web
- Na systému s nainstalovaným docker enginem spustit ve složce řešení příkaz
`docker compose --profile prod up`