Doporučené technologie, metody a postupy tvorby systému elektronické knihy jízd

Bc. Martin Čajánek

Diplomová práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Martin Čajánek**
Osobní číslo:	**A18547**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Doporučené technologie, metody a postupy tvorby systému elektronické knihy jízd**
Téma práce anglicky:	**Car Tracking System Technologies – Best Practices and Methods**

Zásady pro vypracování

1. Popište vhodné technologie a doporučené postupy pro tvorbu systému evidence elektronické knihy jízd.
2. Navrhněte vhodnou architekturu systému evidence elektronické knihy jízd.
3. Implementujte mobilní a webovou platformu pro klienskou část řešení.
4. Navrhněte a vytvořte využití geoserveru pro publikování mapových podkladů.
5. Vytvořte databázi pro perzistentní uchovávání dat.
6. Popište klíčové prvky řešení.

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

Seznam doporučené literatury:

1. SNIDER, Ed. Mastering Xamarin.Forms: Build Rich, Maintainable, Multi-platform, Native Mobile Apps with Xamarin.Forms. 2. Birmingham: Packt Publishing, 2018. ISBN 1788297342.
2. SEINTURIER, Lionel, Renaud PAWLAK a Jean-Philippe RETAILLÉ. Foundations of AOP for J2EE Development. 1. New York City: Apress, 2005. ISBN 1590595076.
3. YOUNGBLOOD, Brian a Stefano IACOVELLA. GeoServer: Beginner's Guide. 1. Birmingham: Packt Publishing, 2013. ISBN 9781849516686.
4. MASSÉ, Mark. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. 1. Sebastopol: O'Reilly Media, 2011. ISBN 1449319904.
5. FARKAS, Gabor. Mastering OpenLayers 3. 1. Birmingham: Packt Publishing, 2016. ISBN 1785281003.
6. RITCHIE, Christopher. WildFly Configuration, Deployment, and Administration. 2. Birmingham: Packt Publishing, 2014. ISBN 1783286237.

Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **28. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně

MARTIN ČAJÁNEK, V.R.

.....

podpis autora

ABSTRAKT

Cílem této diplomové práce je prozkoumat možnosti řešení tvorby komplexního systému pro evidenci elektronické knihy jízd. Tato práce se skládá z teoretické a praktické části. V teoretické části jsou prozkoumány již existující řešení, které tuto problematiku řeší a byly analyzovány jejich vlastnosti a nedostatky. Dále jsou v teoretické části popisovány vhodné technologie, na kterých by bylo možné systém stavět a jsou vyjmenovány i jejich případné alternativy. V praktické části je pak navržen systém elektronické knihy jízd a specifikovány požadavky, které by se na něj měly klást. Následně jsou popisovány postupy řešení implementace jeho klíčových prvků a způsoby překonání problémů při tvorbě systému.

Klíčová slova: Komplexní systém, kniha jízd, Angular, webová aplikace, Xamarin, mobilní aplikace, GeoServer

ABSTRACT

The aim of this diploma thesis is to explore the possibilities of solving the creation of a comprehensive system for the registration of electronic logbooks. This work consists of theoretical and practical part. The theoretical part examines existing solutions that address this issue and analyzed their properties and shortcomings. Furthermore, the theoretical part describes the appropriate technologies on which it would be possible to build the system and lists their possible alternatives. In the practical part, the system of electronic logbook is designed and the requirements that should be placed on it are specified. Subsequently, the procedures for solving the implementation of its key elements and ways to overcome problems in creating the system are described.

Keywords: Comprehensive system, logbook, Angular, web application, Xamarin, mobile application, GeoServer

Rád bych na tomto místě poděkoval panu Ing. Eriku Královi Ph.D za trpělivost a podnětné rady při vedení této diplomové práce.

OBSAH

ÚVOD

Za zavedením počítačů byla zejména snaha o automatizaci, vyšší efektivitu práce a ušetření času. Bohužel v některých oblastech života se stále setkáváme s řešeními, která nereflektují aktuální stav technické vybavenosti společnosti a existuje u nich možnost ke změně.

V rámci této práce je tím míněna evidence knihy jízd, kterou některé firmy stále vedou v nepraktické a nespolehlivé papírové podobě, snadno ovlivnitelné chybou lidského faktoru, ačkoliv se nabízí modernější a efektivnější způsob jejího provozu.

V mé předložené diplomové práci se tedy snažím řešit problematiku tvorby elektronického systému evidence knihy jízd. Tento systém by měl pomoci přejít firmám na lepší úroveň vedení evidence.

Toto téma jsem si vybral protože kolem sebe vidím, jak nepraktické řešení je evidence jízd v papírové formě a zároveň neexistuje jednoduché technické řešení na trhu, které by to nahrazovalo. V dnešní době, když prakticky každý vlastní chytrý telefon, je rozumné najít řešení, které ho bude využívat namísto lehce ztratitelného papírového dokumentu.

V teoretické části je čtenáři vysvětlen pojem evidence knihy jízd a důvody, proč knihu jízd vést. Dále je seznámen s existujícími řešeními a jsou zmíněny jejich přednosti i nevýhody. Následně jsou popsány technologie a jejich alternativy, které jsou pro vytvoření elektronické knihy jízd použity. V rámci teoretické části jsou i prozkoumány legislativní omezení, které se systému týkají dnes a nebo se ho mohou týkat v budoucnu.

V praktické části je navržena architektura celého systému a jsou definovány požadavky, které by měly jednotlivé části systému splňovat. Součástí této práce je i implementace samotného systému - jeho webové aplikace, mobilní aplikace a integrace s geografickým informačním systémem. Jsou popsány tyto klíčové prvky a problémy, na které bylo v průběhu práce naráženo a způsoby či doporučené techniky, jak byly tyto problémy řešeny.

I. TEORETICKÁ ČÁST

1 Kniha jízd

V této kapitole bude vysvětlen termín "kniha jízd", popsána jeho struktura a důvody, proč knihu jízd[1)] vést.

"Kniha jízd je vlastně účetní doklad, kde jsou zaznamenány údaje o jednotlivých jízdách vozidla v obchodním majetku či soukromého vozu používaného k podnikání. V praxi se můžeme setkat s tím, že se kniha vede buď elektronicky, nebo v papírové podobě."[1]

Obecně existují dva důvody proč vést knihu jízd - bezpečnost práce a finanční (daňové) účely. Zákony, které se týkají této problematiky jsou uvedeny níže v následujících kapitolách.

Kniha jízd by měla pro každou jízdu obsahovat[1]:

- **Datum a čas odjezdu a příjezdu** - Datum a čas začátku a konce cesty
- **Destinace** - místo začátku a místo konce cesty
- **Účel** - účel jízdy
- **Počet ujetých kilometrů** - počáteční stav a konečný stav tachometru

Dále by měla obsahovat následující informace[1]:

- **SPZ** - státní poznávací značku vozidla
- **Typ vozidla** - typ provozovaného vozidla
- **Průměrná spotřeba** - průměrná spotřeba paliva podle technického průkazu
- **Záznam o čerpání pohonných hmot** - Počet načerpaných litrů paliva/nabitých kWh a cenu
- **Bezpečnostní přestávky** - V případě povinnosti čerpat bezpečnostní přestávky uvést kdy začaly a končily
- **Zahájení činnosti používání vozidla** - Datum zahájení činnosti provozu vozidla.
- **Ukončení činnosti používání vozidla** - Datum ukončení činnosti provozu vozidla

[1)] V této kapitole i celé diplomové práci bude různě zaměňován termín "cesta", "jízda"či "trasa". Vždy to nicméně znamená činnost vedenou ve vozidle po nějaké trajektorii za určitým účelem, započatou a ukončenou v některém bodě, trvající omezenou dobu, ke které jsou navázány další informace.

- **Stav tachometru k 1.1 a 31.12** - Uvedení stavu tachometru k 1. lednu a 31. prosinci.

2 Systém evidence elektronické knihy jízd

V této kapitole budou definovány důvody, proč je vhodnější využívat elektronickou knihu jízd namísto vedení knihy jízd v papírové podobě. Následně budou také prozkoumána již existující řešení elektronických knih jízd či podobných produktů a porovnány jejich výhody a nevýhody.

2.1 Požadavky na elektronickou evidenci knihy jízd

Důvody pro vedení knihy jízd v elektronické podobě jsou zřejmé - vedení je přehlednější (informace jsou jednoduše dostupné), obecně rychlejší, přesnější i pohodlnější.

2.2 Analýza současných řešení

Jak již bylo zmíněno v úvodu, již existují na trhu produkty, která se snaží nabízet své vlastní řešení elektronické knihy jízd. V této diplomové práci bylo zvoleno několik různých řešení, které zaujaly svými vlastnostmi či způsobem, jakým k dané problematice přistupují.

V rámci porovnání řešení byla také stanovena řada kritérií, které byla považována za důležité a byla u sledovaných existujících řešení porovnávána.

Sledovaná kritéria

- **Typ aplikace** - Je aplikace dostupná pouze formou instalace na počítači (desktopová aplikace) či dostupná z jakéhokoli počítače přes webový prohlížeč (webová aplikace)?
- **Sledování jízdy** - Je nutné speciálního fyzického zařízení pro sledování pozice? Sleduje či eviduje se pozice při jízdě?
- **Možnost vlastního provozu** - Umožňuje provozovat systém v rámci zákazníka nebo je třeba využívat prostředky provozovatele služby a tedy předávat data třetí straně?
- **Cena** - Je systém dostupný zdarma, nutné zakoupení licence či je třeba platba měsíční/jednorázová?

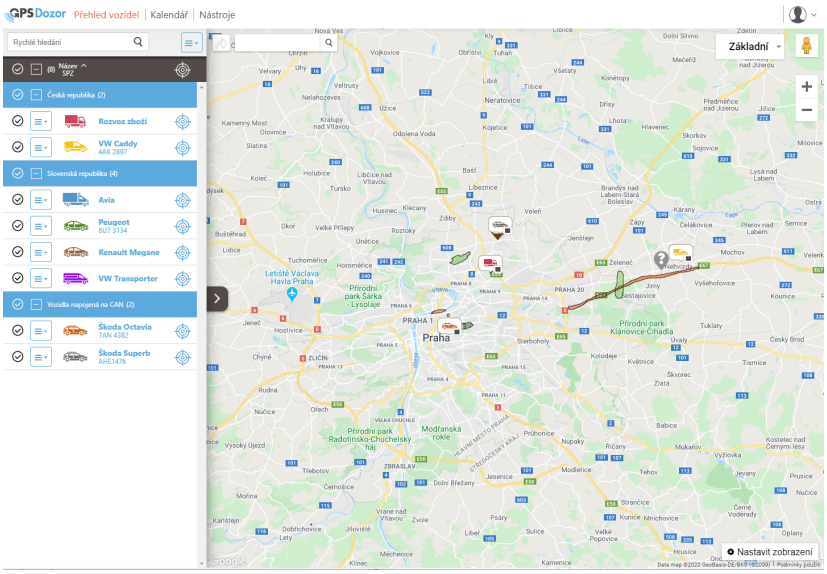
GPS Dozor

Systém satelitního sledování vozidel GPS Dozor se prezentuje jako vysoce profesionální komplexní systém, který přináší zákazníkům za jejich prostředky velkou protihodnotu. Nabízí přehlednou knihu jízd s historií pohybu - zmiňuje dostupná[1)] a bezpečně uložená

[1)] Systém GPS Dozor nabízí zákazníkům zabezpečené API pro stahování dat ze svých serverů.

data. Také nabízí úsporu až 40 % nákladů na provoz vozidla díky zamezením černým jízdám.

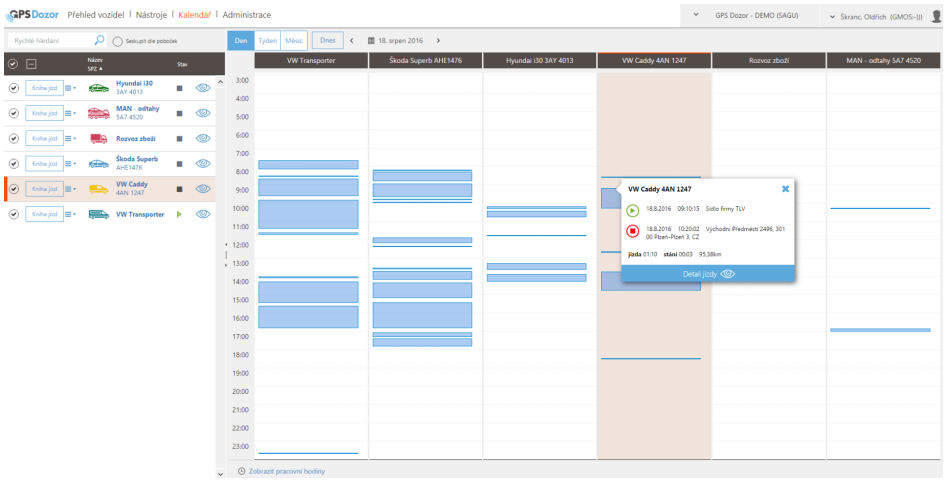
Společnost GPS Dozor také zmiňuje tři unikátní funkce, díky kterým je (GPS Dozor) "krok před konkurencí". Jedná se o chytrou mapu hned na úvodní obrazovce, interaktivní knihu jízd a kalendářové zobrazení knihy jízd[2].



Obr. 2.1 Rozhraní aplikace GPS Dozor,zdroj vlastní

Systém poskytuje přehledné prostředí se zobrazením aktuálních pozic vozidel na mapách i jejich historii. Je také umožněn export knihy jízd do formátu, popisovaného jako "respektovaný finančními úřady bez výhrad". Jako další z užitečných vlastností lze zmínit možnost nastavení povolené oblasti pohybu, kdy při jejím opuštění je generováno automaticky upozornění.

Velmi užitečnou funkcí je také kalendářové zobrazení pohybu více vozidel viz obrázek 2.2, kde je možné jednoduše zjistit kdo začíná práci ve správný čas a kdo naopak pozdě či kdo nedodržuje dohodnutou pracovní dobu.



Obr. 2.2 Rozhraní aplikace GPS Dozor - kalendářový pohled,zdroj GPS Dozor

Společnost aktuálně (březen 2020) nabízí sedm různých produktů, které se liší svým zaměřením, rozšiřitelností, parametry a také samozřejmě i svou cenou. Pro sledování osobních automobilů doporučuje následující čtyři produkty v tabulce 2.1.

Tab. 2.1 Doporučované varianty produktů GPS Dozor pro osobní automobily

Produkt	Modularita	Rychlost odezvy	Cena bez DPH
BASIC	Ne	5 - 10 minut	2390 Kč
PREMIUM	Ano	3 - 5 minut	3590 Kč
CAN	Ano	3 - 5 minut	4890 Kč
ALARM	Ano	30 sekund	5590 Kč

Uvedená cena v tabulce 2.1 je cena za jedno vozidlo. K ní ještě je nutné připočíst paušální poplatek 200 Kč měsíčně pokud bude vozidlo provozováno pouze v ČR, případně 355 Kč měsíčně, pokud bude vozidlo provozováno i v zahraničí[2)].

Všechny produkty od společnosti GPS Dozor vyžadují montáž externího GPS lokátoru do vozidla, což poskytuje v některém z desítek partnerských servisů či přímo u zákazníka.

[2)] Informace získány z aktuální nabídky (duben 2020) ze stránky www.gpsdozor.cz

Obr. 2.3 Fyzická jednotka GPS Dozor Premium, zdroj GPSDozor.cz

Systém GPS Dozor také v ceně produktu nabízí mobilní aplikaci na platformu iOS a Android pro sledování pozice vozidel, která poskytuje podobné funkce jako ve webové monitorovací aplikaci, nikoli měření pozice vozidla v samotné webové aplikaci.

AutoGPS

Produkt AutoGPS společnosti Eurosat CS, řešící elektronickou knihu jízd, je popsán jako "Monitorování osobních a nákladních vozidel, osob, zboží, stavebních a zemědělských strojů a dalších pohybujících se zařízení"[3].

Dále lze o produktu zjistit, že "elektronická kniha jízd AutoGPS slouží nejen k GPS sledování osobních a nákladních automobilů, ale také s ní lze monitorovat stavební a zemědělské stroje, těžkou techniku, závodní stroje či lodě". Také uvádí, že "velkou pozornost věnují monitorování transportů a zásilek cenného zboží", také že "v neposlední řadě může nainstalované GPS zařízení zachránit v případě krádeže a uspoří peníze za havarijní pojištění (cca 7%)"[3].

V rámci této diplomové práce je řešena kniha jízd pro osobní, nákladní vozidla a motocykly, proto se zaměříme na tuto oblast. Systém AutoGPS pro zjišťování aktuální polohy osobních automobilů využívá fyzickou komunikační jednotku PATRIOT XV, do které je vložena SIM karta, využívaná na přenos dat přes GPRS přenášena data do centrálního systému.

Kniha jízd je implementována jako webová aplikace hostovaná u poskytovatele. Umožňuje "výstup dat ve formě více než 150 různých reportů a grafů, které je možné využít pro daňové účely". Také poskytuje "propracovaný systém upozornění pomocí SMS nebo e-mailu". Aplikace také "přehledně zobrazuje všechny důležité informace z historie jízd a poloh sledovaného objektu"[3].

Systém je zpoplatněn zakoupením komunikační jednotky, SIM karty, ročního před-

platného a dalších položek - viz následující tabulka 2.2

Tab. 2.2 Ceník produktu AutoGPS

Položka	**Cena**	**Jednorázově/pravidelně**
PATRIOT XV	4500 Kč	jednorázově
Auto-GPS SIM	100 Kč	jednorázově
Auto-GPS roční předplatné	1500 Kč	pravidelně
GPRS Přenos dat roční předplatné	660 Kč	pravidelně
PATRIOT DALLAS čtečka	399 Kč	jednorázově

Celkem tedy lze za jeden automobil zaplatit za první rok 7159 Kč (bez DPH) a následně 2160 Kč (bez DPH) za každý další rok využívání služby. Objednávky služby pro větší počet automobilů se pak týkají množstevní slevy[3)].

AUTOPLAN Kniha jízd

AUTOPLAN Kniha jízd je aplikace společnosti KROB software, sloužící k evidenci a vyhodnocování provozu vozidel. Aplikace slouží jako elektronická kniha jízd, která poskytuje i pokročilou funkcionalitu [4].

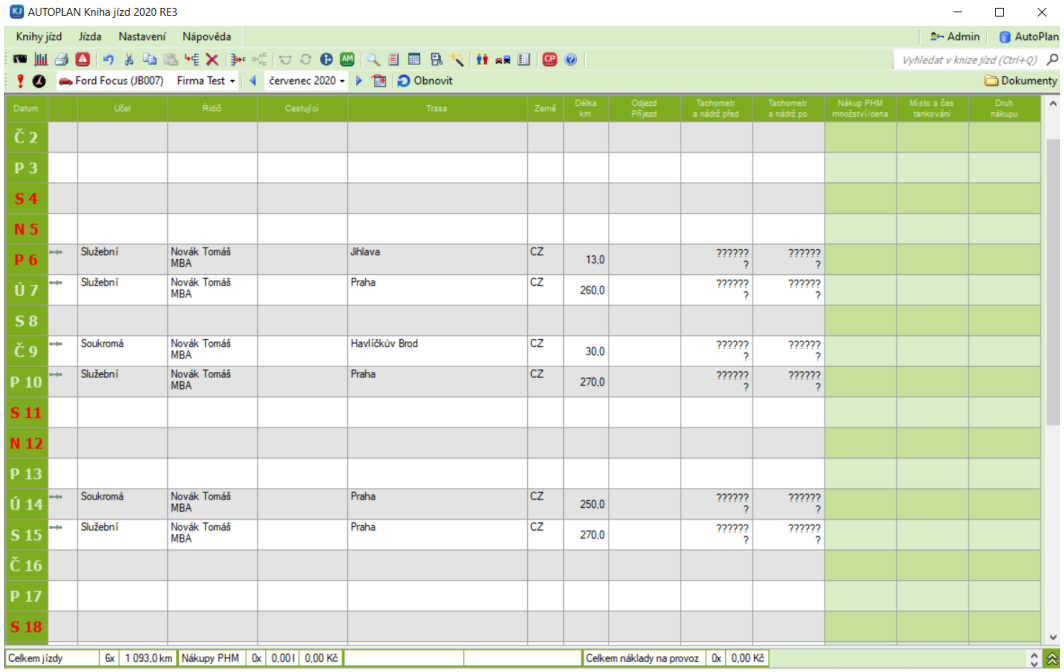
Aplikace umožňuje evidovat jednotlivé jízdy, které do systému vstupují buď ručním zadáním nebo importem jízd z GPS zařízení[4)].

Systém také poskytuje pokročilé funkce jako jsou například výpočet cestovních náhrad, vedení agendy silniční daně, tisk cestovního příkazu a mnoho dalších.

Toto řešení, jako jediné ze všech zkoumaných, není provozováno jako webová aplikace, nýbrž je instalována jako desktopová aplikace (instalovaná jako síťová aplikace či přímo na počítači uživatele) u zákazníka. Data jsou poté ukládána do databáze Microsoft Access nebo Microsoft SQL Server také přímo u zákazníka.

[3)] Informace získány při komunikaci s obchodním zástupcem po získání cenové nabídky.

[4)] Společnost vybízí ke kontaktu a ověření, zda používané GPS zařízení poskytuje vhodná data.



Obr. 2.4 Rozhraní aplikace AUTOPLAN Kniha jízd, zdroj vlastní

Aplikace AUTOPLAN Kniha jízd je zpoplatněna na základě počtu vozidel a počtu licencí. Také jsou dostupné různé rozšíření jako například import z GPS zařízení či anglická lokalizace. I cena za vozidla a licence je určována poměrově a na vyšší počty kusů je uplatňována množstevní sleva. Společnost má na svých stránkách pro tento případ online kalkulačku celkové ceny[4].

Pro modelový případ 30 vozidel se síťovou licenci pro 25 klientských počítačů a s anglickou lokalizací by byla výsledná částka dle aktuálního ceníku 20 375 Kč bez DPH.

Generátor knihy jízd

Aplikace Generátor knihy jízd (dále již jen GKJ) nenabízí monitorování tras v reálném čase, ale naopak zpětnou rekonstrukci tras na základě zadaných dat. Zákazník tedy vloží parametry auta, hodnoty stavu tachometru na začátku a konci období a doplní místa, kde auto v daném období mělo být, služební cesty a provedená tankování[5].

Aplikace poté následně dle průměrné spotřeby a velikosti nádrže vygeneruje cesty, které by se měly blížit či odpovídat skutečnosti. V případě některého z nesouladů - velké odchylky v najetých kilometrech, neodpovídající spotřebě paliva je následně zákazník upozorněn[5].

Obr. 2.5 Nastavení před vygenerováním cest v GKJ,vlastní

Dle oficiálního ceníku je Aplikace GKJ zpoplatněna 500 Kč za auto na rok bez DPH[5].

2.3 Porovnání současných řešení

Během získávání dat byly vyzkoušeny všechny zkoumané aplikace. Některé z nich nabízely velké množství funkcí, které přesahovaly a netýkaly se knihy jízd.

V tabulce 2.3 níže jsou porovnány zkoumané řešení dle definovaných sledovaných kritérií.

Tab. 2.3 Porovnání vlastností existujících řešení

	Typ	Sledování pozice	Vlastní provoz	Platba
GPS Dozor	Web/Mobilní	Ano	Ne	Pravidelná
AUTOPLAN KJ	Desktop	Ne	Ano	Jednorázová
GKJ	Webová	Ne	Ne	Pravidelná
AutoGPS	Web/Mobilní	Ne	Ne	Pravidelná

3 Legislativní omezení

V této kapitole budou prezentovány legislativní důvody proč vést knihu jízd a omezení, které se knih jízd týkají, případně se jich v budoucnu týkat mohou.

3.1 Nutnost vedení knihy jízd

Vést knihu jízd povinné není, ale v mnoha případech je to žádoucí a většinou je to pro majitele výhodné a pohodlné, jelikož ušetří na nákladech a má větší přehled o aktivitách svých případných zaměstnanců, což výrazně zjednodušuje administrativu[6].

V pokynu D-22 Generálního finančního ředitelství jsou mimo jiné vymezeny údaje, které by se měly a musí objevit v evidenci jízd[7].

3.1.1 Kniha jízd z pohledu daně z příjmu

Z hlediska daně z příjmu se musí vést kniha jízd, jestliže si podnikatel do nákladů uznává parkovné a náklady na PHM[8].

Může však také využít paušální výdaje, na maximální počet tří aut, které činí 5 000 Kč za vozidlo (60 00 Kč ročně) u právnických osob a 4 000 Kč (48 000 Kč ročně) u fyzických osob[8].

Právnická osoba
V případě uplatnění paušálních výdajů snížíte v daňovém přiznání výsledek hospodaření o 60 000 Kč a zvýšíte výsledek hospodaření o parkovné a PHM. Následně odpadá povinnost vést knihu jízd, pro ta vozidla, která byla takto v daňovém přiznání upravena[8].

Fyzická osoba
U fyzických osob probíhá podobná procedura, ale u vozidla, které podnikatel využívá pro soukromé účely, se sníží výsledek hospodaření o 48 000 Kč a zvýší se o parkovné a PHM. Navíc se u takového vozidla provede korekce 20% částky za opravy[8].

3.1.2 Kniha jízd z hlediska DPH

Z pohledu DPH musí podnikatel psát knihu jízd, pokud si chce na konci roku nárokovat odpočet DPH, a doložit ji na konci roku[8].

Právnická osoba
V případě nároku na odpočet DPH a nedoložení knihy jízd při případné daňové kontrole, by finanční úřad právnické osobě nepřiznal nárok na odpočet DPH, uložil pokutu

činící 20% částky DPH a úrok z prodlení do dne zaplacení naměřené částky ve výši 14% + REPO sazba za každý den zpoždění[8].

Fyzická osoba
Podobná pravidla platí i u fyzických osob, avšak u vozidla určenému k soukromým účelům si může nárokovat maximálně 80% z celkového DPH u daného vozidla[8].

3.1.3 Dodržování a dokazování bezpečnostních přestávek

Jako jeden z dalších důvodů pro vedení knihy jízd je dokazování bezpečnostních přestávek. Nejdříve bude třeba představit koho se to zejména týká, budeme definovat pojem "*řidič referent*".

Pojem "řidič referent"
Ačkoliv pojem "řidič referent"není jako pojem zakotven v legislativě České republiky, je obecně chápán jako osoba, která využívá v pracovní době vozidlo k přepravě osob, zboží a nebo jiného nákladu při plnění pracovních úkolů - tedy že řídí služební nebo vlastní vozidlo v rámci služební cesty[9].

Jak již bylo dříve zmíněno, řidiči referenti, kteří neřídí dopravní prostředek nad 3500 kg, na který se vztahují speciální předpisy, musí dle nařízení vlády č. 168/2002 Sb. dodržovat bezpečnostní přestávky:[10]

"U zaměstnance, který řídí dopravní prostředek a na kterého se nevztahuje zvláštní právní předpis, je zaměstnavatel povinen zajistit, aby nepřekročil maximální dobu řízení, která činí 4,5 hodiny; za dobu řízení se považuje i přerušení řízení na dobu kratší než 15 minut. Nejpozději po uplynutí maximální doby řízení musí být řízení přerušeno bezpečnostní přestávkou v trvání nejméně 30 minut, nenásleduje-li nepřetržitý odpočinek mezi dvěma směnami nebo nepřetržitý odpočinek v týdnu. Bezpečnostní přestávka může být rozdělena do dvou částí v trvání nejméně 15 minut zařazených do doby řízení"[10].

3.2 GDPR

Z důvodu uchovávání osobních údajů v knize jízd, je třeba se zmínit o nařízení o jejích ochraně, tedy o GDPR (General Data Protection Regulation)[11].

GDPR je obecné nařízení o ochraně osobních údajů, které představuje nový právní rámec ochrany osobních údajů v evropském prostoru. Má za cíl hájit práva občanů Evropské unie vůči neoprávněnému zacházení s jejich daty nebo s jejich osobními údaji. Nařízení se týká všech firem, institucí nebo služeb, které zpracovávají data uživatelů na evropském trhu [11].

GDPR není prvním nářízením o ochraně osobních údajů v Evropské unii. GDPR nahradilo směrnici 95/46/ES, která byla platná od roku 1995. GDPR bylo schváleno 27. dubna 2016 a vešlo v platnost 25. května 2018 [12].

Co je považováno GDPR za osobní údaje

Mezi obecné osobní údaje jsou řazeny následující informace[13]:

- jméno
- pohlaví
- datum narození
- osobní stav
- IP adresa
- fotografický záznam
- e-mailová adresa
- telefonní číslo.

Dále také existují osobní údaje, klasifikovány jako citlivé, které by mohly vést k diskriminaci subjektu, ať už ve společenském či osobním životě[14]:

- rasa, etnický původ
- náboženské vyznání
- politické názory
- zdravotní stav
- sexuální orientace
- záznam z trestního rejstříku.

V případě nedodržení nařízení GDPR hrozí pokuty až 20 000 000 eur, nebo až 4% z celkového ročního obratu firmy, vždy vyšší z obou částek. Možnou výši pokuty ovlivňují faktory jako závažnost porušování, délka porušování, míra škody, či počet zasažených občanů[15].

3.3 Projekt eCall

Z hlediska možného budoucího vývoje je třeba se zmínit i o projektu Evropské komise - eCall.

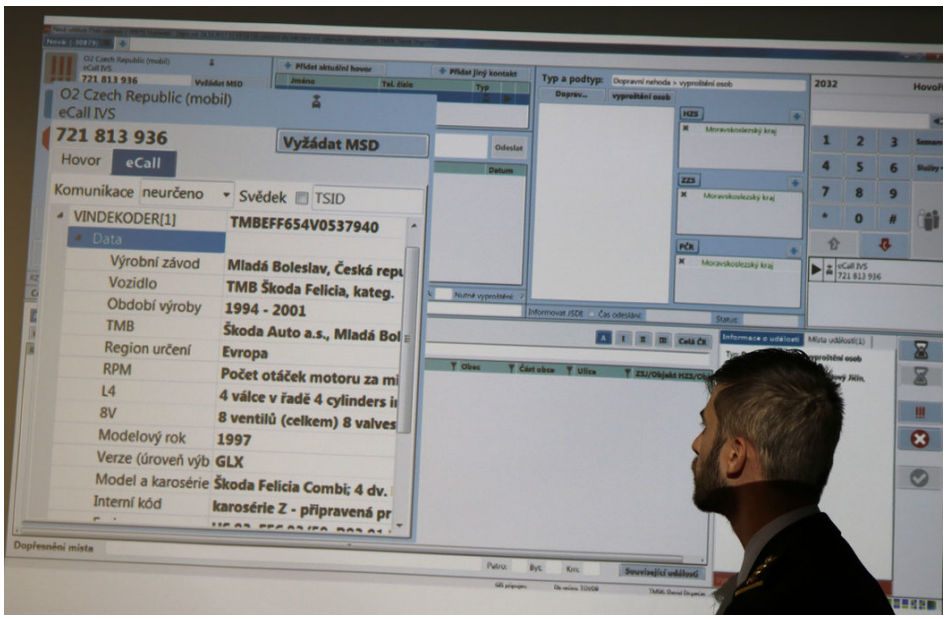
Projekt eCall je iniciativa Evropské komise, která má umožnit co nejrychlejší pomoc řidičům motorových vozidel, kteří se stali účastníky dopravní nehody kdekoliv na území Evropské unie[1)].

Evropská komise odhaduje, že hovor na telefonní linku jednotného evropského čísla tísňového volání (tísňová linka 112) uskutečněná pomocí systému eCall může urychlit odezvu záchranných složek až o 40% v městských oblastech a o 50% ve venkovských oblastech a může snížit počet obětí o nejméně 4% a počet těžkých úrazů o 6%. Tyto odhadované statistiky byly uváděny za hlavní důvod zavádění systému.

Bylo rozhodnuto dne 15. května 2014 nařízením č. 585/2014/EU o povinnosti zavést služby eCall v celé Evropské unii do 30. září 2017 všem členským státům Evropské unie. Od 1. dubna 2018 poté je povinnost, aby každý nový automobil prodaný v Evropské unii obsahoval systém eCall.

I mimo Evropskou unii jsou zaváděny podobné programy - například v Ruské federaci je implementován systém ERA-GLONASS či ve Spojených státech amerických systém AACN (Advanced Automatic Collision Notification).

Na obrázku 3.1 níže lze vidět rozhraní systému, které je dostupné operátorům tísňového volání.



Obr. 3.1 Rozhraní systému eCall, zdroj [17]

[1)] Také země mimo EU se zapojily - například Turecko[16]

Zařízení také současně částečně plní funkci černé skříňky, ukládá totiž diagnostické údaje vozidla za posledních několik sekund pohybu vozu, nicméně nelze považovat za plnohodnotný nástroj pro záznam jízdy. Systém však obsahuje neustále aktivní GPS a mikrofon, který může být dálkově aktivován[18].

Hovor na tísňovou linku 112 ze systému eCall je vyvolán buď automaticky po aktivaci bezpečnostních senzorů ve vozidle nebo po manuálním stisku nouzového tlačítka ve vozidle. Hovor je automaticky přesměrován na lokální záchrannou službu v EU. Systém by měl rozpoznat, že hovor je veden ze zařízení eCall a měl by mít prioritu oproti běžným hovorům[18].

Během tísňového volání je navázáno kromě hlasového spojení i datové a je odeslán i tzv. minimální soubor dat (*MSD - Minimum Set of Data*). Tento soubor dat zahrnuje mimo jiné časovou značku, polohu dle GNSS, směr cesty vozidla, identifikační číslo vozidla (VIN) a další informace. Tento minimální soubor dat je plně definován v technické normě ČSN EN 15722[19].

Je třeba také zmínit projekt HeERO (Harmonised eCall European Pilot Project), který na evropské úrovni připravoval, realizoval a koordinoval pomocí pilotních projektů fázi zavádění eCall do provozu. V rámci tohoto projektu byl připraven pro každou zemi zaváděcí plán pilotního projektu, který poskytoval návod a postupy, jak se má služba eCall implementovat a testovat. Projekt HeERO zahrnoval v první fázi národní a mezinárodní pilotní projekty v Německu, Itálii, Řecku, Rumunsku, České republice, Finsku, Chorvatsku, Nizozemí a Švédsku. Také probíhalo přeshraniční testování kompatibility s ruským[2)] systémem ERA-GLONASS[21].

Projekt probíhal ve dvou fázích - fáze HeERO 1 v letech 2011-2013, kdy se účastnily výše zmíněné státy a fáze HeERO 2 v letech 2013 - 2014, kdy se přidalo šest dalších zemí - Belgie, Bulharsko, Dánsko, Lucembursko, Španělsko a Turecko. Projekt byl částečně financován Evropskou komisí pod ICT-PSP (Information and Communication Technologies Policy Support Programme)[21].

Ačkoliv, zatím není projekt eCall a jeho technologie určen k monitorování pohybu motoristů, je možné očekávat, že po dostatečném zavedení vozidel s technologiemi eCall v populaci budou informace ze zařízení eCall využívány pro dokazování údajů pohybu, případně jejich evidenci pro pozdější zpracování.

Také lze předpokládat, že v budoucnu, v případě otevřeného API technologie eCall nebo některé z podobných technologií, které budou implementované v automobilech, již může být jiná situace. Nebude tedy třeba instalace dodatečných fyzických zařízení pro sledování polohy vozidel, jaké používají některé služby pro vedení evidence knihy

[2)] Původem z Ruské federace, funguje ve státech Euroasijského ekonomického svazu - Ruské federaci, Bělorusku, Kazachstánu, Arménii a Kyrgyzstánu[20]

jízd, ale bude možné využít právě tyto nové technologie.

3.4 Zákaz uchovávání informací u třetích zemí či stran

Je nutné zmínit legislativní omezení, které se v nedávné době začaly týkat uchovávání osobních dat uživatelů. Jedná se o omezení umístění (fyzického místa), kde jsou data státních příslušníků sbírány, zpracovány a ukládány. V některých případech se to týká plošného zákazu přenosu dat do zahraničí (např. Ruská federace), v jiných případech je to zakázáno do doby, než k tomu uživatel dá svolení či je o tom informován (záleží na místních zákonech).

Liší se také typ informací, se kterými je v daných zemích zákázáno pracovat mimo státní území. Například v Číně se to týká určitých osobních a finančních dat uživatelů, v Ruské federaci se jedná o všechny osobní data uživatelů, v Indonésii musí být přímo v zemi umístěna datacentra spracující veřejné služby a v Austrálii se jedná o zdravotní záznamy občanů [22] [23] [24].

Tato problematika se také začala více probírat, dnes v době cloudových řešení, kdy si ani samotné společnosti, pronajímající si určité místo v cloudových úložištích, nemohou být jisty, kde jsou daná data fyzicky umístěna. Díky principu *geo-redundantních* řešení mohou být stejné informace fyzicky uložené například v několika různých a vzdálených místech, kdyby některé z nich z jakéhokoli důvodu nebylo dostupné.

Pokud některé služby nemohou garantovat, kde se budou data uživatelů fyzicky nacházet, mohou tedy dostat zákaz operovat na národním trhu zmíněných zemí, případně mohou riskovat pokutu[25].

V samotné Evropské unii je aktuálně vysoce preferován opačný trend - v roce 2018 byl diskutován a rychle přijat zákon, který zakázal v jakékoli zemi omezení týkající se umístění údajů[26][27].

Také některé společnosti mohou pomocí interních směrnic upravovat práva na zpracovávání či umístění dat, které nechce sdílet s třetími stranami.

4 Vhodné technologie pro tvorbu systému

V této části budou vyjmenované a popsané technologie, které byly zvoleny jako vhodné pro tvorbu systému evidence elektronické knihy jízd. Také budou zmíněné alternativy ke zmíněným technologiím a uvedeny důvody, proč byly zvolené právě tyto technologie.

4.1 Serverová aplikace

Charakteristiky serveru

- **Pasivní** - Server samotný neinicializuje žádnou činnost vůči klientům.
- **Naslouchání** - Server naslouchá na síti a reaguje na požadavky klientů.
- **Obsluha požadavků** - V případě, že požadavek přijde na server, obslouží ho a vrátí klientovi odpověď.

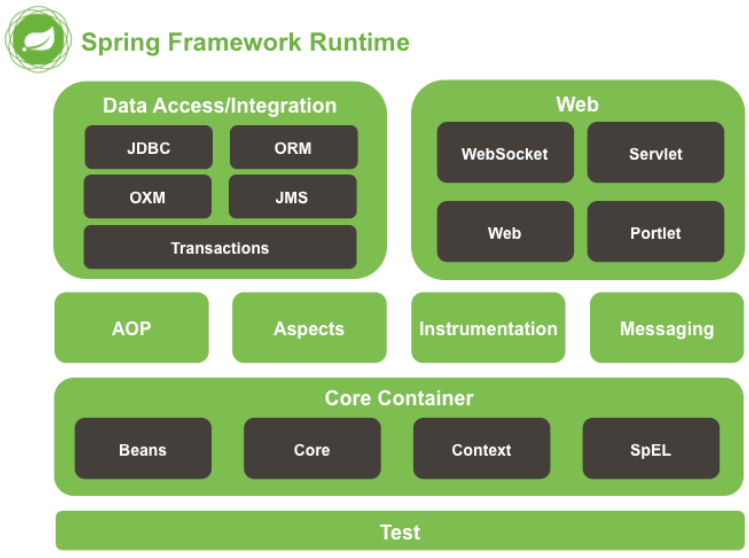
4.1.1 Spring Framework

Jelikož je nyní popisována serverová aplikace, je nutné se zmínit o Spring Frameworku, který bude v rámci této diplomové práce použitý na její tvorbu.

Spring Framework patří mezi nejoblíbenější a nejpoužívanější aplikační frameworky pro vývoj webových aplikací na platformě Java EE [28].

Jednotlivé funkcionality Spring Frameworku jsou rozděleny do jednotlivých částí modulární architektury, zobrazeno na obrázku 4.1.

Spring umožňuje využít pouze moduly, které vývojář sám potřebuje.



Obr. 4.1 Přehled Spring modulů, zdroj vlastní

Mezi cíle frameworku patří zejména zjednodušit a umožnit více produktivní vývoj tzv. enterprise aplikací. Je postaven na principech DI (Dependency Injection) a AOP (Aspect-Oriented Programming)[29].

Dependency Injection je technika pro usnadnění předávání závislostí s potřebnými funkcionalitami mezi sebou bez toho, aby na sebe měly v době kompilace odkaz.

Aspect-Oriented Programming se také týká modularity programu a dobré praxe odebrání opakujících se částí funkcionalit na několika místech. Tyto opakující části mohou být soustředěny do tzv. *aspektů*. Tyto aspekty se poté mohou navázat na potřebnou třídu.

Také existuje platforma Spring Boot, která používá jako základ Spring Framework. Implementuje veškeré funkce Springu, používá defaultní nastavení a slouží k rychlému vývoji a nasazení aplikace.

Alternativy:
V případě volby využití jazyka C# by bylo možné použít například *ASP.NET*.

4.1.2 Apache Maven

Apache Maven (dále jen Maven) je popisován jako nástroj pro správu projektu, řízení a automatizaci sestavení aplikací. Také se stará o správu závislostí.

Maven je založený na konceptu *POM - Project Object Model*. Jedná se o XML reprezentaci Maven projektu v souboru nazvaném *pom.xml*. POM obsahuje všechny nezbytné informace o projektu - jeho konfiguraci, seznam závislostí včetně čísel jejich verzí.

4.1.3 WildFly

WildFly (dříve známý jako JBoss) je aplikační server vytvořený původně společností JBoss, nyní spravovaný společností RedHat, která zakoupila společnost JBoss v roce 2006 a produkt v roce 2014 přejmenovala[30].

Aplikační server Wildfly je napsaný v jazyce Java a implementuje JavaEE specifikovaci. Není vázán na jedinou platformu a je možné ho provozovat na různých operačních systémech.

Tento aplikační server je v systému použitý pro běh serverové aplikace i pro běh GIS aplikace.

Alternativy:
Pro účely projektu by bylo možné použít v podstatě jakýkoli aplikační server, který je kompatibilní s JavaEE verze 8. Mohl by to být například *GlassFish*, *Payara* či *Tomcat*.

4.1.4 Hibernate ORM

Hibernate ORM je knihovna pro objektově-relační mapování v jazyce Java, ačkoliv je snadné ji používat i například s jazykem Kotlin. Knihovna umožňuje mapování objektů do relačního databázového systému[1)], tedy výrazně zjednodušuje proces převodu tříd na jednotlivé databázové tabulky.

Alternativy:
Jako alternativu k Hibernate ORM lze zmínit například ActiveJDBC.

4.1.5 REST

REST (Representational State Transfer) je návrh architektury rozhraní [31] a webové služby, které jsou na principu REST založeny jsou označovány jako *RESTful*. Tyto webové služby pak mohou operovat s předem definovanou sadou operací převzatých z HTTP protokolu, tedy s operacemi GET, POST atd (viz tabulka **??** níže).

JSON
JSON (JavaScript Object Notation) je jedním ze způsobu formátu zápisu dat objektů. Formát používá dvojité uvozovky pro definování řetězců, hranaté závorky pro definování polí (listů) a složené závorky pro popsání objektů (kolekce párů klíč-hodnota). Formát "JSON může reprezentovat čtyři primitivní typy - řetězce, čísla, hodnoty boolean a null a dva strukturované typy - objekty a pole"[32].

Níže jsou pro představu zobrazena data jednoduchého objektu reprezentována ve formě JSON, kde jsou zmíněná pravidla dodržena:

```
{"vehicle": {
  "name": "Toyota Corolla",
  "spz": "2E75634",
  "color": "dark blue",
  "nextSTK": "2021-05-12",
  "data": {
    "gasStops": [
      {"date": "2020-04-11", "price": "1100"},
      {"date": "2020-04-27", "price": "1600"}
    ],
    "paths": []
  }
}}
```

V rámci systému elektronické knihy jízd je tento formát použitý při výměně dat mezi klientskými aplikacemi (webovou a mobilní) a serverovou aplikací a také při získávání dat z GIS standardem WFS.

[1)] S databázemi typu NoSQL lze alternativně pracovat s knihovnou Hibernate OGM.

Alternativy:
Alternativně by bylo možné použít pro komunikaci protokol SOAP, který využívá jako přenosový formát XML, nicméně je (oproti rozhraní REST a jeho nejpoužívanějšímu formátu JSON) komplikovanější a náročnější na datové i výpočetní prostředky.

4.2 Klientská aplikace

Klientská webová aplikace je poskytovaná uživatelům přes počítačovou síť z webového serveru, tedy není třeba ji ukládat do paměti zařízení. Díky tomu je jednodušší dostupnost aplikace pro cílové uživatele - není dostupná pouze na specifickém počítači, ale lze na ni přistoupit z jakéhokoli prohlížeče z jakéhokoli počítače, který má přístup do počítačové sítě s danou aplikací. Mezi její výhody také patří schopnost pracovat bez ohledu na operační systém koncového uživatele.

4.2.1 Angular

Angular je open-source framework pro webové aplikace založený na jazyce Typescript[2)]. Framework je vyvíjen týmem ze společnosti Google [33].

Angular lze využít pro různé účely - vývoj webových, mobilních, nativních mobilních aplikací i nativních aplikací na desktop. Také ho lze využít pro vývoj progresivních webových aplikací (PWA).

Tento webový framework byl navržen pro jednoduché a intuitivní oddělení zobrazovací logiky od aplikační logiky na pozadí.

Angular vs. AngularJS
Je důležité rozlišovat mezi dvěmi generacemi frameworku Angular. První generace frameworku byla vydána v roce 2009 a používala jazyk Javascript. Druhá generace poté byla vydána v roce 2014 a byla nazvána Angular 2. Pro větší přehlednost byla původní verze Angular 1 přejmenována na AngularJS a nyní se tak označuje jakákoli verze Angularu 1.x, který je i nadále podporován[3)].

Druhá generace frameworku - nyní běžně označován jako Angular 2+ nebo pouze Angular je nadále podporována a nyní je aktuálně ve verzi 10, příčemž nová *major* verze je vydávána každých šest měsíců s šesti měsíci aktivní podpory[4)] a poté s dalšími dvanácti měsíci dlouhodobé podpory[5)][36].

[2)] Typescript - nadmnožina Javascriptu, vyvíjený společností Microsoft.
[3)] Aktuální verze AngularJS je 1.7.9 a bude podporována až do 31.června 2021, kdy přestane AngularJS dostávat aktualizace i opravy[34][35]
[4)] Aktivní podpora - pravidelně plánované a vydávané aktualizace a opravy
[5)] Dlouhodobá podpora (LTS) - oprava pouze kritických a bezpečnostních oprav

Aplikace napsané ve frameworku AngularJS a Angular spolu kompatibilní nejsou[6], ale poté i v rámci nové generace Angular (2+) jsou často mezi jednotlivými verzemi určité rozdíly, které neumožňují úplnou zpětnou nebo dopřednou kompatibilitu [7].

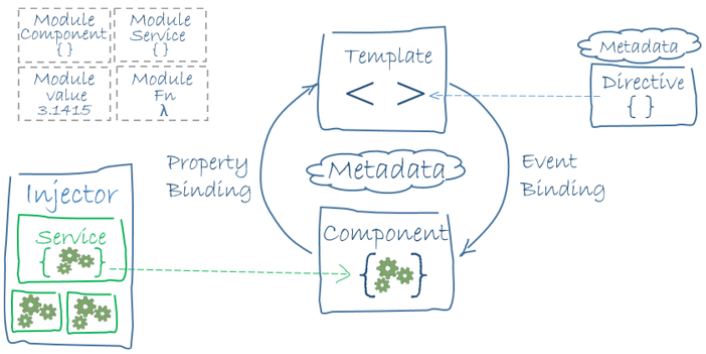
Koncepty frameworku

Jako základní kameny aplikace Angular lze označit *moduly(NgModules)*, které sdružují a poskytují komponentám vlastní *kontext kompilace*[8] a reprezentují tedy funkční jednotky aplikace. Části modulů lze sdílet a exportovat a poté vkládat do jiných modulů.

Architektura frameworku Angular sestává primárně z hiearchické struktury komponent. Moduly, komponenty a direktivy jsou třídy, které využívají *dekorátory*. Ty specifikují výsledný typ a metadata.

Jednou z důležitých součástí frameworku je i *routing*. Je implementovaný díky službě *Angular Router* a umožňuje přecházet mezi definovanými komponentami (pohledy) aplikace. Router mapuje URL v prohlížeči a na základě jejího obsahu je rozhodnuto o zobrazeném obsahu v *router outlet* - tedy se mění část či celý obsah stránky viditelný uživatelem. Router poskytuje i několik dalších užitečných mechanismů - např. ochranu proti neoprávněnému přístupu, defaultní cesty, přesměrování nebo i chránění proti přístupu na neexistující stránku. Také zjednodušuje práci s předáváním hodnot pomocí parametrů v URL cestě[33].

Důležitou součástí je i *"líné načítání"(lazy loading)*, což dokáže optimalizovat výkon aplikace, případně lze části aplikace i načíst dopředu na pozadí.



Obr. 4.2 Vztah mezi součástmi, převzato z [39]

[6] Oficiální stránky frameworku poskytují příručku, jak zdrojový kód transformovat na novou verzi[37].

[7] Na oficiální stránce Angular.io jsou dostupné informace o plánovaných změnách a odebraných vlastnostech[38]

[8] Původně nazváno "compilation context"- způsob sdružení TypeScript souborů, které pak jsou samostatně analyzovány, zda se dovedou jako celek zkompilovat.

Alternativy:
Jako alternativu lze zmínit zejména knihovny React nebo Vue.js. React je JavaScriptová knihovna od společnosti Facebook a Vue.js je nezávisle vyvíjený open-source framework. V době procesu tvorby diplomové práce je knihovna React i framework Vue.js stále více populárnější než framework Angular[40].

4.2.2 Apache HTTP Server

Apache HTTP server je open-source webový server, nezávislý na cílové platformě. Je vyvíjený od roku 1995 a nyní spravovaný neziskovou organizací Apache Software Foundation. Samotný HTTP Server Apache je licencovaný pod Apache License v2.0, která je kompatibilní s GPL. Téměř od svého začátku patří tento webový server mezi jeden z nejpoužívanějších, v roce 2020 byl podle odhadu používán pro obsluhu zhruba 24,24% stránek [41][42].

Mezi jeho hlavní přednosti patří:

- spolehlivost
- výkonnost
- otevřené zdrojové kódy
- široká rozšířenost
- vysoká přizpůsobitelnost

Také je třeba zmínit o XAMPP, balíček volně dostupného otevřeného software - jedná se o Apache distribuci, která obsahuje kromě něj i MariaDB, PHP a Perl. Jeho použití je jednoduchý způsob jak vytvořit lokální webový server. XAMPP také nabízí k instalaci přídavné moduly - například redakční systémy či MediaWiki [43].

Tento webový server je v systému využitý pro vyřizování požadavků na získání zdrojů webové aplikace. Byl zvolen zejména z jeho rozšířenosti a tedy obecné znalosti, ale v rámci systému je to nejjednodušeji nahraditelná část.

Alternativy:
Lze zmínit jako alternativu například webový server *Nginx* nebo *lighttpd*.

4.3 Mobilní aplikace

Za mobilní aplikaci je považován jakýkoli program, který je primárně určen pro mobilní zařízení (například smartphone nebo tablet). Požadavky kladené na takové aplikace se

ve velké míře odlišují od desktopových aplikací, zejména z důvodu rozdílného ovládání a menších rozměrů cílových zařízení.

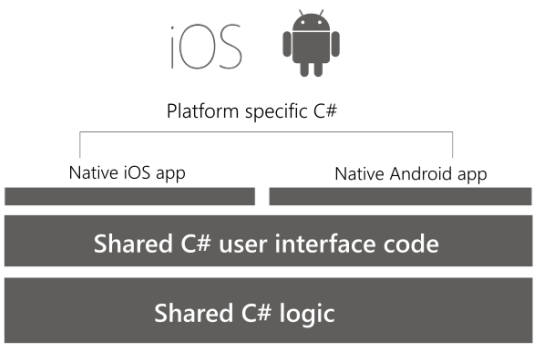
Dále jsou popsány prostředky, které budou využívány pro vývoj mobilní aplikace v rámci této diplomové práce.

4.3.1 Xamarin framework

Xamarin je open-source platforma umožňující vývoj pro mobilní zařízení v jazyce C na platformě .NET. Xamarin rozšiřuje vývojářskou platformu. NET dalšími nástroji a knihovnami, specificky pro vytváření aplikací pro platformy Android, iOS a Windows.

Aplikace vytvořené pomocí frameworku Xamarin poté vypadají a chovají se nativně[9] na každé z platforem. Také mají nativní přístup k rozhraním API daných platforem a používají hardwarovou akceleraci specifickou pro danou platformu. [44].

Sdílení kódu kódu mezi platformami ukazuje následující obrázek (Obr. 4.3).



Obr. 4.3 Sdílení kódu mezi platformami,
zdroj [45]

Dokumentace Xamarinu zmiňuje, že průměrně lze sdílet až 80-90% zdrojového kódu aplikace napříč platformami. To umožňuje vývojářům psát většinu aplikační logiky v jednotném jazyce, ale i tak na různých plaformách dosahovat nativního výkonu a pocitu při používání[46].

Xamarin je postavený na platformě Mono, což je open-source verze .NET Frameworku.

Výhody Xamarinu

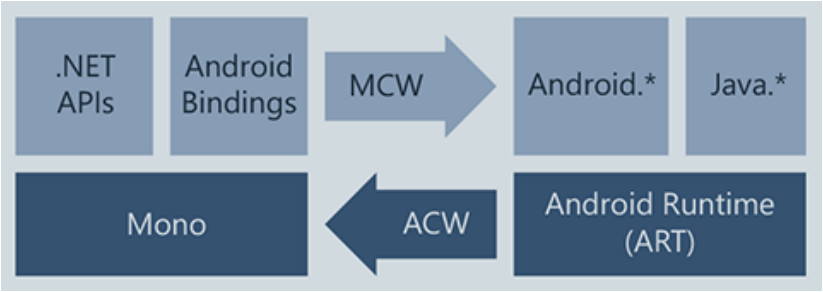
- **Začlenění jazyků Objective-C, Java, C a C++** - Xamarin poskytuje způsob, jak přímo volat knihovny jazyků Objective-C, Java, C a C++, což dává možnosti použít velké množství kódu třetích stran[46].

[9] Tedy uživatel nepozná rozdíl od standardního chování či vzhledu dané platformy.

- **Moderní jazykové konstrukce** - Aplikace pro Xamarin jsou napsány v jazyce C#, který poskytuje některé výhody oproti Objective-C a Java, jako například paralelní programování nebo LINQ[46].
- **Podpora více mobilních platforem** - Xamarin poskytuje podporu pro dvě hlavní platformy Android a iOS a aplikace pohou sdílet velké množství svého kódu. Xamarin.Essentials také nabízí unifikované API pro přístup k nejpoužívanějším funkcím na obou platformách. Sdílení kódu pak může citelně snížit náklady i čas na vývoj aplikace pro obě platformy[46].

Xamarin.Android

Aplikace Xamarin.Android jsou kompilované z jazyka C# do *mezijazyka* (intermediate language), který je následně Just-in-Time kompilován do nativního *jazyka symbolických adres* ve chvíli, kdy je aplikace spuštěna.



Obr. 4.4 Xamarin.Android, zdroj [46]

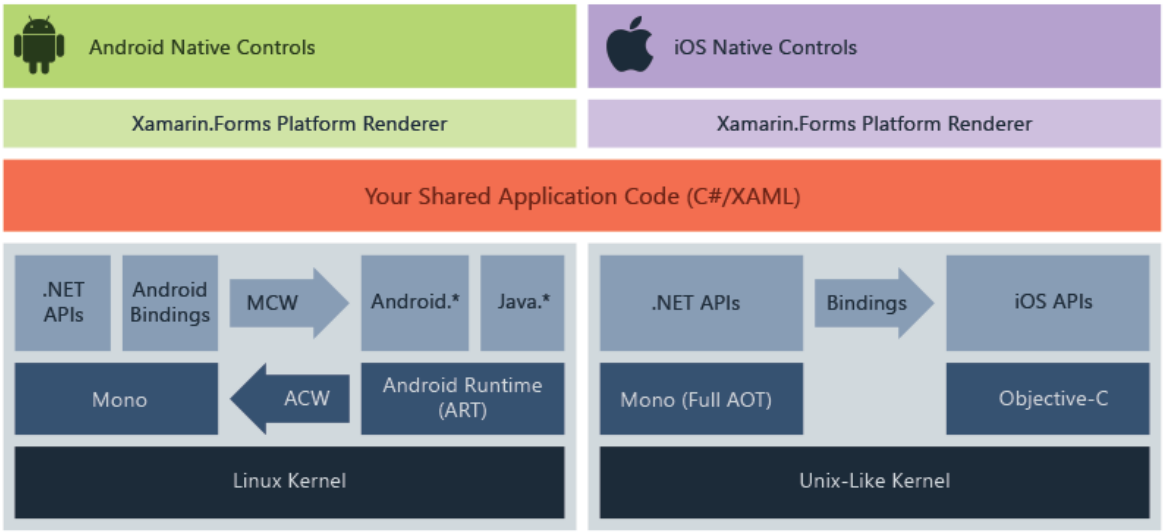
Platforma Xamarin byla založena v roce 2011 a v roce 2016 byla odkoupena společností Microsoft. Xamarin je dostupný jako rozšíření do vývojového prostředí Microsoft Visual Studio pod názvem Visual Studio Tools for Xamarin.[46]

4.3.2 Xamarin.Forms

Xamarin.Forms je framework umožňující uživatelům vyvíjet aplikace pro všechny platformy najednou. Xamarin.Forms je určený pro rychlou tvorbu GUI. Pomocí komponent definujeme GUI a za běhu aplikace jsou modely zmapovány na nativní komponenty každé platformy. Samotný Xamarin.Forms nijak neovlivňuje výsledný vzhled aplikace[44][47].

4.3.3 Model-View-ViewModel

Model-View-ViewModel (MVVM) je návrhový vzor, vhodný pro využití mimo jiné v Xamarin Forms, umožňující oddělení logiky aplikace od uživatelského rozhraní. Díky vyčlenění kódu od popisu uživatelského rozhraní je kód obecně přehlednější a logika aplikace je nezávislá na podobě uživatelského rozhraní[48].

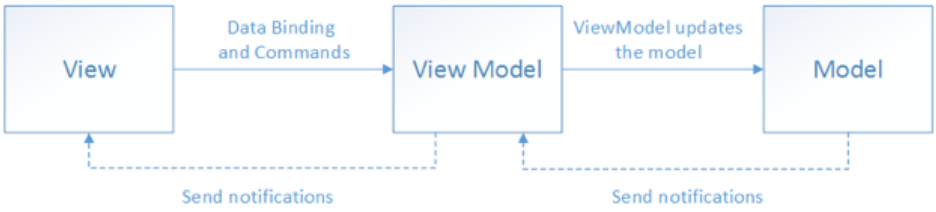


Obr. 4.5 Jak funguje Xamarin.Forms, zdroj [46]

Vzor MVVM obsahuje tři základní komponenty: Model, View a ViewModel. Na obrázku 4.6 je vidět vztah mezi těmito třemi součástmi.

Součásti vzoru MVVM:

- **Model** - nevizuální třídy, které popisují aplikační data,
- **View** - reprezentace uživatelského rozhraní v jazyce XAML, může se jednat o okno aplikace, navigační panel, stránku, formulář nebo ovládací prvek,
- **ViewModel** - spojuje Model a View a udržuje stav aplikace



Obr. 4.6 Vrstvy a vztahy v MVVM, zdroj [48]

Výhody použití vzoru MVVM jsou následující:

- Lze vytvářet *unit testy*[10)] pro ViewModel bez potřeby uživatelského rozhraní.
- Návrháři UI a vývojáři mohou pracovat nezávisle na sobě během procesu vývoje.
- Lze měnit uživatelské rozhraní aplikace (View) bez zásahů do kódu, pokud bylo implementováno pouze v jazyce XAML.

[10)] Unit testy - automatické ověření správnosti funkcionality implementace samostatné/dílčí části systému.

4.3.4 XAML

Jazyk XAML (eXtensible Application Markup Language) je založený na značkovacím jazyce XML vytvořený společností Microsoft a používaný pro definování uživatelského rozhraní ve frameworcích jako je například WPF (Windows Presentation Foundation), UWP (Universal Windows Platform) a Xamarin.Forms [49].

Každý dokument jazyka XAML je tedy i XML dokument, který obsahuje pouze jeden kořenový element a vnořené elementy potomků. Takový dokument musí dodržovat strukturu podle specifikace 2009 XAML a obsahuje různé elementy, atributy a jmenné prostory.

Zjednodušený příklad níže:

```
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:d="http://xamarin.com/schemas/2014/forms/design"
    mc:Ignorable="d"
    x:Class="LogBook.Views.LoginPage"
    BackgroundColor="#fcfcfc">
  <ContentPage.Content>
    <StackLayout Padding="30,0" VerticalOptions="Center"
       HorizontalOptions="Center">
      <Image Source="login\_logo.png"/>
      <Entry x:Name="login" Placeholder="email"/>
      <Entry x:Name="password" IsPassword="true"/>
      <Button Clicked="LoginUser" Text="Login"/>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Jazyk XAML je v této diplomové práci použitý pro definování uživatelského rozhraní - stránek (*views*) ve frameworku Xamarin Forms.

4.4 NuGet balíčky

V případě vytváření, sdílení, a používání užitečného kódu veřejně mezi vývojáři lze použít mnoho způsobů - v případě jazyka *JavaScript* je možné využít *npm* či *Yarn*, pro jazyk *Python* existuje *pip* a pro platformu .NET a Xamarin lze použít balíčkovací systém *NuGet*. NuGet je open-source balíčkovací systém využívaný ve vývojářských platformách Microsoftu.

Systém samotný zjednodušuje hledání balíčků (např. ve vývojovém prostředí Visual Studio lze vyhledávat z nakonfigurovaných dostupných zdrojů, defaultně z veřejného hostitele nuget.org), stažení jejich správné verze a nakonfigurování do projektu. NuGet samotný také hlídá a sám instaluje potřebné závislosti.

Mobilní aplikace vytvářená v rámci této diplomové práce využívá několik balíčků poskytovaných tímto způsobem.

5 Mapy

Značná část této diplomové práce se týká využívání nebo vytváření mapových podkladů, a proto je důležité některé termíny přiblížit čtenáři. Zejména bude popsána tématika mapových serverů a způsoby vytváření a získávání mapových podkladů.

5.1 Pojem mapa

Mapa se dá definovat jako zobrazení určité části zemského povrchu, či vesmírných těles. Existuje mnoho druhů map (např. politické, topografické, obecně zeměpisné, hospodářské)[50].

Dále se mapy dělí podle rozsahu měřítka. Měřítko znázorňuje poměr reálné vzdálenosti ku vzdálenosti na mapě (např. měřítko 1:100 000 - 1 cm na mapě se rovná 100 000 cm ve skutečnosti)[50].

- **mapy malého měřítka** - měřítko větší než 1:1 000 000,
- **mapy středního měřítka** - měřítko 1:200 000 - 1:1 000 000,
- **mapy velkého měřítka** - měřítko menší než 1: 200 000[50].

V rámci této diplomové práce se obecně týkají zejména map malého měřítka - služební cesty mohou být velmi dlouhé. Systém nicméně umožňuje zobrazování mapy s různým měřítkem.

5.2 Mapové servery

Mapové servery jsou používány pro publikování prostorových dat a pro jejich snadné poskytování v uživatelsky přívětivé formě.

5.2.1 Architektura mapových serverů

Mapové servery pracují na principu klient-server, kdy server obsahuje mapové podklady a na vyžádání jednotlivých klientů je poskytuje. Vztah klient-server ale také platí v případě, kdy klient publikuje na mapový server prostorová data.

5.2.2 Shapefile

Shapefile je datový formát pro ukládání prostorových dat. Je vyvíjen a udržován firmou Esri, nicméně je považován za otevřený formát pro přenos dat mezi různými GIS aplikacemi [51].

V tomto Shapefile formátu jsou uložená data jako geometrické primitivní geometrické tvary - čáry, body, polygony.

Soubor typu Shapefile je v této práci použit jako způsob přenosu dat na GeoServer.

5.2.3 GeoServer

GeoServer je mapový server vyvíjený open-source a naprogramovaný v jazyce Java. Umožňuje upravovat a sdílet prostorová data. GeoServer vznikl v roce 2001 díky neziskové organizaci The Open Planning Project. Aktuální stabilní verze v březnu 2020 je 2.16.2 [52].

Jednou z důležitých součástí je webové administrační rozhraní, pomocí kterého lze přidávat, upravovat a sdílet data.

Data jsou sdílena ve vrstvách (layer) a sdružována ve "skladech"(store) a "pracovních prostorech"(workspace)[53].

Klíčovou vlastností GeoServeru je interoperabilita - projekt dodržuje standardy organizace Open Geospatial Consortium (OGC) a podporuje například WMS, WFS nebo WCS. Dále umožňuje pracovat s daty z různých databázových systémů a podporuje množství vektorových a rastrových datových formátů[53].

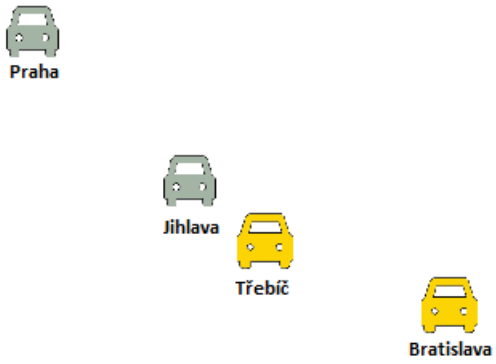
WMS - Web Map Service

WMS je nejpopulárnější standard OGC. Tato služba jako výstup poskytuje klientovi data ve formě rasterových obrázků nazývaných *dlaždice (tiles)*. Převážně se jedná o data ve formátu PNG, GIF nebo JPEG. Data jsou poskytována přes jednoduché HTTP rozhraní. Jednotlivé vrstvy získávané pomocí WMS lze libovolně kombinovat a překrývat [56].

Klient posílá u WMS parametry požadavku - specifikaci formátu výstupního souboru, jeho průhlednost, o jakou vrstvu se jedná a jaká oblast je požadována.

```
https://ADRESA/geoserver/logbook-prod/wms?service=WMS\
&version=1.1.0\&request=GetMap\&layers=current\%3Alogbook\_STATIONARY\
&bbox=14.452446\%2C48.144671\%2C17.129144\%2C50.061654\
&width=768\&height=550\&srs=EPSG\%3A4326\&format=application/openlayers
```

V rámci požadavku je také možné si všimnout o jakou vrstvu bylo žádáno (logbook_stationary), o jak velkou oblast dat se jedná (souřadnice v parametru bbox) a další zajímavé parametry - formát, projekce atd.



Obr. 5.1 Příklad výstupu WMS požadavku,zdroj vlastní

V tomto případě je získán z GeoServeru obrázek 5.1 s průhledným pozadím na kterém jsou zobrazeny body zájmu dle specifikovaného stylu. Důvod získávání obrázků s průhledným pozadím je kvůli zobrazování dat nad mapovými podklady.

WFS - Web Feature Service

WFS je další ze standardů OGC, který popisuje rozhraní pro přenos geografických vektorových dat mezi mapovými servery a klienty. Na rozdíl od WMS není výsledkem dotazu rastrový obrázek, ale soubor ve formátu XML s geografickými informacemi ve formě GML - Geography Markup Language.

Příklad dotazu a příklad odpovědi protokolu WFS:

```
https://ADRESA/geoserver/logbook-prod/ows?service=WFS
\&version=1.0.0\&request=GetFeature\&typeName=logbook-prod
\logbook\_STATIONARY\&maxFeatures=50
```

V rámci požadavku je také možné si všimnout o jakou vrstvu bylo žádáno (logbook_stationary), o jaký typ požadavku se jedná (GetFeature) a také, že je žádáno pouze o padesát zájmových bodů (maxFeatures=50).

Ukázka odpovědi níže je z důvodu stručnosti zjednodušená.

```
<wfs:FeatureCollection xmlns="http://www.opengis.net/wfs"
    xmlns:wfs="http://www.opengis.net/wfs"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:logbook_dev="http://logbook_dev">
     <gml:featureMember>
         <logbook_dev:currentPosition fid="currentPosition.1">
             <logbook_dev:the_geom>
                 <gml:Point>
                     <gml:cdnt>15.778,49.2586,0</gml:cdnt>
                 </gml:Point>
             </logbook_dev:the_geom>
             <logbook:vehicleId>e71f18e0</logbook:vehicleId>
             <logbook:vehicleSPZ>2J6-4196</logbook:vehicleSPZ>
             <logbook:driverName>Prost Al</logbook:driverName>
             <logbook:driverId>c877bd3</logbook:driverId>
             <logbook:onPath>false</logbook:onPath>
         </logbook:currentPosition>
     </gml:featureMember>
</wfs:FeatureCollection>
```

Stylování vrstev

Prostorová data samotná nemají žádné vizuální stylování a při přístupu k datům službou WMS jsou reprezentována pouze základními geometrickými tvary. Aby bylo možné data lépe zobrazit, je třeba je stylovat - např. specifikovat barvu, tloušťku nebo i zvolit obrázky, které budou některé z geografických informací reprezentovat.

Pro stylování se používá na GeoServeru značkovací jazyk založený na XML nazvaný *Styled Layer Descriptor* (SLD). Geoserver používá SLD jako primární jazyk pro stylování dat.

Daný existující styl (dokument, popisující stylování geografických dat) se poté vztahuje na vrstvy a specifikuje, jak mají být renderovány při požadavku na jejich data. V případě úpravy stylu jsou tedy data stylována dle aktuální preference.

Geoserver poskytuje ve vektorových datech tři typy tvarů - body, čáry a polygony. Čáry, které jsou definovány počátečním a koncovým bodem, lze stylovat nejjednodušeji, je možné jim nastavit pouze hranu (anglicky "edge"), zatímco polygonům (mnohoúhelníkům), které jsou definovány svými okrajovými body lze stylovat hranu a výplň. U bodů, které jsou definovány pouze pozicí je možné specifikovat jak výplň, tak okraj i velikost[52].

Také je možné využít pro reprezentaci bodu ikony element PointSymbolizer. Obrázek, který by se zobrazil je možné získat z lokálních zdrojů (uložené u instance serveru) nebo i online[52].

5.3 Zobrazování map

V této části budou popsány technologie použité pro zobrazování mapových dat v systému.

5.3.1 Google Maps

Mapy od společnosti Google jsou nejpopulárnějšími mapami na světě. Mimo základní zobrazení mapy také nabízí mapu satelitní.

Google získává data na úpravu map všemožnými způsoby. Jedním z nejdůležitějších je *Partnerství s Mapami*, díky kterému můžou lokální, či dokonce celostátní organizace poskytovat data pro aktualizaci map.

5.3.2 OpenStreetMap

OpenStretMap (OSM) je projekt, tvořen komunitou uživatelů, jehož cílem je tvorba a poskytování veřejně dostupných geografických dat v podobě topologických map[54].

OpenStreetMap data byla v dřívější době publikována pod licencí *Creative Commons Attribution-ShareAlike 2.0*, ale od roku 2008 jsou poskytována pod licencí Open Database License (ODbL)[54].

Data jsou dostupná například na adrese *openstreetmap.org*, kterou jako zdroj mapových podkladů defaultně používá javascriptová knihovna OpenLayers. Projekt OSM byl založen 9. srpna 2014, když byla zaregistrována doména openstreetmap.org[55].

5.3.3 OpenLayers

OpenLayers je knihovna pro jazyk JavaScript určená pro zobrazování mapových podkladů ve webových prohlížečích na straně klienta. Je dostupná kompletně zdarma a vyvíjena pod licencí FreeBSD open-source komunitou vývojářů[57].

Knihovna využívá principu klient-server komunikace a poskytuje API pro použití

ve vlastních aplikacích. Umožňuje načítat data implementací protokolů standardních služeb OGC - WMS a WFS[57].

Základní komponentou OpenLayers je mapa - *ol.Map*. Tato komponenta je poté renderována do žádaného elementu na stránce a obsahuje samotnou mapu. Mapa může mít nastavené požadované vlastnosti při své inicializaci nebo je možné je upravovat za běhu aplikace[57].

6 Perzistentní uložení dat

V rámci této diplomové práce bude třeba perzistentně uchovávat data, a proto v následující kapitole bude představeno a vysvětleno několik technologií, které budou využívány.

6.1 ORM

Objektově relační mapování (Object-Relational Mapping) je způsob automatické transformace dat mezi relačním databázovým systémem a objekty v objektově-orientovaném programování. Jedná se o vztah mezi třídou objektu, který se nazývá entita, a relací. Díky objektově-relačnímu mapování lze plně využít výhod objektově-orientovaného programování.

6.1.1 Hibernate

Hibernate je framework napsaný v jazyce Java, umožňující objektově-relační mapování. Framework Hibernate je jednou z implementací Java Persistence API (JPA).

6.1.2 SQLite

SQLite je relační databázový systém, který je implementovaný knihovnou v jazyce C. Databázový systém je považovaný za rychlý, vysoce dostupný, s širokou paletou funkcí a zároveň skromný na výpočetní prostředky[58].

SQlite je také popisován jako nejpoužívanější databázový systém na světě - zejména díky jeho všeobecnému využití v mobilních telefonech.

Na rozdíl od jiných databází, které obvykle data ukládají do velkého množství souborů, SQLite má uloženou celou databázi (schéma i data) v jediném souboru, který na každé platformě funguje stejně - tedy pokud je databáze vytvořena na jednom místě, lze ji používat i kdekoli jinde - nezávisí na operačním systému, preferenci big-endian či little-endian nebo podobně [59].

Implementovat práci s SQLite databázi lze v různých programovacích jazycích - například v C/C++, C#, Java nebo i Pythonu.

Zdrojový kód SQLite je také šířen pod licencí *public domain*[1)].

[1)] Zveřejnění k volnému využití bez nároku na další ochranu díla.

II. PROJEKTOVÁ ČÁST

7 Návrh a architektura systému

V této kapitole bude přiblížen návrh komplexního systému pro evidenci elektronické knihy jízd. Budou navrženy části, ze kterých se systém bude skládat, bude definované, pro koho je systém určen a za jakým účelem.

7.1 Záměr řešení

Cílem tvořeného řešení je poskytnutí dostupného a univerzálního systému elektronické knihy jízd.

Systém by byl určen pro firmy, které doposud řešily knihu jízd v papírové podobě a chtěly by si ji vést pohodlněji, více transparentně a přehledně.

Tento systém by mohl být zajímavá alternativa pro firmy, které sice knihu jízd chtějí řešit elektronickou cestou, ale zároveň chtějí (nebo potřebují) mít kontrolu nad všemi svými daty. Jelikož ostatní řešení jsou obecně poskytována formou produktu jako služby[1)], takže jsou data uložena na cizích serverech, zatímco toto řešení by bylo nabízeno formou klasického softwarového produktu a provozováno na infrastruktuře zákazníka. A tím pádem by byla všechna data ve správě a zodpovědností zákazníka.

Dále by tento systém mohl být využit pouze pro jednoduché sledování aktuální polohy vozidel bez nutnosti využívat ostatní funkcionality knihy jízdy.

7.2 Zdroj dat o jízdách

Do systému musí data o provedených jízdách nějakým způsobem vstupovat. Po analýze technických možností a existujících řešení lze stanovit několik cest jak to provádět.

- **Specializovaná fyzická jednotka** - Dedikovaná fyzická jednotka s GPS modulem, která přenáší automaticky data do systému přes veřejnou síť díky SIM kartě nebo jiné technologii.
- **Import ze souboru** - Import dat z existujícího souboru s naměřenými souřadnicemi jízdy či jinými informacemi.
- **Mobilní zařízení** - Získání aktuální pozice přes GPS modul mobilního zařízení a odesílání dat přes veřejný internet.
- **Palubní počítač** - Připojení se k palubnímu počítači automobilu a získávání informací z něj.

[1)] Princip SaaS (Software as a service.) - hostování aplikace provozovatelem služby, dále popisováno jako "služba".

- **Ruční vytvoření** - Vytvoření záznamu o provedené cestě se zadanými informacemi o vozidlu, účelu a délkou cesty, případně ruční nakreslení cest do elektronické mapy.

V aktuálním okamžiku bohužel nelze využít k účelu získání aktuální polohy systém eCall (ačkoliv obsahuje všechny potřebné komponenty). Není možné vyloučit, že v budoucnu bude možné tento systém nebo některou z budoucích existujících alternativ používat.

Jelikož je snahou této diplomové práce vytvořit systém, který je pro zákazníky co nejdostupnější, není možné využít první možnosti.

Také je komplikované získávání dat z palubního počítače - jsou způsoby získání potřebných informací přes sběrnici CAN, nicméně to vyžaduje drahý adaptér pro připojení, což také eliminuje tuto metodu.

V rámci této diplomové práce tedy byla zvolena jako primární metoda sběr dat přes mobilní zařízení. Pro to je nutné vytvořit aplikaci na mobilní zařízení, které bude tuto činnost implementovat. Dále bylo rozhodnuto o možnosti ručního vytvoření záznamů.

7.3 Zobrazení dat

Systém by měl poskytovat komfortní přístup a zobrazení dat. K tomuto účelu by mohla sloužit webová aplikace, zabezpečená a dostupná pouze registrovaným uživatelům. V aplikaci by mohla být zobrazována všechna získaná data jízd, včetně případné mapy trasy.

Ačkoliv to není účelem evidence knihy jízd, bylo by vhodné zobrazovat aktuální (poslední známé) pozice monitorovaných vozidel na mapě.

Také by bylo vhodné mít možnost si zobrazit historii vozidel na mapě - zobrazení provedených jízd.

Téže je nutné mít možnost informace knihy jízd exportovat ze systému v čitelném formátu pro případné další zpracování.

7.4 Návrh architektury

Na základě výše uvedených potřeb, je nutné obsáhnout v systému všechny zmíněné komponenty.

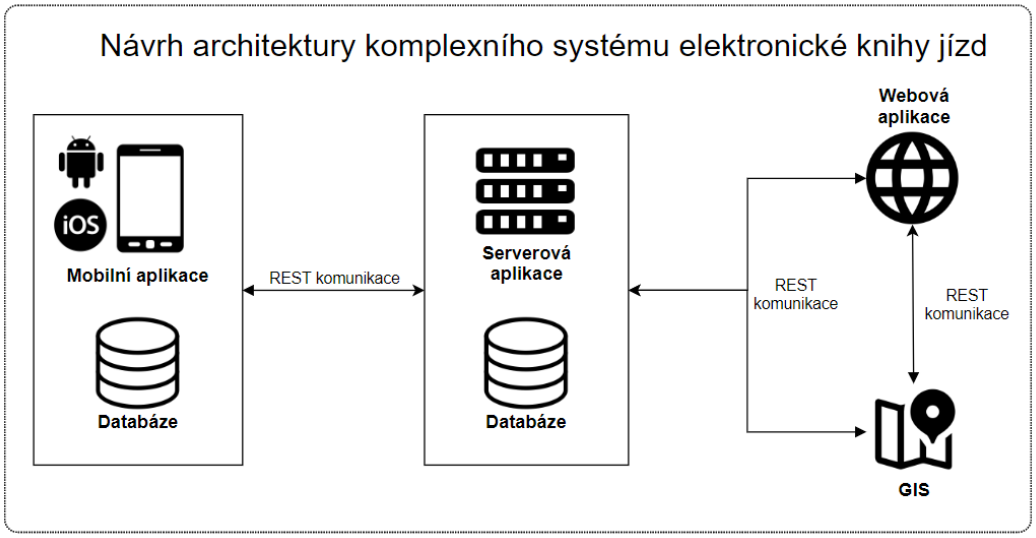
Do systému tedy je třeba zahrnout centrální serverovou aplikaci, se kterou bude komunikovat mobilní aplikace - bude jí odesílat data o proběhlých jízdách a jiné informace.

Součástí bude rovněž klientská webová aplikace, do které se uživatelé budou moci přihlásit a zobrazovat si existující informace. Pro vystavení klientské webové aplikace

bude využit webový server.

Také je tedy nutná přítomnost mapového serveru, na který budou publikována data ze serverové aplikace. O data z mapového serveru poté bude žádat webová aplikace pro zobrazení do map.

Jak by návrh takové architektury systému mohl vypadat je možné vidět na obrázku 7.1.



Obr. 7.1 Návrh architektury systému, zdroj vlastní

7.5 Uživatelé systému

Lze definovat množinu typů uživatelů, kteří by měli se systémem pracovat. Daní uživatelé budou moci se systémem provádět předem definované operace.

Primárním zájmem uživatelů systému je získání informací o aktuální poloze vozidel a o jejich dřívějších cestách.

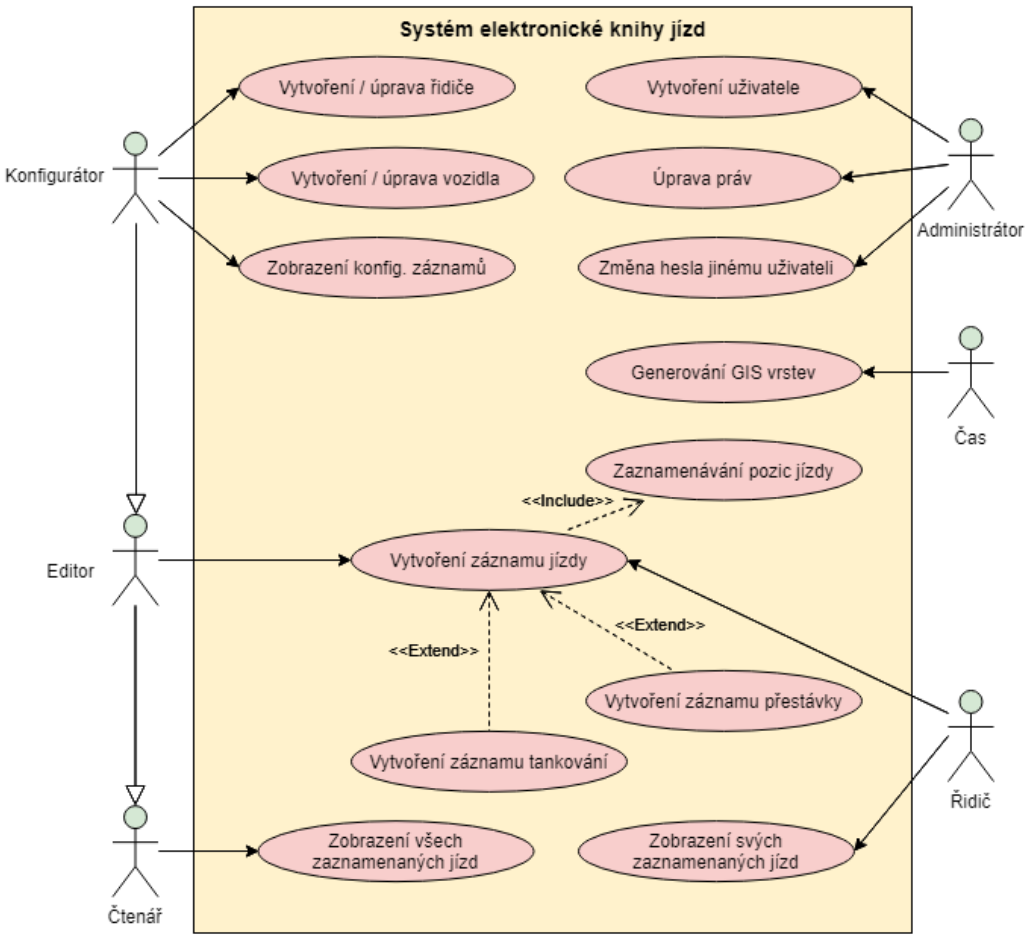
Je třeba existence uživatele typu Řidič, který bude mít právo vytvářet záznamy jízd, zaznamenávat k němu trasu a odesílat data z mobilní aplikace do centrální serverové aplikace.

Také by měli existovat uživatelé typu Čtenáři, kteří budou mít přístup do systému a budou moci prohlížet data - záznamy z knihy jízd. Nebudou ale moci informace upravovat - na to budou mít právo pouze uživatelé typu Editoři. Ti budou moci navíc data knihy jízd upravovat a také i přidávat.

Následně by také měl v systému existovat uživatel Typu Administrátor, který bude moci vytvářet a editovat ostatní uživatele.

Na základě výše popsaných akcí lze zformulovat níže uvedený jednoduchý diagram

případů užití[2)]:



Obr. 7.2 Diagram případů užití systému, zdroj vlastní

Ve skutečném implementovaném systému se budou role uživatelů prolínat a skutečný fyzický uživatel bude moci pracovat současně s oprávněními několika typů uživatelů.

7.6 Odlišení se od existujících řešení

V této části budou popsány hlavní rozdíly, kterými se toto řešení liší od již existujích a používaných a jaké to přináší výhody.

Možnost provozu na vlastní infrastruktuře

Na rozdíl od ostatních řešení (s výjimkou aplikace AUTOPLAN Kniha jízd), které jsou poskytovány jako služba, jde tento systém používat na vlastní infrastruktuře a tedy mít jistotu, že data nejsou zneužívána třetí stranou.

[2)] Vzhledem k potřebě provádění některých akcí pravidelně bez zásahu uživatele byl přidán do diagramu užití i aktér "Čas".

Systém také díky tomu může mít pod plnou kontrolou zákazník a pokud by mohlo dojít k jakémukoli výpadku, bylo by to jen kvůli problému na jeho straně.

Nepotřeba externích zařízení

V rámci komplexního systému pro vedení elektronické knihy jízd, řešeném v této práci je zahrnuta i mobilní aplikace určená primárně ke vstupu dat o cestách do evidence. Ostatní zkoumané systémy k tomuto účelu využívají speciální fyzickou monitorovací jednotku, což přináší i své výhody a nevýhody.

Výhody	Nevýhody
• Přesné měření díky dedikovanému GPS modulu	• Při nákupu většího počtu fyzických jednotek se cena může vyšplat na desetitisíce i statisíce Kč,
• Snadnější komunikace s dalšími čidly v automobilu	• Dočasně nemožnost monitorovat polohu vozidla, po HW nebo SW poruše, do opravy zařízení,
	• Nutnost pravidelné platby za dedikovanou SIM kartu.

Práci této fyzické jednotky tedy nahrazuje v řešeném systému mobilní aplikace cílená na nejrozšířenější mobilní platformy Android a iOS. Odpadá tedy jak nutnost nákupu speciálního zařízení, které samo o sobě v podstatě nelze k jinému účelu využít (je předpoklad, že zaměstnanci vlastní kompatibilní mobilní telefon) a také odpadá i nutnost platit tarif za dedikovanou SIM kartu určenou k přenosu informací během jízdy.

Data z mobilní aplikace lze synchronizovat i dodatečně například přes síť WiFi, ale i tak bylo v systému speciálně zajištěno, aby data z mobilní aplikace příliš nezatěžovala přenosovou kapacitu. Toto bylo řešeno v kapitole 10.2.2.

Po analýze ostatních řešení je zřejmé, že jedna z jejich největších nevýhod je závislost na fyzické jednotce instalované v automobilu.

Systém tedy byl navržen pro co nejjednodušší nasazení s potřebou nejmenších prostředků, bez nutnosti žádného dalšího speciálního externího zařízení, které některé služby pro evidenci elektronické knihy jízd využívají.

Primárním zdrojem informací o cestách tedy bude mobilní aplikace, která bude na základě GPS modulu sledovat aktuální pozici řidiče a vozidla.

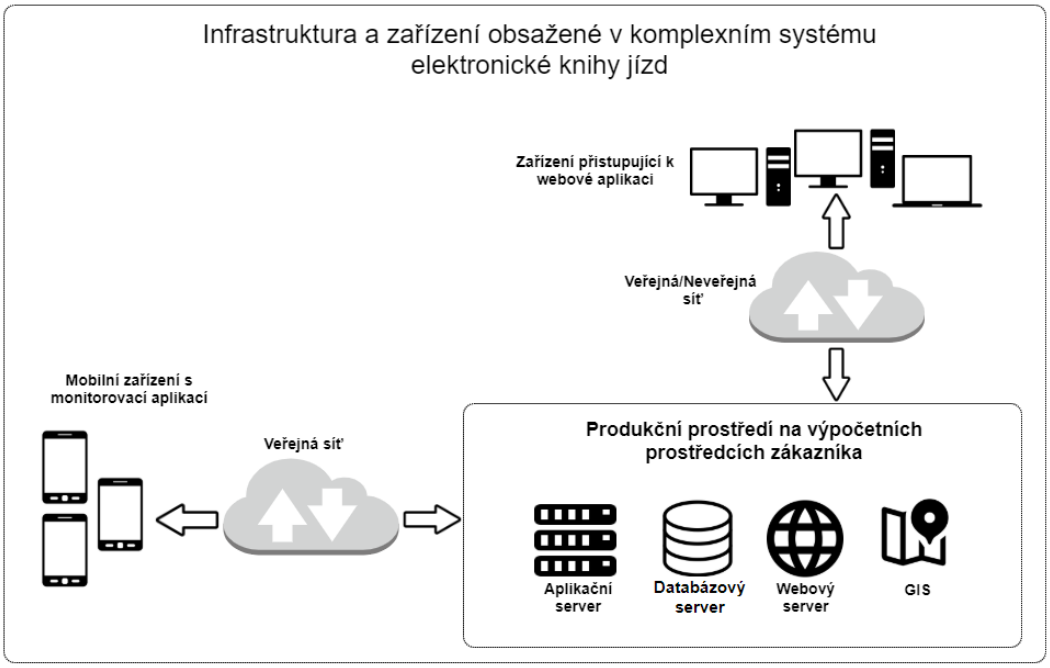
7.7 Vyžadovaná infrastruktura

Na základě výše uvedeného návrhu architektury systému, je třeba sestavit seznam vyžadovaných technologií a technických řešení, na kterých bude fungující systém nakonec nasazen.

Pro jednotlivé části systému byly zvoleny následující technická řešení zejména z důvodu již existujících zkušeností autora s nimi a kvůli jejich obecné rozšířenosti v podnikové sféře.

- **Aplikační server** - Wildfly,
- **Databázový server** - PostgreSQL,
- **Webový server** - Apache HTTP Server,
- **Geografický informační systém** - GeoServer.

Všechny navržené technologie jsou veřejně dostupné a zdarma.



Obr. 7.3 Vyžadovaná infrastruktura systému, zdroj vlastní

Systém nevyžaduje, aby byly jednotlivé části pohromadě, pouze je nutné, aby mezi nimi existovalo dostupné spojení. Je tedy možné provozovat systém i na existujících technických prostředcích zákazníka.

8 Definice požadavků na systém

V této kapitole budou analyzovány potřeby systému elektronické knihy jízd dle výše uvedeného návrhu a definovány požadavky, které budou na dané část systému kladeny.

Požadavky jsou rozděleny do sekce funkčních požadavků, kdy je definováno, co by systém měl dělat či umět a do sekce nefunkčních požadavků, které specifikují jaký by systém měl být.

8.1 Požadavky na systém elektronické knihy jízd

V této části jsou definovány požadavky, které se týkají celého systému elektronické knihy jízd.

8.1.1 Funkční požadavky

1. **Evidence informací** - V systému by mělo být možné evidovat vozidla, řidiče a informace, které se jich týkají
2. **Vstup dat** - Systém by měl umožňovat vkládat data uživatelsky přívětivou cestou
3. **Export dat** - Měla by existovat možnost výstup dat ve formě běžné knihy jízd
4. **Dohled** - Systém by měl poskytovat možnost zobrazit aktuální pozice vozidel

8.1.2 Nefunkční požadavky

1. **Splnění GDPR** - Systém by měl dodržovat obecné nařízení o ochraně osobních údajů (známé zejména jako GDPR[1)])
2. **Zabezpečení dat** - Data uložená v systému by neměla být dostupná nepověřeným osobám.
3. **Stabilita a dostupnost** - Systém by měl být stabilní (bezproblémové zvládání množství souběžných požadavků) a vysoce dostupný (systém nebude mít výpadky znamenající dostupnost nižší než 99% doby)

8.2 Požadavky na mobilní aplikaci

V této části jsou definovány požadavky na mobilní aplikaci určenou zejména pro sběr dat.

[1)] Popsáno v kapitole 3.2.

8.2.1 Funkční požadavky

1. **Monitorování trasy** - Mobilní aplikace by měla umožňovat zaznamenávat aktuální pozici vozidla.
2. **Perzistentní uložení dat** - Data v mobilní aplikaci by měla být uložena trvale a měla by být dostupná i po vypnutí aplikace či restartování telefonu
3. **Synchronizace dat**
   (a) Aplikace by měla umět synchronizovat do serverové aplikace záznamy během cesty.
   (b) Měla by být možnost synchronizovat záznamy, které nebyly dříve synchronizovány.
4. **Zaznamenání dalších informací do knihy jízd**
   (a) Možnost zaznamenat bezpečnostní přestávku během cesty,
   (b) Možnost zaznamenat návštěvu čerpací stanice a informace o ní.
5. **Zobrazení mapy cesty v mobilní aplikaci** - V aplikaci by měla existovat možnost vizualizace předchozích cest zaznamenaných v zařízení.
6. **Zobrazení mapy probíhající cesty** - Mělo by být možné si zobrazit aktuální probíhající trasu.

8.2.2 Nefunkční požadavky

1. **Dostupnost aplikace** - Aplikace by měla být spustitelná na co nejširším spektru mobilních telefonů
2. **Zabezpečení aplikace** - Aplikace by měla být dostupná pouze oprávněným uživatelům a měla by být zabezpečena heslem či gestem
3. **Připravenost a spolehlivost** - Aplikace by měla být spuštěna a připravena k záznamu jízdy do 20 vteřin. Aplikace by také neměla trpět pády do operačního systému.
4. **Uživatelský komfort** - Aplikace by neměla trpět pomalou odezvou a zasekáváním prostředí.
5. **Úsporné využití přenesených dat** - Aplikace by neměla odesílat ani přijímat přebytečné množství dat.
6. **Multijazyčnost** - Aplikace by měla být dostupná alespoň v češtině a angličtině

8.3 Požadavky na serverovou aplikaci

Serverová aplikace by měla být veřejně viditelná (dostupná i z veřejných zdrojů), aby do ni bylo možné posílat data z mobilní aplikace. Serverová aplikace také bude komunikovat s webovou aplikací

8.3.1 Funkční požadavky

1. **Zpracovávání požadavků z mobilní aplikace** - Serverová aplikace umožňuje obsluhu požadavků z mobilní aplikace.
2. **Generování a publikování mapových podkladů do GIS** - Serverová aplikace umí generovat různé mapové podklady.
   (a) Vrstva aktuálních pozic vozidel
   (b) Vrstvy s historií pohybu vozidel
3. **Možnost výstupu dat v čitelném formátu**
   (a) Generování knihy jízd
   (b) Další způsoby exportu dat

8.3.2 Nefunkční požadavky

1. Serverová aplikace musí být schopná obsluhovat požadavky až 15 klientských aplikací zároveň a obsluhovat požadavky nejméně od 100 instancí mobilních aplikací zároveň bez zpoždění znemožňující práci.
2. Serverová aplikace musí ke své práci vyžadovat pouze prostředky, které jsou dostupné zdarma.

8.4 Požadavky na webovou aplikaci

Webová aplikace by měla být hlavním místem pro zobrazování získaných informací uživatelům. Aplikace by také měla řešit administraci dat a konfiguraci systému.

8.4.1 Funkční požadavky

1. Zobrazování evidovaných jízd,
2. Zobrazení aktuální pozice vozidel na mapě,
3. Správa informací o vozidlech a řidičích,
4. Administrace uživatelů a přístupových oprávnění.

8.4.2 Nefunkční požadavky

1. Webová aplikace by měla fungovat a vypadat stejně v nejrozšířenějších a nejpoužívanějších webových prohlížečích,
2. Webová aplikace by měla být zabezpečena a umožnit přístup k datům a práci s nimi pouze přihlášeným uživatelům,
3. Ve webové aplikaci by měl být omezen přístup k datům dle úrovně práv,
4. Aplikace by měla být dostupná ve více jazycích - alespoň v češtině a angličtině.

9 Tvorba elektronické knihy jízd

V této části bude popsán proces tvorby systému elektronické knihy jízd. Budou popisovány problémy, které bylo třeba řešit, vysvětlovány rozhodnutí, které byly během práce udělány a obecně popisován celý sled práce.

9.1 Plnění obecných požadavků

V této části bude popsáno řešení obecných požadavků, které jsou kladeny na celý systém knihy jízd.

Dodržování podmínek GDPR

Pro dodržení podmínek obecného nařízení o ochraně osobních údajů byl sestaven obecný dokument (v českém i anglickém jazyce, dostupné v přiloženém médiu) pro souhlas s poskytnutím osobních dat s jakoukoli entitou, provozující systém elektronické knihy jízd založený na této diplomové práci.

Tento dokument je součástí obsahu přiloženého CD disku a měl by být podepsaný každým registrovaným uživatelem webové aplikace a každým uživatelem mobilní aplikace, jelikož jsou o nich v systému uchovávány osobní informace.

V případě žádosti uživatele o podání informace o evidovaných datech je možné udělat výpis informací z webové klientské aplikace.

Pokud by uživatel zažádal o vymazání svých informací ze systému, bylo by k tomu bohužel nutný zásah přímo v databázi. Bylo by nicméně možné implementovat formulář, dostupný pouze uživateli s vyšším oprávněním, pomocí kterého by bylo možné

Generování knihy jízd

Do systému jsou sbírány informace, které je nutné získat ven nějakým intuitivním řešením. Byl zvolen export dat

Samotný princip generování knihy jízd je popsán níže v kapitole 11.4.

10 Tvorba aplikace na mobilní zařízení

V této kapitole bude popisován postup tvorby aplikace na mobilní zařízení, která bude sloužit primárně ke sběru dat pro systém knihy jízd.

10.1 Volba nástrojů

Mobilní aplikaci jsem tvořil v multiplatformním frameworku Xamarin Forms. Použil jsem ho zejména pro jeho nespornou výhodu používání nativních elementů cílového prostředí - aplikace poté vypadá stejně, jako by byla vyvíjena přímo pro danou platformu.

Ačkoliv původně byl zamýšlený plán vyvíjet aplikaci souběžně pro platformu iOS a Android, bylo od toho brzy z důvodu nedostatku času a technických prostředků upuštěno.

Aplikace je tedy kompatibilní se systémem Android 5.0 (API 21) a vyšším, ačkoliv v průběhu vývoje byl testován zejména na přístrojích Android 8.0 (API 26), Android 8.1 (API 27) a Android 9.0 (API 28).

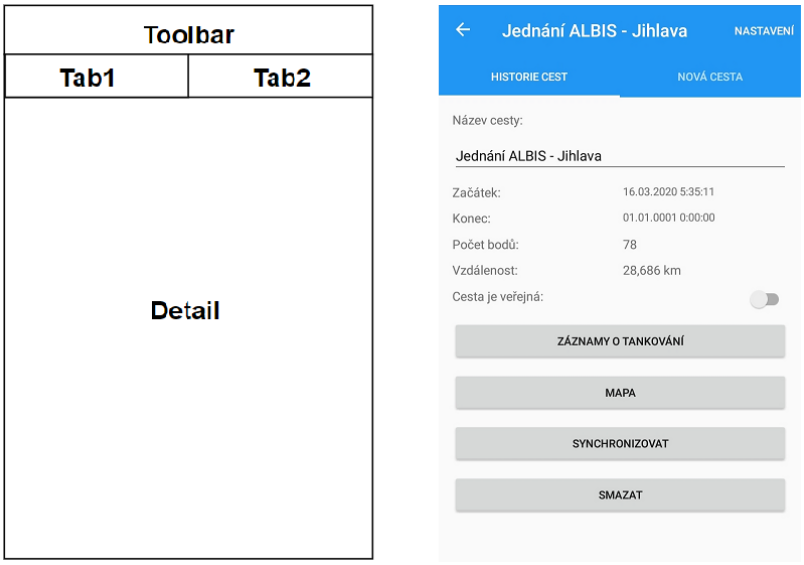
10.2 Návrh aplikace a implementace požadavků

V této části budou popsány nejdůležitější části řešení mobilní aplikace. Bude vysvětleno, jak byly řešeny klíčové požadavky, jaké problémy byly překonávány a jakým způsobem.

Návrh uživatelského rozhraní

Pro hlavní rozložení aplikace jsem se rozhodl použít rozložení *TabbedPage*. To rozděluje obrazovku do seznamu karet (v obrázku 10.1 v návrhu jako "Tab1"a "Tab2", v implementaci jako "Historie cest"a "Nová jízda".) a z větší části do detailu, kde je zobrazena aktuální vybraná stránka.

Také bylo použito panel nástrojů (anglicky "*Toolbar*"), kam byl umístěn odkaz na stránku nastavení aplikace a kromě toho se tam zobrazovaly informace v závislosti na aktuálním kontextu.



Obr. 10.1 Návrh rozložení a ukázka implementované aplikace, zdroj vlastní

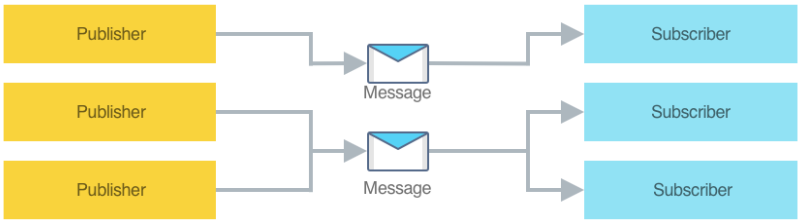
Dále byly v některých případech použity modální okna, kdy je celé rozhraní aplikace překryto jinou stránkou a nelze s aplikací pracovat než je stránka, která je obsahem modálního okna skryta.

Architektura aplikace

Jak již bylo zmíněno v kapitole 4.3.3 framework Xamarin je vhodný k implementaci dle návrhového vzoru *MVVM*. Jako základ aplikace jsou tedy jednotlivé stránky (anglicky "*page*"), které mají popsány svůj vzhled v jazyce XAML a jsou podpořeny kódem na pozadí (anglicky "*code-behind*") a k nim pak existují i ViewModel a data (Model).

Pro programátorský komfort byly také vytvořeny třídy, sloužící pro manipulaci dat, které byly vkládány. V aplikaci bylo referováno o těchto třídách jako o službách (PathService, VehicleService atd.)

V některých případech bylo nutné komunikovat napříč aplikací mezi komponentami, které na sebe neměly odkaz. Bylo k tomu využito rozhraní *MessagingCenter*, které umožňuje odesílat zprávy i s daty napříč aplikací. Na dané viz obrázek 10.2.



Obr. 10.2 Návrh rozložení a ukázka implementované aplikace, zdroj [60]

V níže uvedené ukázce kódu je odeslána zpráva:

```
MessagingCenter.Send<object, string>(Application.Current,
    "UpdateRunningMap", idPath);
```

A v níže uvedené ukázce kódu je zpráva v jiné části aplikace zachycena a zpracována.

```
    MessagingCenter.Subscribe<object, string>(Application.Current,
        "UpdateRunningMap", (sender, uuid) =>
    {
        this.idPath = uuid;
        UpdateMapOnline();
    });
```

Návrh a implementace komunikačního rozhraní

Jak již bylo uvedeno v kapitole 7.4, mobilní aplikace bude komunikovat se serverovou aplikací přes REST komunikační rozhraní.

Na začátku práce, při analýze potřebných metod, bylo dosaženo zjištění, že je třeba implementovat pouze sadu jednoduchých operací v následující tabulce:

Tab. 10.1 Navržené operace pro komunikaci se serverovou aplikací

Operace	Typ požadavku
Odeslání informací	POST
Zažádání objektu	GET

Proto byla pro komunikaci implementována třída *HTTPApiService*, kde byly vytvořeny generické metody pro odesílání a načítání dat. Pro samotnou komunikaci bylo využito knihovny *System.Net.Http.HttpClient*, která poskytuje prostředky pro odesílání a příjem HTTP požadavků.

Nejdříve tedy bylo využíváno tohoto zmíněného komunikační rozhraní pro získávání hodnot, objektů nebo listů objektů. Nicméně postupně byl však zjištěn ve třídě *HTTPApiService* problém s deserializací získaných objektů, a proto je nyní obecně získávána hodnota typu řetězec (*string*), která je deserializovaná či jiným způsobem analyzovaná až následně v metodě, která třídu HTTPApiService využívá.

Při práci s komunikačním rozhraním byla také objevena kuriózní potíž s komunikací na platformě Android v případě, kdy neexistuje způsob navázání připojení (na zařízení nejsou dostupná mobilní data ani Wi-Fi). V tom případě je vyvolána výjimka *Java.Net.UnknownHostException*, která nicméně nepatří do knihoven C# a nedá se přímo zachytit. Bylo to tedy vyřešeno následovným způsobem:

```
catch (Exception e) {
    if (e.GetType().ToString() == "Java.Net.UnknownHostException") {
        throw new HttpException(HttpExceptionReason.NO_CONNECTION, "");
    }
}
```

Obecně bylo během práce naráženo na problémy s komunikačním rozhraním, které ale byly řešeny proprietánimi metodami (kde se přímo pracuje s třídou *HttpClient*) na místech, kde bylo třeba komunikovat.

Výsledkem je, že postupem času přerostlo komunikační rozhraní "přes hlavu" a v případě další práce na projektu by bylo rozhodně jako jedno z prvních, které by bylo od začátku přepsáno.

Struktura databáze a její řešení

Struktura databáze v mobilní aplikaci je jednoduchá - existuje v ní několik relací, které mezi sebou nemají žádné přímé vazby (žádné cizí klíče).

Relace v databázi odpovídají objektům, se kterými se v aplikaci pracuje. Po analýze problematiky bylo nalezeno pět typů záznamů, které bylo potřeba uchovávat. Během práce se objevil důvod uchovávat ještě další typ - *PathSyncDto*.

Níže jsou popsány jednotlivé uchovávané typy záznamů.

- **BreakDto** - Záznamy o provedené přestávce během cesty - začátek a konec přestávky, odkaz na cestu.
- **GasStopDto** - Záznamy o zastávce na čerpací stanici - datum zastávky, celková cena, cena za litr, počet načerpaných litrů, název čerpací stanice a odkaz na cestu.
- **PathDto** - Záznamy o uskutečněné cestě - účel a cíl, začátek a konec, přiznak soukromá/služební cesta, počet kilometrů na začátku a počet kilometrů na konci
- **PathSyncDto** - Informace o synchronizaci cest na centrální serverovou aplikaci - odkaz na cestu, čas poslední synchronizace, .
- **PointDto** - Záznamy pozic zjištěných během cesty - souřadnice, čas, odkaz na cestu.
- **VehicleDto** - Záznamy evidovaných vozidel - model, barva a SPZ vozidla.

Databáze v mobilní aplikaci je implementována pomocí knihovny *SQLite.NET* získanou z balíčku NuGet. Pro práci s daty byla vytvořena třída *DBService*, která využívá *SQLiteAsyncConnection*.

V konstruktoru *DBService* je samotná databáze vytvářena při spouštění aplikace. Soubor, ve kterém je databáze uložena, je vytvářen do soukromé složky aplikace - na platformě Android se nachází v "/data/data/NÁZEV_BALÍČKU/files".

Manipulaci s databází byla řešena přes metody, serializující objekty do relací (využívání principu ORM).

Pouze v jednom případě bylo třeba využít něco jiného, bylo třeba efektivně zjišťovat počet bodů v dané cestě a byl k tomu využit pojmenovaný dotaz níže:

```
await _database.ExecuteScalarAsync<int>(
    "select count(*) from PointDto where idPath = '" + pathId + "';"
);
```

Bohužel stejně jako v případě komunikačního rozhraní, i v případě práce s DB bylo nakonec definováno více metod v různých třídách místo jen na jednom místě v *DBService*. V tomto případě to nicméně nebyl takový problém. Tyto metody byly definovány v třídách, kde se s těmito objekty manipulovalo - pro ukázku například získání pouze nejnovějších bodů v dané trase:

```
    return await _database.Table<PointDto>()
                  .Where(i => ((PointDto)i).idPath.Equals(pathId))
                  .Where(i => ((PointDto)i).numberInPath > numberFrom)
                  .ToListAsync();
```

Obecně se dala označit práce s databází za méně problémovou část vývoje mobilní aplikace.

Přihlášení do mobilní aplikace

Mobilní aplikace je zabezpečena nutností ověřit se emailem a heslem. Přihlásit se do aplikace mohou pouze uživatelé, kteří jsou zaregistrováni (jsou v databázi serverové aplikace) v systému knihy jízd, kde jsou vedeni jako "Řidiči".

Obr. 10.3 Ukázka přihlašovací obrazovky mobilní aplikace, zdroj vlastní

Požadavek na přihlášení uživatele mobilní aplikace je odeslán POST požadavek do serverové aplikace na REST rozhraní. V požadavku je obsažen objekt obsahující uživatelské jméno ve formě emailu a hash hesla algoritmem *SHA-512*.

Na základě navráceného objektu ze serveru je uživatel buď přihlášen do aplikace a je přesměrován na hlavní obrazovku, případně je mu zobrazena odpovídající chybová hláška.

Pro používání aplikace je třeba mít při prvním spuštění dostupné připojení k veřejnému internetu, aby bylo možné ověřit uživatele vůči serverové aplikaci.

Aby se uživatel nemusel přihlašovat při každém spuštění aplikace (a také pro možnost používat aplikaci offline), jsou po prvním úspěšném přihlášení uloženy informace[1)] o uživateli aplikace do perzistentního slovníku vlastností Xamarin *Application.Current.Properties*. Pokud tyto informace nejsou uloženy, je uživatel přesměrován na přihlašovací obrazovku aplikace.

Lokalizace aplikace

Aplikace má dvě jazykové verze, českou a anglickou. V jakém jazyce bude aplikace závisí na nastavení mobilního telefonu. Defaultní jazyk je čeština. Při lokalizaci aplikace se využivají soubory XML s příponou .resx, do kterých se zapisují níže uvedená data.

- **Název** - proměnná, která se používá v aplikaci pro vypsání hodnoty (např. *alertIncorrectPassword*)

[1)] Jedná se o unikátní identifikátor, který uživateli náleží v serverové aplikaci.

- **Hodnota** - přeložený text, který se zobrazí v aplikaci (např. "Heslo není správné.", "The password is incorrect.")
- **Komentář** - nepovinná část, která se používá pro zapsání dodatečných informací

Pro správnou funkčnost je potřeba v souboru Assembly.info specifikovat váš požadovaný jazyk v NeutralResourceLanguage, pro češtinu to vypadá následovně:

```
[assembly: NeutralResourcesLanguage("cs")]
```

Aktuálně není implementována ruční změna jazyka v aplikaci.

Proces záznamu trasy

Proces monitorování trasy začíná jejím manuálním spuštěním v mobilní aplikaci. Jeho jednotlivé kroky lze popsat následovně:

1. Uživatel musí před spuštěním cesty zadat její účel a cíl. Také musí zvolit ze seznamu vozidlo, kterým bude cestu absolvovat a jeho stav na tachometru. Dále také musí zvolit, zda bude cesta soukromá či veřejná.
2. (a) Během cesty může uživatel vytvořit záznam o natankování.
   (b) V průběhu cesty také uživatel může zaznamenat přestávku.
3. Před ukončením trasy uživatel zkontroluje, zda odpovídá vypočítaný počet kilometrů na tachometru s jeho skutečným a v záporném případě vloží správnou hodnotu.

Monitorování trasy - způsob a proces získání polohy

Monitorování tras vozidel je hlavní důvod vývoje mobilní aplikace. Při návrhu aplikace se počítalo se dvěma způsoby monitorování trasy.

První (a preferovaný) z nich spočíval v připojení mobilní aplikace přes Bluetooth k vozidlu a získání aktuální pozice a dalších informací (rychlost vozidla, stav tachometru, palivové nádrže..) z vozidla.

Tento způsob nicméně není možné použít ve všech případech - velká část automobilů (a motocyklů) neobsahuje palubní počítač, případně instalovaný palubní počítač neumožňuje připojení a získání informací.[2)]

Proto byl navržen druhý způsob monitorování - využití GPS modulu v samotných mobilních telefonech a získávání aktuální pozice přes dostupné API. Dodatečné potřebné informace (stav tachometru atd.) poté musí být vložen samotným uživatelem.

[2)] Toto téma již přesahuje zadání práce.

Aktuální pozice dle GPS je tedy získávána v pravidelných intervalech v průběhu spuštěného zaznamenávání trasy.

Pro pravidelný běh úlohy na pozadí byla použita funkcionalitu z NuGet balíčku Matcha.BackgroundService. Tato služba běží na pozadí a má zaregistrovanou úlohu, kterou vykonává dle nastavené periody.

V případě této aplikace je vykonávanou úlohou získání aktuální polohy. Následně je tedy volána metoda *SaveCurrentPoint* ve třídě *GPSPointSaveService*, kde je vytvořena žádost na získání aktuální polohy s požadovanou přesností. Pro získání aktuální pozice je použita funkcionalita balíčku Xamarin.Essentials.

Nejprve je tedy vytvořen požadavek s nastavenou úrovní přesnosti:

```
var request = new GeolocationRequest(GeolocationAccuracy.High);
```

A následně je zažádáno o asynchronní získání aktuální polohy:

```
var location = await Geolocation.GetLocationAsync(request);
```

Poté může nastat několik případů - pokud bude korektně získána aktuální hodnota, je navrácen objekt typu *Location*. Může nicméně dojít k různým výjimkám, které je nutné ošetřit - například v situaci, že aplikace nemá přiřazeny dostačující práva či pokud není možné data získat. V případě úspěchu nicméně lze z navráceného objektu získat souřadnice a uložit je.

Pro platfomu Android je třeba umístit do souboru *AndroidManifest.xml* žádost o přístup k GPS API ve formě povolení.

- *android.permission.ACCESS_FINE_LOCATION*
- *android.permission.ACCESS_COARSE_LOCATION*

Zjištění nejvhodnějšího nastavení měření polohy

Jak bylo výše zmíněné, lze při žádání o zjištění polohy zvolit více možných úrovní přesnosti. API Xamarin.Essentials poskytuje výčtový typ *GeolocationAccuracy* s pěti (respektive šesti[3)]) různými úrovněmi přesnosti, které jsou různě náročné na výkon telefonu.

Při vývoji aplikace byly testovány zmíněné úrovně, aby bylo určeno, která z nich je pro aplikaci nejvhodnější. Poznatky jsou uvedeny níže v tabulce 10.2. Experimentálně byla zjištěna přibližná praktická přesnost (maximální délka odchylky od správné lokace) a průměrná délka odezvy (doba od požádání o získání aktuální polohy do momentu získání hodnoty). Hodnoty byly rozděleny do dvou kategorií - ve městě a ve volné krajině, níže bude vysvětlen důvod.

Tab. 10.2 Zjištěné informace o nastavení přesnosti GPS

Úroveň	Uváděná přesnost	Odchylka krajina/město	Odezva krajina/město
Lowest	1000-5000 metrů	30 m / 300 m	0.6 s / 2.4 s
Low	300-3000 metrů	30 m / 270 m	0.8 s / 2.7 s
Medium	30-500 metrů	25 m / 120 m	0.9 s / 3.6 s
High	10-100 metrů	10 m / 70 m	1 s / 3.8 s
Best	do 10 metrů	7 m / 60 m	1.3 s / 4.4 s

Je nutné podotknout, že kromě krajních případů (které nebyly do statistik uváděny), kdy zařízení nečekaně oznámilo polohu stovky metrů daleko od té skutečné, byly získávány překvapivě velmi dobré výsledky.

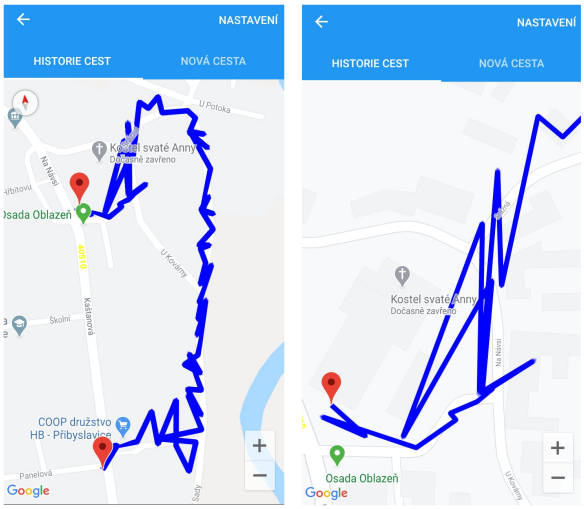
Problémy nastávaly ve městech nebo obecně na místech s hustší a vyšší zástavbou. Zde začínaly mít zařízení problém a získání polohy jim trvalo déle a bylo výrazně nepřesnější. U horších (*Lowest*, *Low*) nastavení docházelo k odchylkám v řádech stovek metrů, ty lepší úrovně nastavení (*High*, *Best*) se zpravidla pohybovaly s odchylkami v rozmezí 5 m - 20 m.

Na základě zjištěných hodnot by se dalo říci, že pro měření bude nejvhodnější zvolená přesnost *Best*, jelikož má nejlepší přesnost, na odezvě (kromě extrémních případů) nezáleží a rozdíl ve výdrži zařízení nebyl patrný.

Nicméně při testování na různých zařízeních se objevoval vcelku častý problém (zhruba v 10 % případů) s nastavením *Best*, kdy zařízení po startu nemohlo získat přibližně 30 - 60 s aktuální pozici. Tento problém se objevoval i s ostatními nastaveními i když mnohem vzácněji.

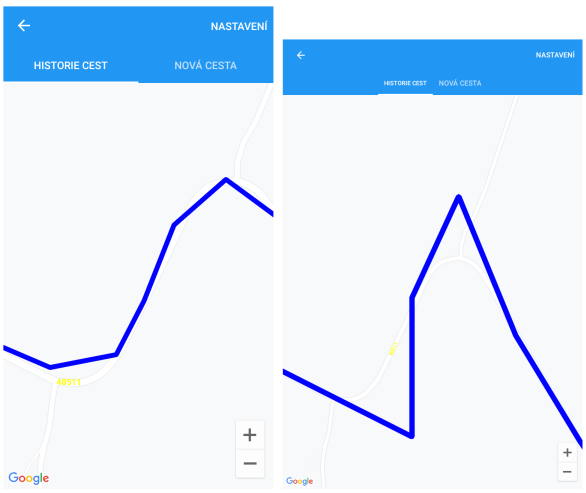
Na následujím obrázku 10.4 je vidět reprezentativní kvalita přesnosti Lowest, kdy zařízení mělo v některých případech velké problémy získat korektní pozici.

[3)] Výčtový typ GeolocationAccuracy obsahuje šest různých hodnot, nicméně hodnota *Default* odpovídá hodnotě *Medium*, a proto se o ni dále nebudeme zmiňovat.



Obr. 10.4 Ukázka přesnosti získání pozice s nastavením Lowest, zdroj vlastní

Na dalším obrázku 10.5 níže pak je v detailu porovnána přesnost nastavení High a Low, kde je vidět, že při obou nastavení byla získána přibližná poloha, nicméně pouze u nastavení lze hovořit o jejím praktickém využití pro měření vzdálenosti.



Obr. 10.5 Ukázka přesnosti získání pozice s nastavením High (nalevo) a Low (napravo), zdroj vlastní

Bylo také nutné zvolit ideální periodu zjišťování aktuální polohy. Vzhledem k určité odezvě při samotném získávání GPS pozice přes API a také k přihlédnutí na datovou náročnost nebylo možné zvolit příliš nízkou hodnotu, jelikož by bylo ukládáno nadměrné množství zbytečných dat. V případě zvolení příliš vysoké hodnoty by ale naopak byla ztrácena přesnost.

Například v případě měření každých 10 vteřin při rychlosti 90 km/h by automobil

mezi každým měřením ujel zhruba 250 metrů. Toto nastavení stačí pro získání přibližné polohy, ale není dostatečně přesné pro výpočet délky trasy a její vykreslení.

Nicméně experimenty na několika zařízeních bylo zjištěno, že nejvhodnějším nastavením monitorování pozice je získávání hodnoty s přesností *High* a měření je prováděno každých 5 vteřin.

Výpočet vzdálenosti cesty

Jeden ze sledovaných údajů v knize jízd je také ujetá vzdálenost. Bylo možné ji počítat několika způsoby.

- **výpočet na základě stavu tachometru** - rozdíl stavu tachometru na začátku a konci trasy,
- **výpočet na základě GPS souřadnic** - součet vzdáleností mezi získanými pozicemi trasy.

Při výpočtu vzdálenosti na základě měření GPS je nutné brát v potaz několik faktorů. Během cesty je poloha získávána jen s pevně danými intervaly a tedy mohou být některé zatáčky, které jsou projety mezi měřenými body, opomenuty a místo nich je počítána vzdálenost jako přímka. Také samotné získané GPS souřadnice nejsou vždy přesné a může v nich dojít k odchylce.

Při vývoji a testování aplikace bylo zjištěno, že se vypočítaná vzdálenost liší od skutečné zhruba o 3 - 8 % v závislosti na typu trasy[4)]. Na základě experimentů a následného porovnání zjištěné vzdálenosti vůči online mapám[5)] bylo zjištěno, že délka skutečné trasy odpovídá té vypočítané pokud je ponížena o zhruba 7 %.

Bylo nicméně zvoleno kompromisní řešení - při ukončení trasy je uživateli zobrazen odhadovaný stav tachometru zjištěný ze součtu hodnoty počátečního stavu tachometru a naměřené vzdálenosti. Uživatel může buď tuto hodnotu přijmout nebo ji změnit. V tom případě je pak uložena vzdálenost trasy jako rozdíl zvolené hodnoty a počátečního stavu tachometru.

Zobrazování cest na mapě

Jako přidanou hodnotu uživateli bylo rozhodnuto o zobrazení mapy jeho aktuální pozice a probíhající jízdy na mapě v mobilní aplikaci. Odkaz na tuto mapu se přidá do seznamu karet (hlavního navigačního panelu v aplikaci) po spuštění trasy a mapa se mu vykreslí po získání alespoň jednoho bodu cesty. V mobilní aplikaci bylo pro zobrazení map využito NuGet balíčku *Xamarin.Forms.Maps*.

[4)] Cesty, probíhaly z větší části ve městě a s delšími zastávkami na semaforech či křižovatkách se lišily kvůli větší odchylce až o 6-11 %.

[5)] Porovnáváno s hodnotami získanými z Mapy.cz a Google Maps

Na platformu Android bylo použito nejjednodušší řešení získání mapových podkladů, tedy využití Google Maps API. Pro jeho používání je nutné do aplikace dodat Google Maps Android API klíč v2. Klíč lze získat registrací na platformu Google Cloud Platform a vytvořením projektu, pro který je aktivováno API Maps SDK for Android. Dále je nutné pro vytvořený projekt zapnout fakturaci a použití existujícího či vytvoření nového fakturačního účtu. Je třeba spojit s účtem kreditní/debitní kartu.

Po získání klíče je třeba ho vložit do souboru *AndroidManifest.xml*. Také je nutné přidat i další nastavení pro kompatibilitu s API verzí 22 a nížší (povolení zápisu do externího úložiště) a pro kompatiblitu s API verzí 28 (specifikace použití Apache HTTP knihovny).

Výše zmíněné nastavení by se pak mělo nacházet v souboru AndroidManifest.xml v následující podobě:

```
<manifest ...>
  <uses-permission
     android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <application ...>
    <meta-data android:name="com.google.android.maps.v2.API_KEY"
               android:value="KLIC" />
    <uses-library android:name="org.apache.http.legacy"
       android:required="false" />
      <meta-data android:name="com.google.android.gms.version"
                 android:value="@integer/google_play_services_version" />
    </application>
</manifest>
```

Jako zdroj mapových podkladů v mobilní aplikaci byl tedy použitý Google Maps API, to limituje využití map pouze s připojením k internetu. V případě nedostupnosti připojení je na to uživatel upozorněn.

Jako rozšíření je tedy vhodné se zamyslet nad možností cachovat mapové podklady v telefonu, či je poskytovat v telefonu jako offline řešení.

Samotné mapy jsou vytvářeny ze seznamu získaných souřadnic, kdy se do mapy vykreslují *špendlíky* (anglicky "*pin*"), které specifikují začátek cesty a aktuální pozici. Mezi všemi body jsou poté vedeny lomené čáry (anglicky "*polyline*").

V průběhu cesty se dynamicky přidávají nejnovější pozice. Při získání nové pozice se aktuální pozice přepíše na nejnovější pomocí odesílání zpráv přes MessagingCenter.

V aplikaci je také možnost zobrazit mapu u již dokončené cesty. Princip zobrazení je podobný jako při zobrazování probíhající mapy. Při zobrazení dokončené cesty se ze všech bodů na mapě získá nejsevernější a nejjižnější šířka a nejzápadnější a nejvýchodnější délka a následně se podle těchto údajů přizpůsobí zoom na mapě, tak aby byla vidět celá cesta i se začátkem a koncem.

10.2.1 Synchronizace dat

V této části bude popsána synchronizace dat, která probíhá z mobilní aplikace do centrální serverové aplikace.

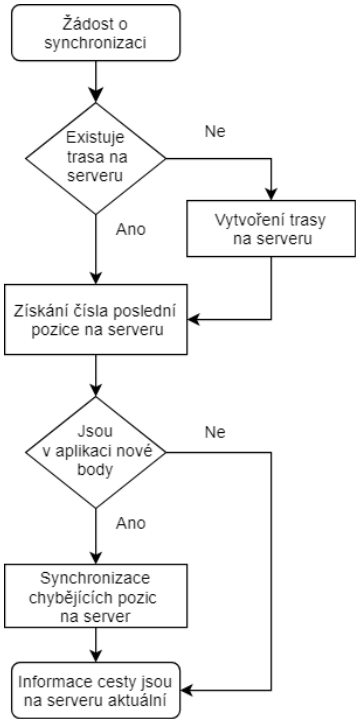
Do synchronizovaných dat patří záznamy provedených cest a jejich zjištěných pozic. Dále se synchronizují i data o provedených přestávkách a natankování na čerpací stanici.

Synchronizace probíhá během samotné trasy, ale je možné ji spustit i manuálně a synchronizovat jednotlivě celou trasu, případně synchronizovat i všechna data v aplikaci najednou.

Nejčastěji je prováděna synchronizace během prováděné trasy. Pro tento účel byla vytvořena statická třída *BackgroundOnlineSyncService* a metoda *syncCurrentPath* - ukázka kódu níže, která jednoduchým způsobem zajišťuje, aby byl prováděn pouze jeden proces synchronizace najednou.

```
public static async void syncCurrentPath(String idPath) {
    if (!syncInProgress) {
        syncInProgress = true;
        SyncService syncService = new SyncService();
        await syncService.SyncPathPointsLightweight(idPath);
        syncInProgress = false;
    }
}
```

Samotný proces synchronizace je implementován ve službě *SyncService*, kde je prováděna činnost dle následujícího vývojového diagramu:



Obr. 10.6 Vývojový diagram synchronizace dat během cesty, zdroj vlastní

Serverové API je pro mobilní aplikaci dostupné na následující adrese:

```
'BASE\_URL'/rest/logbook/sync/
```

Kde 'BASE_URL' odpovídá adrese, kde bude mít zákazník vystavenu aplikaci.

Pro ověření u těchto metod přes HTTP je využito *basic access authentication* - tedy ověření jménem a heslem.

Tohoto témata synchronizace se velmi blízko týká následující kapitola, jelikož frekvence a množství odesílaných dat se může velmi rychle vymknout z rukou a proto je nutné tento proces optimalizovat.

10.2.2 Optimalizace přenesených dat

Jelikož v rámci systému je kladen důraz na provoz s co nejmenšími náklady, bylo třeba optimalizovat množství odesílaných dat.

Primární optimalizace byla provedena omezením množství odesílaných zjištěných pozic - je odesílán jen vždy rozdíl dat od poslední synchronizace na server. Tedy pokud na serveru je uložených 400 pozic a v mobilní aplikaci již je aktuálně vytvořených 410, bude odesláno pouze posledních deset pozic, tato optimalizace byla zásadní.

Další optimalizace přenášených dat spočívala zejména z přejmenování parametrů v

serializovaných objektech na kratší názvy, což snížilo objem přenášených dat o zhruba 25 %.

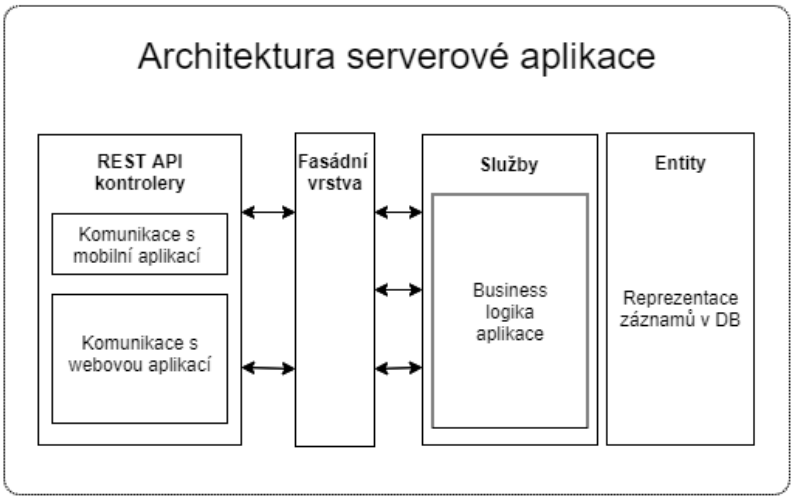
11 Serverová aplikace

Jak bylo zmíněno v kapitole 7.4, serverová aplikace bude sloužit k obsluze požadavků z webové a mobilní aplikace. Na základě definovaných požadavků a návrhu tedy lze považovat serverovou aplikaci za "srdce" celého systému.

11.1 Architektura

Serverovou aplikaci jsem se rozhodl založit na rodině frameworků Spring, zejména z důvodu již existující zkušenosti s nimi, ale také i pro další prohloubení vědomostí. Kromě samotného Spring Frameworku také byla využita knihovna *cz.envinet.java.ewa*, která výrazně usnadňuje práci s databázemi.

Pro serverovou aplikaci byla navržena jednoduchá architektura založené na komponentách ze Spring Frameworku.



Obr. 11.1 Schéma architektury serverové aplikace, zdroj vlastní

Jelikož většina činností v této aplikaci je iniciována z vnějšku - z mobilní či webové aplikace bude popsána architektura dle postupu, v jakém pořadí jsou jednotlivé části architektury pro tuto činnost potřebné.

Pro komunikaci s webovou a mobilní aplikací byly vytvořeny *controllery* - *RestController*, pomocí kterých je implementovány v aplikaci RESTFul webové služby. Pokud tedy přijde požadavek na serverovou aplikaci a pokud je správně *namapován*[1)] na některý z *controllerů* a některou z metod [2)] a zároveň odpovídá i typ požadovaných dat,

[1)] Controllery a metody v něm definované jsou přístupné přes "RequestMapping"- anotace s odpovídajícím řetězcem(označením).

[2)] Cestě v URL požadavku vyhovuje přesně jeden controller a metoda.

je přijat.

Následně je přes fasádní vrstvu prováděna business logika v některé z tříd reprezentujících *službu* (anotováno @Service), které jsou dostupné jako *bean* (tedy možné používat i bez nutnosti se starat o jejich vyváření díky principu IoC). V rámci business logiky pak může být prováděna práce s perzistentními daty. Ty jsou v aplikaci reprezentovány jako *entity*.

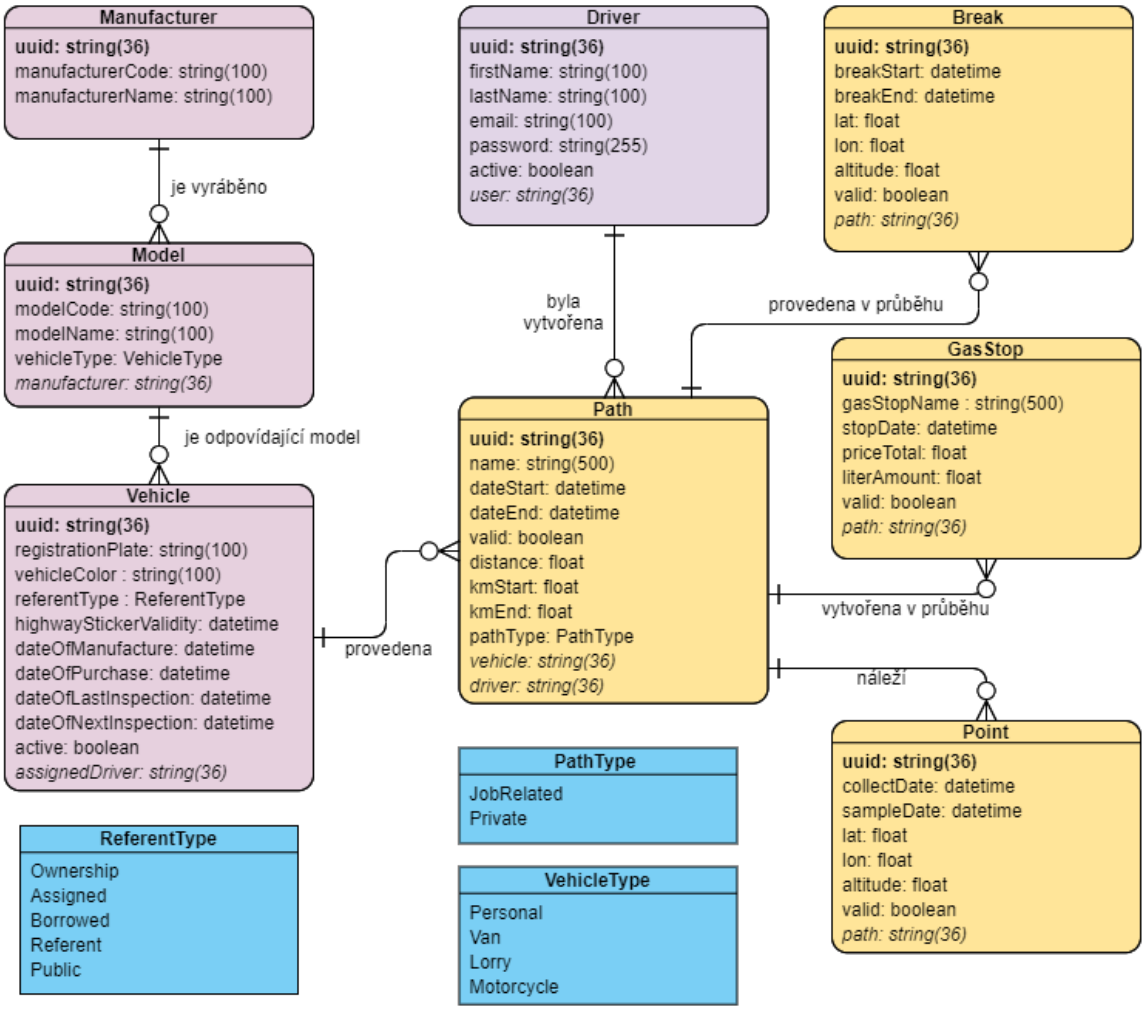
11.2 Databázová struktura

Na základě požadavků systému byla navržena a implementována databáze.

Zde, v textu práce, je popsána zejména část databáze, která se týká logiky řešící knihu jízd. V databázi se ještě nachází další databázové relace, které obsahují data pro fungování celého systému (správa uživatelů a jejich práv, aplikační proměnné, šablony dokumentů atd.).

Na následujícím obrázku 11.2, který obsahuje zjednodušený[3)] návrh DB schéma, jsou pro přehlednost rozděleny databázové relace různým obarvením do dvou skupin. Databázové relace obsahující konfigurační záznamy (informace vytvořené ručně uživateli, které se často nevytváří nebo nemění) jsou obarveny šedě a databázové relace s "datovými"záznamy (záznamy, které vyjadřují nasbírané informace a ze kterých se zejména skládá kniha jízd) mají žlutobéžovou barvu. Primární klíče jsou označeny tučným fontem a cizí klíče kurzívou. Také ve schéma byly dodány použité výčtové typy, označené modrou barvou.

[3)] Pro stručnost byly ve schématu zobrazeny pouze ty důležité parametry. Také názvy relací ve schématu odpovídají názvům entit v aplikaci, nikoliv těm v DB a datové typy odpovídají definovaným typům v entitách.



Obr. 11.2 Zjednodušené DB schéma, zdroj vlastní

Velká část konfigurace se týká uložení metadat automobilů. V relaci *Manufacturer* jsou ukládány záznamy o výrobcích vozidel a v relaci *Model* jsou naopak ukládány jednotlivé modely, přiřazené k některému z výrobců. Toto bylo vytvořeno pro jednoznačnost a elegantnější evidenci vozidel. Následuje již důležitá relace *Vehicle*, ve které jsou uchovávány záznamy a důležité informace všech vozidel evidovaných v systému. Vozidlo může být evidováno pouze pod jedním modelem.

V případě evidovaného vozidla "Ford Focus s poznávací značkou AB1 41745"v řešeném systému lze říci, že v relaci *Vozidla* je registrované s SPZ "AB1 41745", které je navázané na záznam v relaci *Model* s kódem "Focus"a ten je navázán na záznam s kódem "Ford"v relaci *Manufacturer*.

Dále jsou v databázi v rámci konfigurace uchovávány záznamy řidičů - v relaci *Driver*. U řidičů je uložen jejich email (nebo reálně jakýkoli řetězec), kterým se identifikují v mobilní aplikaci a hash hesla algoritmem SHA-512.

Jak lze ze schéma vidět, záznam řidiče je možné provázat s uživatelem. Aplikace je

tak připravena na rozšíření, kdy bude přihlášeným uživatelům zobrazeny pouze jeho data.

Ostatní tabulky v této části schéma DB knihy jízd se již týkají dat. Hlavní relací, na kterou jsou ostatní navázány je *Path*, což je záznam provedené jízdy. Ostatní jsou relace *GasStop* - záznam o návštěvě čerpací stanice, *Break* - záznam o provedené přestávce v neposlední řadě i relace *Point*, ve které je očekáváno nejvíce záznamů, jelikož obsahuje jednotlivé naměřené pozice vozidla na mapě.

11.3 Generování a publikování mapových podkladů

V této části bude popsán proces generování a publikování dat knihy jízd do geografického informačního systému.

11.3.1 Volba dat pro publikování

Součástí této diplomové práce je také navržení a vytvoření využití GIS pro publikování mapových podkladů.

Data v systému nabízí několik různých využití - každá zaznamenaná jízda nabízí naměřené body - souřadnice, ze kterých by bylo možné vytvořit datovou vrstvu pro zobrazení v mapách. Nicméně je nutné vzít v potaz jak velký smysl dává vytvářet jednotlivou vrstvu pro každou uloženou trasu. Proto bylo nutné zvolit vhodné využití GIS.

Aktuální pozice vozidel
Prvním způsobem využití GIS bude generování aktuálních pozic vozidel a zobrazení jejich stavu.

V okamžiku generování vrstvy jsou z databáze získány poslední pozice všech aktivních[4)] vozidel a je zjištěn jejich stav - může nabývat čtyř různých hodnot:

- **Aktivní** - Vozidlo je v provozu a odesílá data o své poloze. Místo značí aktuální polohu.
- **Přestávka** - Vozidlo je v provozu (v průběhu cesty) ale je v módu přestávka. Místo značí aktuální polohu vozidla.
- **Neaktivní** - Vozidlo neodeslalo svá data po dobu pěti minut a místo značí její poslední známou pozici.
- **Ukončena** - Vozidlo dokončilo svou jízdu a nyní je nevyužívané na daném místě. Pozice vozidla na mapě je označena poslední sl

[4)] Aktivní - příznak *active* u záznamu vozidla je kladný.

Na základě publikovaného stavu je bod reprezentující vozidlo ve vrstvě jiným způsobem stylován. Ukázka použitého SLD souboru je v příloze této práce.

Historie pohybu
Druhý způsob využití GIS je generování vrstev historie pohybu vozidel. Tímto způsobem je možné zobrazit všechna místa navštívená vozidlem za uplynulé období - v podstatě teplotní mapu (anglicky "*heat map*").

V tomto případě se jednotlivé trasy mohou překrývat a nedokážeme získat informace o jízdách v užším časovém období, jelikož není možné v existujících datech WMS na GIS filtrovat a zobrazovat pouze část z nich. Bylo by nicméně možné publikovat informace po drobnějších částech - například publikovat historii pohybu po týdnech či měsících.

Poté by bylo možné žádat jen o potřebné vrstvy a zobrazovat pouze ty. Tímto způsobem by bylo možné zobrazovat jen zvolené období a tím efektivně načítat jen část dat z GIS.

11.3.2 Proces generování a publikování

Bylo třeba se rozhodnout kdy generovat a publikovat vrstvy do webové aplikace.

Z důvodů optimalizace prostředků aplikačních serverů, na kterých je provozována serverová část webové aplikace a GIS server současně, jsem se rozhodl k pravidelnému publikování vrstev historie pohybu vozidel každou hodinu, což by nemělo neúměrně zatěžovat aplikační server a zároveň to příliš neomezuje uživatele - tyto údaje nejsou jeho primárním zájmem.

Co je pro uživatele důležitější je generování mapových podkladů pro aktuální pozice vozidel. To je také i mnohem jednodušší a tedy i rychlejší než generování historie. Vždy bude třeba jen získat seznam aktuálních pozic vozidel na mapě místo tisíců bodů v minulosti a tedy bude generována každých pět minut.

Pro generování mapových podkladů byla vytvořena třída *GeoserverPublisher*. V této třídě jsou vytvářena data před publikováním a prováděny operace komunikace s GIS.

Pro samotnou REST komunikaci mezi serverovou aplikací a instancí GeoServeru byla použita knihovna *geoserver-manager* ve verzi 1.7.0.

Nejprve je publikován styl ve formě SLD souboru do GIS. Ve zdrojovém kódu serverové aplikace je definován styl pro jednotlivé generované mapové podklady.

Pro vrstvu obsahující data s aktuálními pozicemi automobilů je určen styl ze souboru *logbook_current_position_styles.xml* a styl pro vrstvy s historií pohybu vozidel je v souboru *logbook_vehicle_map_styles.xml*.

Tyto soubory tedy obsahují styl vrstev popsaný SLD a jelikož je uložený přímo ve zdrojovém kódu, není ho aktuálně možné bez nasazení jiné verze aplikace měnit. Je možné nicméně styl upravovat přímo na GeoServeru.

Ze stylovacího XML souboru je následně vytvořen binární soubor a je publikován pod specifikovaným názvem přes třídu *GeoServerRESTPublisher* na GeoServer. Pokud

tam již styl s tímto názvem existuje, je přepsán novým stylem.

Pro vygenerování samotného shapefile souboru byla implementována třída *ShapeFileGenerator*.

Z dat je vytvořen shapefile soubor následujícím způsobem:

Je vytvořen objekt *DefaultFeatureCollection*, do kterého jsou vkládány jednotlivé *feature* - tvary, zde body reprezentující jednotlivé pozice vozidel na mapě.

V případě vrstvy aktuální polohy vozidel jsou vkládány informace o vozidle, trase, řidiči a dalších, které budou na GIS uloženy. Pro vrstvy historie pohybu vozidel je odesíláno méně informací - pouze účel cesty, řidiče a datum.

Názvy těchto parametrů jsou důležité, jelikož musí odpovídat názvům parametrů ve stylovacích souborech i později s nimi bude i pracovat webová aplikace.

11.3.3 Styl vrstev

Jak bylo zmíněno v kapitole 5.2.3, publikovaná data ve vrstvách jsou bez stylu prezentována pouze základními geografickými objekty a data tedy nejsou uživatelsky komfortně použitelná. Také bez aplikovaného stylu nemusí být korektně předaná část informací uživateli.

Ukázka stylu aplikovaného na vrstvu aktuálních pozic vozidel je v příloze této práce.

Jak je ze stylu patrné, pro reprezentaci ikon elementem PointSymbolizer byly použity lokální zdroje. To vyžadovalo vložení potřebných obrázků do složky *data/styles* v kořenové složce instance GeoServer (defaultně na Windows), případně na jiné místo dle způsobu zprovoznění serveru[5)].

Zmíněné obrázky jsou dostupné v elektronické příloze na přiloženém médiu.

11.4 Generování knihy jízd

Jedním z důležitých požadavků na tento systém je také vhodný výstup dat.

Byl zvolen tabulkový formát klasické knihy jízd, kde každý řádek odpovídá jednomu záznamu jízdy a jednotlivé sloupce jsou parametry.

Z technického hlediska bylo třeba realizovat export dat do cílového formátu, v tomto případě XLSX dokumentu. Proto byla v systému vytvořena databázová tabulka pro uložení šablony dokumentu v binární podobě.

V šabloně jsou vyplněna předem připravená pole se zástupnými řetězci, která se poté nahradí za reálné informace - jedná se zejména o informace o zvoleném časovém rozsahu generovaných záznamů. Týká se to také záhlaví tabulky knihy jízd, jelikož exportovaný dokument je lokalizován dle jazyka klienta, který si o dokument zažádal.

[5)] Je možné specifikovat cestu k adresáři nastavením aplikační proměnné při spuštění serveru.

Použitá šablona je součástí obsahu elektronické přílohy na přiloženém médiu.

Samotné generování využívá knihovnu Apache POI. Ve zjednodušené následující ukázce kódu je načtena binární podoba šablony z databáze a je z ní vytvořen objekt typu XSSFWorkbook, se kterým se následně bude při generování pracovat.

```
TemplateDto templateDto =
    templateService.getTemplateDto(templateType);
InputStream templateIs;
if (templateDto != null && templateDto.getData() != null) {
  templateIs = new ByteArrayInputStream(templateDto.getData());
} else {
  throw new LogbookException(ExportException.TemplateNotFound);
}
XSSFWorkbook wb;
try {
  wb = new XSSFWorkbook(templateIs);
} catch (IOException e) {
  throw new LogbookException(ExportException.TemplateCannotOpen);
}
```

Jako první krok jsou nahrazeny zástupné řetězce v šabloně skutečnými informacemi. Do mapy zástupných řetězců jsou doplněny informace, v tomto případě překlady (dle zvoleného jazyka) či datum rozsahu a následně jsou díky metodě *replaceExcelWildcards* i nahrazeny - viz následující zjednodušená ukázka kódu.

```
wildcards.put("#header#",reader.getMessage("logbook.report.header"));
wildcards.put("#dateFromTo#", dateStartEnd);
wildcards.put("#vehicle#",reader.getMessage("logbook.report.vehicle"));
wildcards.put("#name#",reader.getMessage("logbook.report.pathname"));
wildcards.put("#start#",reader.getMessage("logbook.report.dateFrom"));
wildcards.put("#info#",reader.getMessage("logbook.report.disclaimer"));
DocumentUtils.replaceExcelWildcards(sheet, wildcards, templateType);
```

A na následující zjednodušené ukázce textu je zobrazen způsob samotného vkládání jednotlivých záznamů jízd do dokumentu. Jak je možné si všimnout, v cyklu jsou vytvářeny buňky a vkládány do nich informace o jednotlivých cestách. Na začátku těla cyklu je také akce, která může vypadat podivně - kopírování informací v řádku o řádek níže. Je to z důvodu prohlášení (anglicky "disclaimer"), který je umístěn na konci souboru a jelikož předem nevíme kolik záznamů cest se bude generovat, je posouván vždy o řádek níže.

```
for (PathReportDto reportDto : pathReportDtos) {
    DocumentUtils.copyExcelRow(wb, sheet, currentRow, currentRow + 1);
  DocumentUtils.createCell(sheet, currentRow, startValuesCol,
      reportDto.getVehicleHumanName(), csMediumAll);
  DocumentUtils.createCell(sheet, currentRow, startValuesCol + 1,
      reportDto.getName(), csMediumAll);
  DocumentUtils.createCell(sheet, currentRow, startValuesCol + 2,
      reportDto.getDateStart().toString("yyyy-MM-dd HH:mm:ss"),
      csMediumAll);
  currentRow += 1;
}
```

Poté je objekt *XSSFWorkbook* převeden na pole bytů a odeslán ve stejném poža-

davku jako *blob* na frontend, kde si ho uživatel přes *FileSaver* může uložit jako Excel dokument.

12 Webová klientská aplikace

V následující kapitole bude popsána implementace webové klientské aplikace

12.1 Popis

Webovou aplikaci jsem se rozhodl vytvořit ve frameworku Angular. Nepoužil jsem původní verzi AngularJS, ale novou verzi Angular 2+.

Zvolil jsem si tento framework kvůli již existujícímu zájmu a zkušenostem, které jsem chtěl během úsilí na diplomové práci prohloubit. Také oceňuji schopnost frameworku načítat pouze komponenty, které jsou potřeba, což umožňuje efektivnější běh aplikace. Velmi také oceňuji princip routingu, který přináší široké využití a pomáhá při vývoji.

Pro stránku jsem využil zakoupenou licenci *Bootstrap* šablony *SmartAdmin* z online tržiště WrapBootstrap[61]. Použil jsem danou šablonu zejména z důvodu nedostatečných uměleckých sklonů a talentu na vývoj frontend aplikací. Šablona samotná totiž poskytuje zejména design a uživatelské prostředí webové stránky, které je responsivní pro webové prohlížeče na osobních počítačích i na mobilních zařízeních.

Do šablony jsem poté již tvořil samotné komponenty s jednotlivými funkcionalitami.

12.2 Architektura webové klientské aplikace

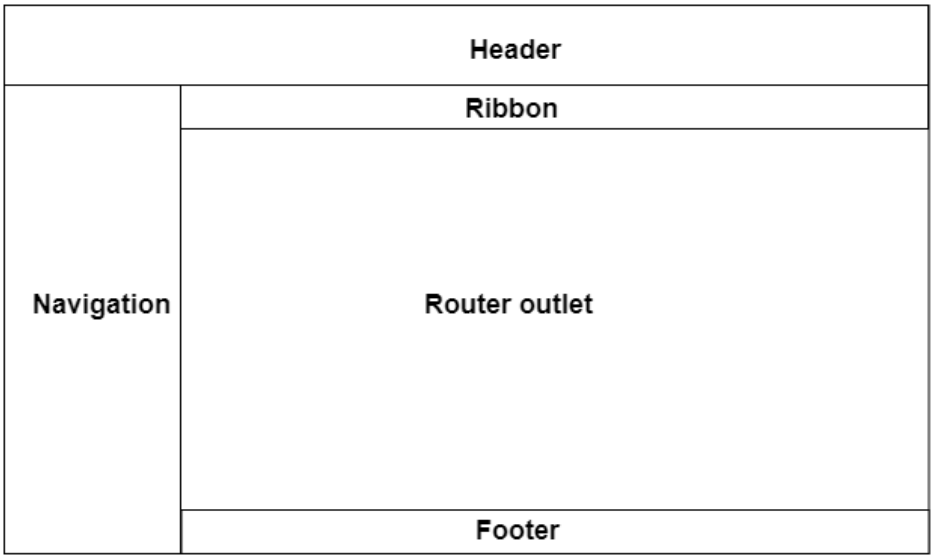
Webová aplikace pro tento systém staví, stejně jako ostatní webové aplikace založené na frameworku Angular, na modulech. Každá část aplikace - Jízdy, Řidiči, Vozidla atd. má dedikovaný svůj vlastní modul.

Jednotlivé moduly v aplikaci sestávají zpravidla z přehledu s tabulkovými daty a z detailu, který je dostupný jen uživatelům s dostatečnými právy. Některé moduly ovšem obsahují data dostupná pro všechny přihlášené uživatele - například mapa aktuálních pozic vozidel.

12.2.1 Rozložení webové aplikace

Uživatelské rozhraní webové klientské aplikace je založené zejména na zmíněné šabloně SmartAdmin, která poskytuje základní stavební prvky aplikace a stará se i o jeho stylování.

Návrh rozložení webové aplikace je následovné - obsah stránky je rozdělen do sekcí *Navigation*, *Header*, *Ribbon*, *Router outlet* a *Footer*:

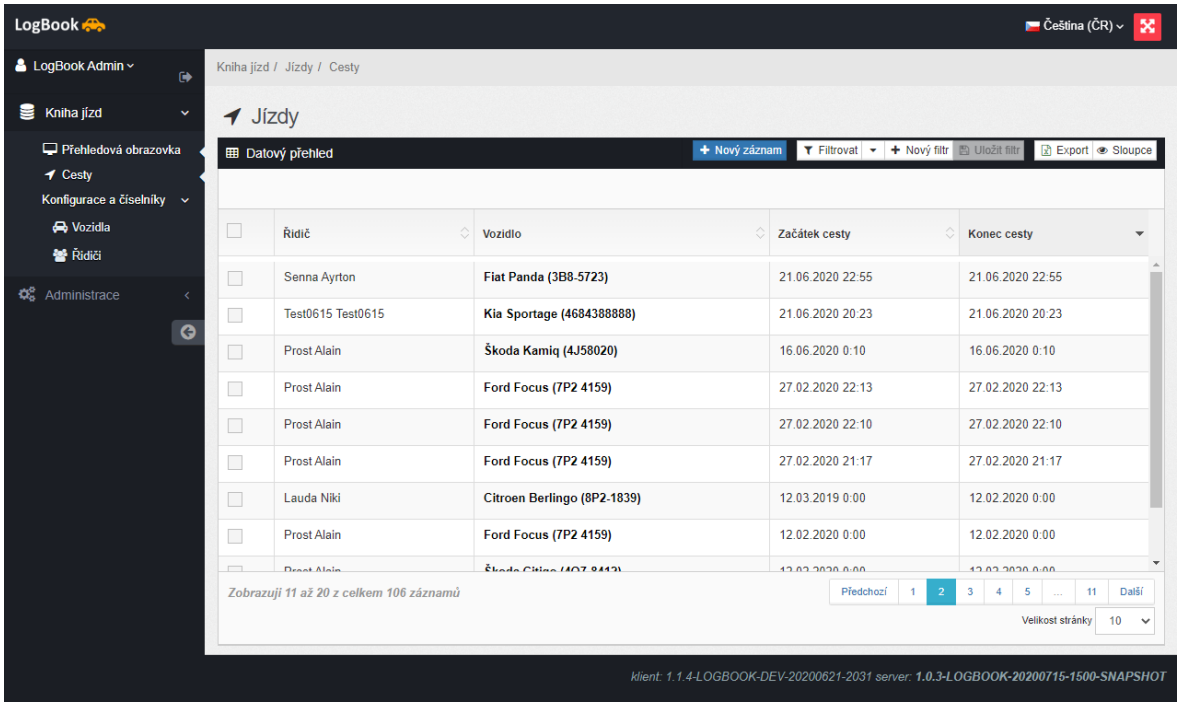


Obr. 12.1 Popis rozložení webové stránky, vlastní

Obsah jednotlivých sekcí

- **Navigation** - umístěno menu aplikace a zobrazení přihlášeného uživatele,
- **Header** - zobrazeno logo aplikace, a možnost změny jazyka,
- **Ribbon** - umístěna drobečková navigace (breadcrumbs navigation) - odkazy na navštívené předchozí stránky,
- **Footer** - zobrazena čísla aktuálních verzí frontendu a backendu,
- **Router outlet** - nejdůležitější část, místo, kde je zobrazována aktuální komponenta, na kterou je aplikace přesměrována.

A na následujícím obrázku 12.2 je již zobrazeno prostředí v implementované aplikaci.



Obr. 12.2 Ukázka rozložení implementované webové aplikace, zdroj vlastní

12.3 Zabezpečení

Do aplikace se mohou přihlásit pouze uživatelé, kteří jsou v systému již zaregistrováni (existují jejich záznamy v tabulce LB_USER)[1)].

Jak již bylo zmíněno v kapitole 12.2, uživatelé mají povolen či zakázán přístup do částí aplikace na základě jejich přiřazených práv.

První vrstva zabezpečení je již samotné skrytí položek v menu na základě vyhodnocení Angular podmínky *ngIf*, která závisí na přiřazených datech uživatele. V případě, že nemá dostatečná oprávnění, zmizí položka z DOMu stránky.

Následně je také zabezpečen přístup díky Angular Router. U definovaných cest v *routingu* je zakázán či povolen přístup do jednotlivých částí aplikace podle rozhraní *canActivate*, kdy definovaná třída *AuthGuard* vyhodnotí, zda uživatel má přiřazené dostatečné oprávnění.

Níže je ukázka kódu zabezpečení aplikace na úrovni modulů v *routingu*. Pokud by se uživatel pokusil přejít na adresu */path* nebo */vehicle*, a neměl dostatečná oprávnění (v ukázce definovaná v parametru "roles"), nebude mu přístup povolen.

```
{ path: 'path',
    loadChildren: 'app/logbook/path/path.module#PathModule',
    data: { roles: ['ADMINISTRATOR', 'LB_READ'] },
    canActivate: [AuthGuard],
},
```

[1)] Aby bylo možné se vůbec přihlásit, je v při nasazování serverové aplikace vytvořen administrátorský uživatel.

```
{ path: 'vehicle',
    loadChildren:
        'app/logbook/vehicle/vehicle.module#VehicleModule',
    data: { roles: ['ADMINISTRATOR', 'LB_CONFIGURATION'] },
    canActivate: [AuthGuard],
},
```

12.4 Tabulkové přehledy

Pro zobrazení dat v tabulkové podobě byl použit plugin DataTables pro jQuery JavaScriptovou knihovnu. Díky jeho použití je možné v aplikaci zobrazovat přehledné tabulkové přehled, které umožňují stránkování, řazení atd.

DataTables umožňují stránkování (pagination) na serveru či přímo v prohlížeči. Bylo zvoleno řešení, kdy se stránkuje na serveru - to znamená, že z webové aplikace je odeslán požadavek na získání dat.

Daný požadavek obsahuje počet záznamů na stránku, pořadí počátečního záznamu a také informace o aktuálním řazení tabulky.

Druhý způsob, tedy stránkování v prohlížeči, získá ze zdroje dat (v tomto případě serverové aplikaci) všechna data do přehledu najednou při inicializaci přehledu a poté se již při přechodu mezi stránkami dat neodesílají další požadavky.

Zvolené řešení stránkování na serveru bylo Vybráno z důvodu snížení délky čekání při prvním zobrazení přehledu a také pro ulehčení prohlížeči - v případě načtení velkého množství dat do DataTable může docházet k problémům s pamětí.

Také je možné v datových přehledech implementovat filtrování dle některého ze sloupců.

12.5 Mapy

V této části bude popsán proces zobrazování map v aplikaci. Bude vysvětlen princip využití knihovny OpenLayers, různé způsoby získání mapových podkladů i samotných dat z mapových serverů a za jakých okolností je použít.

12.5.1 Zobrazení mapy

Pro zobrazení map v aplikaci byla použita JS knihovna OpenLayers.

Mapu na stránku lze vložit na stránku použitím jakéhokoli elementu *div*, například:

```
<div id="map" style="width: 100\%, height: 400px"></div>
```

Mapu pak je nutné vytvořit do daného elementu následujícím způsobem, zjednodušeně popsaným v ukázce kódu:

```
let mapElement = document.getElementById('map');
```

```
this.map = new ol.Map({
           layers: this.layers,
           target: mapElement,
           view: view,
           overlays: [this.overlay],
           controls: ol.control.defaults({
               attribution: false
           }).extend([attribution, scaleLineControl])
       });
```

Následně je možné pracovat s mapou jako s objektem.

12.5.2 Mapové podklady

Definice zdroje mapového podkladu se vkládá do parametru *layers* přímo do kořenového elementu *ol.Map*.

Je více zdrojů, odkud mapové podklady získat. V případě aplikací, které mají dostupnou síť veřejného internetu, je nejjednodušší použít veřejně dostupné servery projektu OpenStreetMap, poté stačí jednoduše definovat OSM jako zdroj mapových podkladů a knihovna OpenLayers si již vše řídí sama - viz. následující ukázka kódu:

```
    new ol.Map({
      layers: [
        new ol.layer.Tile({
          source: new ol.source.OSM(),
          visible: true
        })
     ]});
```

Také je možné se vydat opačným směrem a vytvořit vlastní mapové podklady[2)], které [62]

Nicméně v případě, že je server provozován v uzavřené síti a počítač, který by měl zobrazovat mapy nemá přístup na servery OpenStreetMap nebo jiné online zdroje, je nutné tento problém řešit. V opačném případě by se totiž ve webové aplikaci zobrazovaly pouze obrazce reprezentující trasu bez mapového podkladu.

Nabízí se možnost stáhnout nebo vytvořit kolekci vektorových či rastrových dlaždic[3)]

V tomto případě je ale nutné se rozhodnout a zvolit s jak velkým detailem budou mapová data poskytována. Počet mapových dlaždic a zejména jejich velikost se totiž se zvyšující úrovní zoomu zvyšuje exponenciálně. Zatímco totiž exportovaná vektorová mapa světa s úrovněmi zoomu 4 až 8 (měřítko odpovídá zhruba 1 palec vůči 2000 km - 50 km[4)], obsahuje přes 87 tisíc dlaždic a má velikost okolo 54 MB, nicméně s úrovní zoomu 4 až 10 již pracujeme s téměř 1,4 milionem dlaždic a při úrovni zoomu

[2)] Tento případ se již příliš nedotýká evidence knihy jízd, nicméně lze ho ve specifických případech použít.

[3)] Mapová dlaždice - Použita při vykreslování mapy v prohlížeči sloučením

[4)] Vztaženo na oblast ČR a okolí, jelikož měřítko se kvůli projekci přizpůsobuje a např. na pólech se již výrazně liší a odpovídá zhruba 500 km - 10 km.

14 (zhruba 1000 m na jeden palec), která se dá již využít k zobrazení přehledných map jízd, je použito téměř 358 milionů dlaždic s celkovou velikostí přes 200 GB.

Jedním ze způsobů, jak omezit tato rychle narůstající čísla, je například povolení pouze vybraných úrovní přiblížení - například úrovně zoomu 4,6,8,10,12. Další možností je zobrazovat uživateli jen relevantní část mapových podkladů - v případě elektronické knihy jízd určené menším českým společnostem není pravděpodobně důležité zobrazovat cokoliv mimo střední Evropu. Tímto lze omezit velikost vyžadovaných dat na přijatelnější úroveň.

Jednotlivé dlaždice jsou sdružovány dle úrovně přiblížení a jejich pořadí při namapování na mapu světa do adresářové struktury[5)].

Pokud jsou tedy dostupné obrázky s jednotlivými dlaždicemi pouze pro několik úrovní přiblížení nebo pouze pro určitou oblast mapy, je nutné zamezit, aby uživatel nemohl žádat o jiné mapové podklady, které nejsou dostupné.

Knihovna OpenLayers k tomu poskytuje nástroje a v na následující ukázce kódu je omezena úroveň přiblížení mezi 8 a 16 a zároveň je i omezena oblast na mapě, kterou je možné zobrazit:

```
new ol.View({
    center: [328627.563458, 5921296.662223],
    zoom: 10,
    minZoom: 8,
    maxZoom: 16,
    extent: [-572513.341856, 5211017.966314, 916327.095083,
       6636950.728974]
})
```

Aby byly pro OpenLayers dlaždice viditelné, je nutné je ještě zpřístupnit přes webový server. Pokud by byly dané zdroje (uložené např. ve složce s názvem "maps") ve stejném adresáři jako je i webová aplikace, bylo by možné v tomto případě definovat zdroj mapových podkladů následovně jako relativní cestu:

```
new ol.Map({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM({url: "maps/{z}/{x}/{y}.png"})
      visible: true
    })
 ]});
```

12.5.3 Zobrazování informací z GIS

Data pro zobrazení z GeoServeru lze získat několika způsoby - zde je nejvhodnější využít jedné z poskytovaných služeb - buď WMS (Web Map Services), nebo WFS

[5)] Záleží na umístění dlaždice ve správném adresáři - OpenLayers následně hledá v adresářích podle klíče "z/x/y.png", kde "z"je hodnota aktuálního přiblížení, "x"je pořadí hledaného snímku dle zeměpisné délky a "y"dle zeměpisné šířky.

(Web Feature Service). Obě dvě služby by bylo možné využít a jsou v OpenLayers podporovány. Rozdíl mezi nimi je ve formátu získaných dat.

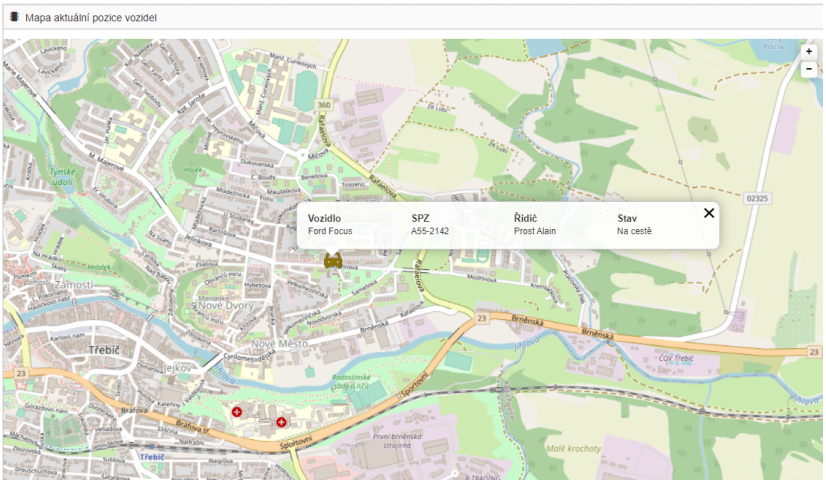
Zatímco pomocí služby WFS lze získat popis mapové vrstvy ve formátu GML (vektorová data), ze služby WMS jsou získány data ve formě obrázků.

Obecně by v tomto případě (zobrazení jednotek až desítek bodů) bylo po přenosové a výpočetní stránce efektivnější získání dat využitím služby WFS (načtení XML souboru o velikosti několika desítek kB), nicméně z důvodu snazší práce s daty[6)] bylo zvoleno použití služby WMS. Při něm GeoServer poskytne již sadu obrázků s průhlednou vrstvou a vykreslenými ikonami, které knihovna OpenLayers pouze umístí na správné místo na mapové podklady.

Nyní tedy mapa zobrazí požadované informace ve formě obrázku a na mapě je možné vidět jednotlivé body zájmu. V případě mapy aktuálních pozic vozidel je reprezentována každá ikona na mapě pozici jednoho vozidla[7)].

Nicméně je nutné zobrazit i ostatní informace, které jsou na GIS uloženy a aktuálně je není možné vidět. Při kliku na místo na mapě je navázána akce knihovny OpenLayers *forEachFeatureAtPixel*, která odešle na službu WMS v GIS dotaz typu *GetFeatureInfo*. Tento požadavek vrátí informace o všech bodech zájmu (feature), které se na daném místě nachází.

V případě získání dat z GIS je poté na mapě vykresleno pop-up okno s parsovanými a nastylovanými informacemi - viz 12.3. Pokud je navrácen objekt bez získaných dat, okno zobrazeno není.



Obr. 12.3 Ukázka zobrazení informací z GIS, zdroj vlastní

Naopak v druhém případě - zobrazení historie jízd vozidel, kdy by na mapě měly být

[6)] Během práce jsem narazil na problém s vykreslováním ikon přes OpenLayers knihovnu a získání dat přes WMS bylo nakonec snažší.

[7)] Nebo více vozidel, pokud se vozidla nachází blízko sebe a v dané úrovni přiblížení se překrývají.

viditelné trasy vozidla za delší časový interval a tedy i velké množství dat (desetitisíce až statisíce pozic vozidla). Nyní se již zobrazuje velké množství zájmových bodů (*feature*) a je tedy již efektivnější využít službu WMS, protože stahování (a zejména vykreslování) dat při použití služby WFS by trvalo neúměrně dlouho.

12.5.4 Zobrazování mapových dat bez GIS

V některých případech nedává smysl jejich publikování na GIS. Jedná se zejména o evidenci velkého množství záznamů, které se následně již nijak nemění. V řešeném systému elektronické knihy jízd se jedná zejména o jednotlivé cesty vozidel. Kdybychom publikovali každou z nich na mapový server, existovalo by jich na mapovém serveru během celého životního cyklu systému až tisíce, což by zbytečně plnilo GIS daty, se kterými se nepracuje. Proto se každá z jízd do GIS nepublikuje.

Protože ale stále chceme trasu jízdy vykreslit do mapy, je možné si poradit jinak. V tomto systému pro to byl využit formát *GeoJSON*. Jedná se o otevřený formát pro reprezentaci a výměnu prostorových dat založený na formátu JSON. GeoJSON objekt může popisovat široké množství různých útvarů - body, lomené čary, polygony. GeoJSON se také může skládat z typů *Feature* a *FeatureCollection*, které obsahují jeden, respektive více geometrických útvarů s definovaným geometrickým typem a dalšími atributy. Takto vytvořená struktura je poté analogická objektu získaným službou WFS z GIS[63].

Tento formát byl využit pro serializaci a přenos dat ze serverové aplikace do klientské webové aplikace, kde ho dokáže knihovna OpenLayers zpracovat a zobrazit.

V serverové aplikaci byla implementace této úlohy vcelku jednoduchá. Byly získány naměřené body cesty - pro jejich získání byl použit pojmenovaný dotaz (anglicky *NamedQuery*) místo překládání Hibernate entit do objektů, což je pro načítání velkého množství stejných objektů z databáze mnohem úspornější a tedy i rychlejší. Ze získaných bodů byly vytvořeny a naplněny objekty popisující GeoJSON formát.

Níže je ukázka sestaveného GeoJSON objektu odesílaného do klientské aplikace. V něm je pro stručnost uložena jen jedna pozice (bod) v poli *"features"*, u jízd evidovaných v systému jsou jinak odesílány stovky bodů. Kromě bodů lze odesílat i jiné typy geografických útvarů.

```
{
  "type" :"FeatureCollection",
  "features" :[{
    "type" :"Feature",
    "geometry" :{
      "type" :"Point",
      "coordinates" :[12.8635783, 50.3063533, 453.3000000 ]
    },
    "properties" :{
      "color" :"#2640F4",
      "sampleDate" :"2020-04-05T15:42:55.000Z",
```

```
      "uuid" :"01110d55-8bc7-4c2e-a568-98fa930a7f65"
    }
  }],
  "projection" :"EPSG:3857"
}
```

Pro řešený systém je tedy výhodou jak nezatěžování GIS, tak i načtení celé trasy najednou (tedy nejsou odesílány žádné další dotazy na data).

13 Testování a zjišťování limitů systému

V této kapitole bude popsán proces provedeného testování systému.

Testování systému je důležitá a nedílná součást vývoje SW. Je nutné během něj zjistit jak problémy způsobené nesprávnou implementací programů, tak i zjištění, zda je výsledný produkt pro uživatele použitelný.

První typ testování, které ověřuje samotnou funkcionalitu a zjišťují problémy bylo při vývoji také použity, nicméně zde budou popisováno nefunkční testování - tedy otestování vlastnosti aplikace jako je pohodlnost použití či výkonnost.

13.1 Uživatelské testy

Cílem uživatelských testů (anglicky *"User testing"*) je zjištění chyb a nedostatků, které byly při vývoji opomenuty či ignorovány, ale mohly by se objevit při časté práci s aplikací.

13.1.1 Sady testů

Pro uživatelské testy byla připravena sada úloh pro mobilní a webovou aplikaci. V obou případech je řešena pouze základní funkcionalita aplikací, která může být běžnými uživateli prováděna. V samotných testech byly uvedeny i specifické hodnoty, které závisely na konfiguraci systému v daný čas, zde byly nahrazeny řetězcem "XYZ".

Testy pro mobilní aplikaci

- Přihlaste se do aplikace s údaji XYZ
- Začněte novou pracovní cestu s vozidlem XYZ a 4963 km na tachometru. Ostatní údaje doplňte sami.
- Udělejte si pauzu na 2 minuty.
- Načerpejte 30 litrů za 27 kč/l na čerpací stanici MOL. Ostatní údaje doplňte sami.
- Zobrazte si aktuální mapu cesty, najděte na mapě vaši aktuální pozici, sledujte ji a zjistěte název nejbližšího bodu zájmu.
- Ukončete cestu se stavem na tachometru 4967 km.
- Změňte Vámi vytvořenou cestu z pracovní na soukromou.

Testy pro webovou aplikaci

- Přihlaste se do aplikace.

- Změňte barvu u vozidla XYZ z černé na modrou.
- Vytvořte nového řidiče se jménem Tomáš Novák a emailem XYZ a heslem XYZ. Ostatní údaje doplňte sami.
- Zjistěte, kde se pohybovalo vozidlo XYZ v den XYZ.
- Vytvořte jízdu vozidlem XYZ dne 24.5.2020 s počátečním stavem tachometru 5020 km a koncovým stavem tachometru 5080 km. Ostatní údaje doplňte sami.

13.1.2 Průběh testování

Testování se zúčastnili čtyři uživatelé. Všichni měli alespoň základní zkušenosti s prací v mobilních a webových aplikacích, dva z nich měli i pokročilé zkušenosti.

Uživatelé před testem neměli bližší informace o systému a lze tedy považovat jejich testovací zkušenost se systémem i za jejich první. Také kromě zadání testů nedostali uživatelé žádné další informace a osoba dohlížející na testování nepomáhala uživatelům. Uživatelé se tak snažili splnit dané úkoly sami.

Nejdříve byla testována mobilní aplikace. Zde uživatelé neměli problém s vytvářením cesty ani s plněním úkolů během ní. Jeden uživatel měl počáteční problém najít a zobrazit si naměřenou trasu ze seznamu ostatních jízd, ale brzy na to přišel. S vyloučením čekání na vytvoření záznamu pauzy zabral tento test uživatelům průměrně dvě minuty a 10 vteřin.

Následně bylo provedeno testování webové aplikace. Všem uživatelům se podařilo upravit barvu vozidla i vytvořit nového řidiče. Největší problém, na který uživatelé naráželi bylo zjištění polohy vozidla v daný den. Způsob zjištění byl možný ze dvou zdrojů - z mapy samotné cesty a případně i z mapy historie pohybu vozidla. Bohužel dvěma uživatelům se tento úkol splnit nepodařilo. Průměrně zabraly testy webové aplikace čtyři minuty a osm vteřin, ale bylo to ovlivněno zejména hledáním polohy vozidla.

Celkově lze považovat výsledek testování za úspěch. Uživatelé neměli problém s drtivou většinou zadaných úkolů a dle svého dojmu aplikaci považovali za dostatečně přehlednou.

13.2 Zátěžové testy

Systém je také nutné podrobit zkouškám, které ukáží použitelnost za různých okolností a ukáží jeho aktuální limity či odhalí nečekané slabiny.

13.2.1 Test odezvy z mobilní aplikace

Test odezvy přímo z aplikace se nejvíce blíží reálnému používání uživatelem a tedy i skutečnému uživatelskému zážitku, na kterém záleží. Jako jedna z nejdůležitějších metrik kladených uživatelem na kvalitu aplikace byla prozkoumán její stav v této aplikaci.

Při práci s aplikací je možné narazit na několik míst, kdy je nutné provést akce na pozadí, které by mohly uživatele zdržet. Jedná se zejména o ukládání a přenos dat.

Dále je v mobilní aplikaci několik požadavků s přenosem dat po síti, ale jen jeden typ, který je velikostně náročnější by mohl kazit uživatelský zážitek. Jedná se o dodatečný přenos kompletní naměřené jízdy, kdy jsou odesílány informace o jízdě samotné včetně seznamu všech souřadnic, kde byla zjištěna pozice vozidla.

Jako benchmark byla zvolena trasa s 2080 body, což odpovídá zhruba tříhodinové cestě. Při přibližně naměřené rychlosti připojení 18 Mbit/s trval přenos dat zhruba 14 vteřin.

Vzhledem k daným okolnostem to lze považovat za dostatečně rychlé a uživatel neměl s aplikací negativní zkušenost.

13.2.2 Zátěžové testování aplikací SoapUI

Z důvodu nemožnosti provést průkazné zátěžové testování přímo ze spuštěné mobilní aplikace[1)] bylo nutné přijít s jiným přístupem.

Proto byl využit program SoapUI, což je jeden z nejpoužívanějších programů pro automatizované testování SOAP a REST rozhraní.

V tomto programu byly vytvořeny programy testování (anglicky "*TestSuite*") a jednotlivé kroky provádění vykonávání testů (anglicky "*TestCase*").

Každý klient vyslal postupně tři požadavky, lišící se pouze náhodně generovanými UUID záznamů a povinných unikátních polí, která měla všechna stejnou délku, aby se velikost odesílaných požadavků nelišila. Byl zvolen požadavek o synchronizaci cesty obsahující data za zhruba dvě hodiny cesty.

Serverová část webové aplikace, na kterou byly posílány požadavky, běžela na lokálním serveru spolu se svou databází. Na stejném zařízení byly spuštěny aplikace SoapUI, ze kterých byly požadavky odesílány. Tímto způsobem by mělo být eliminováno jakékoli případné síťové zpoždění nebo podobné anomálie, které by mohly zkreslit test.

V prvním sloupci tabulky 13.1 je počet současných klientů, kteří komunikovali se serverovou aplikací. Ve druhém sloupci se nachází minimální délka sumy obsluhy požadavků, ve třetím se nachází maximální délka sumy obsluhy požadavků a v posledním

[1)] Vyžaduje to použití velkého množství mobilních telefonů a neposkytuje ani vhodnou platformu pro sběr výsledků.

sloupci se nachází aritmetický průměr sum dob obsluh požadavků.

Tab. 13.1 Test zátěže serverové aplikace

Počet klientů	Min. délka	Max. délka	Průměrná. délka
5	8,7 s	14,3 s	10,7 s
10	9,4 s	19,7 s	13,6 s
30	13,4 s	38,1 s	18,4 s
50	22,1 s	52,2 s[2)]	31,2 s

Ze zjištěných údajů testu zátěže lze určit, že systém by měl bez větších potíží zvládat obsluhu požadavků nižších desítek zařízení (mobilních aplikací - vozidel), která by posílala nadprůměrné množství dat. Při obsluze padesáti požadavků již bylo naráženo na technické limity, které by se daly v budoucím rozšíření vyřešit.

Je možné odhadovat, že při konstantní větší zátěži by bylo nutné přijít s jiným přístupem ke zpracování požadavků.

13.3 Test využití dat

Jednou z částí, na kterou se nesmí zapomenout, je i optimalizace dat odesílaných na server. Proto bylo experimentálně zjišťováno kolik dat je během monitorování využito.

Největší část objemu přenášených dat v mobilní aplikaci sestává zejména z informací, které popisují pozice vozidla, už jen z toho důvodu, že jich je nejvíce - ke každé zaznamenané cestě jich jsou obecně stovky až tisíce[3)].

Po provedených optimalizaci bylo zjištěno, že při dvouhodinové trase je odesláno zhruba 5,5 MB dat.

Tato hodnota byla zjištěna jak pomocí aplikace třetí strany *GlassWire Data Usage Monitor*, tak i využitím nativního nástroje na platformě Android *Použití dat*. Zjištěná hodnota by měla obsahovat data z aplikace včetně síťových hlaviček, tedy reálné množství účtovaných dat.

Ačkoliv již byly implementovány během vývoje optimalizace (popsáno v 10.2.2), nabízí se další způsoby jak zmenšit množství odesílaných dat:

- **Jiný způsob serializace** - Ačkoliv formát JSON je datově velmi úsporný, bylo by možné najít a využít lepší - možné vyzkoušet například formát *MessagePack*.
- **Komprese** - Vzhledem k samotné nátuře dat, text obsahující souřadnice, lišící se pouze v části informace, bylo by možné data odesílat ve značně komprimované

[2)] V některých případech nastávaly problémy s technickými prostředky - nedostatek paměti, selhávající transakce s databází.

[3)] Pro dvouhodinovou jízdu lze uvažovat zhruba 1400 naměřených pozic.

podobě.

- **Další zkrácení jmen parametrů** - Lze odhadovat, že při další rozumné[4)] optimalizaci objemu odesílaných dat by bylo možné ušetřit až dalších 60 % velikosti dat.

- **Zvýšení periody odesílání dat** - Aktuálně je navázána frekvence procesu synchronizace na frekvenci procesu získávání dat. Bylo by možné synchronizaci řešit ve vlastním frekvenci a ačkoliv by již aktuální poloha na serveru neodpovídala tak přesně skutečné aktuální poloze vozidla, úspora dat by byla citelná. V případě zvýšení periody na jednu minutu by se jednalo přibližně o 85 % zmenšení množství odesílaných dat.

[4)] Bylo by možné přejmenovat parametry v objektech až na jednopísmenné názvy, ale za cenu přehlednosti.

14 Zhodnocení řešení a možné rozšíření

V poslední kapitole této diplomové práce je zhodnocena úspěšnost řešení komplexního systému elektronické knihy jízd a jeho porovnání s ostatními již existujícími řešeními. Také bylo uvažováno nad možností způsobu monetizace vytvořeného systému. Dále jsou popsány další možné a vhodné rozšíření, které byly zjištěny na základě zpětné vazby a v průběhu procesu vývoje.

14.1 Zhodnocení vytvořeného řešení

V rámci práce byly implementovány všechny části systému - webová aplikace, včetně její klientské webové a serverové části a mobilní aplikace.

Bohužel ne všechny nuance implementovaného systému bylo možné předem při návrhu objevit a v systému nebylo vše vytvořeno podle plánu. Je tedy nutné přiznat, že velké množství věcí by se v případě implementace podobného řešení či pokračující práce na tomto, řešila jiným způsobem. Také část funkcionalit se kvůli časové tísni do systému nedostala.

V každém případě, systém byl otestován a funguje zamýšleným způsobem a splňuje požadavky, které na něj jsou kladeny.

14.2 Porovnání řešení s existujícími řešeními

Žádná ze zkoumaných existujících řešení neumožňuje provozování systému ve formě webové aplikace na vlastních výpočetních prostředcích, tedy v tomto případě má vytvořený systém velkou výhodu[1)].

Jako velkou výhodu oproti porovnávaným řešením lze uvést odpadnutí potřeby pořizovat a instalovat fyzickou jednotku do vozidla. Také ostatní zkoumané řešení neumožňují provozování služby ve vlastních HW prostředích a tedy určitým způsobem citlivé údaje (informace o zaměstnancích společnosti, ale zejména aktuální pozice firemních vozidel) jsou v držení třetí strany.

Co se týče porovnání jednotlivých parametrů - jako další výhodu lze rozhodně uvést cenu. Některá z řešení jsou velice nákladná a zakoupení desítek fyzických monitorovacích jednotek spolu s pravidelnou platbou za SIM karty může některé zákazníky odrazovat. Zejména ty, kteří by velkou část funkcionalit nevyužili.

Nicméně řešení navržené v této práci má i své nevýhody. Na rozdíl od fyzických jednotek, které vykonávají svou činnost bez vstupu uživatele, navržené řešení využívá k získání dat mobilní aplikaci a ta vyžaduje spolupráci uživatele. Ačkoliv je možné

[1)] Zejména pokud bude v budoucnu řešen výrazněji princip lokalizace dat.

vynutit korektní používání aplikace například interními předpisy, pokud by uživatel skutečně nechtěl aby byla jeho cesta monitorována, nemusel by si aplikaci spouštět[2)].

Toto řešení také neumožňuje instalaci dodatečných modulů, které by rozšiřovaly jeho možnosti a zvyšovaly hodnotu systému - např. systém GPS Dozor umožňuje instalaci těch rozšiřujících modulů do vozidla řešící i jiné záležitosti (např. zastavení vozidla na dálku, kontrola stavu paliva v nádrži, ověření identity řidiče a další).

Navíc bohužel nelze dosáhnout stejné přesnosti GPS dat z mobilních zařízení jako z dedikovaných GPS modulů fyzických jednotek, nicméně pro přibližné určení aktuální polohy toto řešení stačí.

Tyto veškeré nevýhody by se nicméně mohly řešit výrobou vlastních fyzických monitorovacích jednotek, případně nabízením nakonfigurovaných již existujících jednotek na trhu. Bylo by to možné, jelikož tento řešený systém poskytuje API pro přijímání dat.Tímto by nicméně již byly popírány principy, na základě kterých tento systém staví - tedy levné a dostupné řešení.

14.3 Monetizace systému

V předchozí kapitole byly porovnány vlastnosti ostatních řešení elektronické knihy jízd již nabízených na trhu s řešením vytvořeným v rámci této diplomové práce.

Vzhledem k faktu, že systém bude hostovaný na výpočetních prostředcích zákazníka a autor systému nad ním následně nemá kontrolu, lze považovat za nejvýhodnější řešení se inspirovat business modelem společnosti JetBrains, která nabízí zejména vývojářské prostředí jako například Intellij IDEA, PhpStorm či CLion.

Tato společnost nabízí své softwarové produkty pod modelem "*perpetual fallback license*"[3)], což znamená že zákazník, který si zakoupí licenci (reálně předplatné), získává po určitou dobu zdarma nové verze aplikace. Po vypršení zmíněné doby mu aplikace v dané (nejnověji aktualizované) verzi zůstane k používání. Pokud bude chtít získat novou verzi (kterou aktuálně nemá), musí si znovu předplatné licence zakoupit.

Při využití tohoto způsobu kapitalizace by bylo možné nabízet systém za přijatelnou částku a vzhledem k velkému potenciálu pro rozšíření systému by zároveň existovaly důvody pro zákazníky k předplacení licence na další období pro získání nových funkcí.

[2)] Pokud by ale zaměstnanec skutečně záměrně jízdu vozidlem nezaregistroval, neodpovídal by počet ujetých kilometrů u vozidla

[3)] Podrobně popsáno na oficiálních stránkách společnosti: sales.jetbrains.com/hc/en-gb/articles/207240845

14.4 Rozšíření řešení

Na základě zjištěných poznatků a přijatých návrhů při testování systému byla definována některá rozšíření systému. Nápady, návrhy a poznatky byly rozděleny do skupin, které by rozšiřovaly celý systém a do částí, které by se týkaly pouze jednotlivé komponenty systému.

14.4.1 Rozšíření celého systému

Úprava pro nasazení systému jako službu
Ačkoliv systém byl navržený pro využití pouze jednou firmou či autoritou, bylo by možné jednotlivé části upravit, aby se dal systém využívat jako služba dostupná více zákazníkům.

V jedné databázi (serverové aplikace) by pak sice byla uloženy data více zákazníků, ale sami uživatelé by mohli vidět a upravovat pouze relevantní data.

Vytvoření nástrojů pro administraci GDPR - Bylo by vhodné vytvořit nástroje pro zjednodušení práce správce osobních údajů. Tyto nástroje by poskytovaly následující funkcionality:

- **Zjištění informací vedených o uživateli** - Poskytnutí nástrojů ke komfortnímu zjištění všech informací vedených o uživateli,
- **Vymazání informací na žádost uživatele** - Systém by umožňoval mazat informace bez nutnosti zásahu přes databázi.

Rozšíření podpory pro možnost použití ve více kulturách - Ačkoliv je aplikace lokalizována do více jazyků (češtiny a angličtiny), v některých ohledech je limitována - vzdálenost je měřena pouze v kilometrech a jako měna je aktuálně využívána pouze Kč.

Evidence tankovacích karet - V průběhu práce byl získán poznatek, že by bylo vhodné implementovat i správu tankovacích karet a sledování utracených prostředků.

14.4.2 Rozšíření funkcionality jednotlivých částí

Mobilní aplikace

- **Dokončení připojení pomocí Bluetooth k automobilům** - Ačkoliv byl proveden průzkum a zkoušeny způsoby komunikace telefonu s
- **Podpora více uživatelů na jednom zařízení** - Zprovoznění možnosti evidovat trasy knihy jízd více uživatelů na stejném zařízení.

- **Dokončení verze aplikace pro platformu iOS.** - Ačkoliv je aplikace vyvíjena v multiplatformním frameworku, je třeba část kódu specificky implementovat pro danou platformu. Je také nutné danou aplikaci sestavit a otestovat, což u platformy iOS je možné pouze se zařízeními od společnosti Apple.
- **Hlídání bezpečnostních přestávek** - Aplikace by mohla na základě nakonfigurovaných dat hlídat a upozorňovat řidiče na potřebu bezpečnostních přestávek. Zároveň by porušení i zaznamenávala a v případě nedodržení odeslala informaci na server.
- **Výpočet a ukládání kontrolního součtu trasy** - Pro zajištění nepovolené úpravy trasy cesty by bylo možné počítat kontrolní součet ze souřadnic, času a jiných metadat. Při úpravě jakékoli informace by pak vypočítaná hodnota neodpovídala a bylo by zřejmé, že existuje nějaký problém.
- **Dodatečná datová optimalizace** - Během práce bylo zjištěno, že i přes proběhlou optimalizaci odesílaných dat klientem je prostor pro další zlepšení. Bylo by možné serializovat a odesílat data úsporněji. Také by bylo možné implementovat adaptivní zjišťování pozice, kdy by nebyla aktuální pozice měřena v pevně daném intervalu, ale na základě aktuální rychlosti.

Webová aplikace

- **Hlídání dodržování zákonů či interních předpisů** - Systém by mohl na základě nastavení hlídat různé zákony - například maximální rychlost vozidla (ačkoliv by bylo třeba určit, kde se dané vozidlo nachází, zda na dálnici či ve městě atd.). Systém by také mohl varovat před blížícím se termínem technické kontroly vozidla či před vypršením platnosti dálniční známky - tyto informace jsou již u vozidel evidovány, ale uživatel není o nich aktivně upozorňován.
- **Generování souhrných reportů** - Bylo by možné implementovat generování reportů statistik, které lze z dat v systému zjistit - například vytíženost jednotlivých vozidel, řidiče s nejvíce najetými kilometry atd.
- **Možnost importu tras z jiných aplikací** - Vyžadovalo by pouze menší úsilí implementovat do aplikace možnost importovat trasy ze souborů exportovaných z geografických aplikací, např. QGIS, ArcGIS či některých proprietárních aplikací. Bylo by třeba implementovat rozparsování daných formátů a doplnění potřebných metadat pro uchování v systému.
- **Výměna dat pomocí SOAP** - Pro rozšíření budoucích schopností systému by bylo i vhodné implementovat webové služby pro komunikaci přes protokol SOAP.

Umožňovalo by to lepší standardizaci a univerzálnost komunikace zejména při výměně dat s více stranami.

ZÁVĚR

Hlavním cílem této práce bylo popsat doporučené technologie, metody a postupy pro tvorbu systému elektronické knihy jízd.

V teoretické části práce jsem zmínil různé technologie, které jsem pro tvorbu systému použil. Také jsem prozkoumal a porovnal vlastnosti existující produktů, řešících různými způsoby evidenci knihy jízd. Dále jsem analyzoval legislativní omezení a možné budoucí změny, které by mohly tuto problematiku ovlivnit.

V praktické části jsem poté navrhl a implementoval systém elektronické knihy jízd a popsal jeho klíčové části, zejména jsem kladl důraz na využití mapových dat. Také jsem uvedl problémy, na které jsem narazil a způsob jejich řešení. Provedl jsem testování systému, zjistil jeho limity, zhodnotil řešení a porovnal ho s ostatními existujícími produkty.

Při psaní textu této práce jsem si uvědomoval, že bylo řešeno tolik různých funkcionalit, že jsem narážel na limity diplomové práce a popisoval jsem tedy zejména ty nejdůležitější a nejzajímavější části.

K tématu mě motivovala možnost vytvořit produkt, který by se doopravdy dal v budoucnu plnohodnotně používat a rozvíjet a byl by o něj zájem. Osobně považuji toto snažení za úspěch.

Dalším hnacím motorem pro mě byl zájem se zdokonalit v technologiích, které jsem při implementaci systému používal. To se povedlo a zjistil jsem nové způsoby jak řešit různé problémy, které mohou při mé další pracovní kariéře vyvstávat, ale také jsem si uvědomil limity mých vědomostí a našel důvod je dále rozvíjet.

Jako jednu ze zajímavějších a příjemnějších částí práce bych rád uvedl prováděné experimenty pro testování aplikace a nastavení těch nejoptimálnějších proměnných využívaných v systému.

Chtěl bych také uvést, že byla škoda nepokračovat v práci a dále nerozvíjet již zmíněný implementovaný systém elektronické knihy jízd, který ačkoliv plní svůj účel a funguje, má velký potenciál rozšířit svou hodnotu. I pro to bylo během práce sepsán seznam vhodných rozšíření systému, které by hodnotu systému ještě více zvýšily.

Rád bych na konec práce vyzdvihl fakt, že řešený systém elektronické knihy jízd má soubor dohromady unikátních vlastnosti, které nenabízí žádný z jiných existujících produktů na trhu, od kterých se tímto odlišuje. Zároveň je dostatečně intuitivní a může být pro zákazníky v budoucnu zajímavý a některé z nich přesvědčit k používání místo využívání papírové knihy jízd.

SEZNAM POUŽITÉ LITERATURY

[1] KANDLEROVÁ, Kateřina. Jak správně vést knihu jízd?*Portál POHODA: Informace pro účetní a podnikatele* [online]. 2015, 25. 11. 2015 [cit. 2020-04-11]. Dostupné z: portal.pohoda.cz/pro-podnikatele/uz-podnikam/jak-spravne-vest-knihu-jizd/

[2] *GPS Dozor: Satelitní sledování vozidel*, 2020 [online]. [cit. 2020-05-10]. Dostupné z: www.gpsdozor.cz

[3] *AutoGPS: Elektronická kniha jízd*, 2020 [online]. [cit. 2020-05-11]. Dostupné z: www.auto-gps.eu

[4] *Kniha jízd, Cestovní příkazy, Automapa: AUTOPLAN, KROB sofware s.r.o.* [online]. [cit. 2020-06-22]. Dostupné z: www.autoplan.cz

[5] *Elektronická kniha jízd online: Generátor-knihy-jízd.cz v2.0* [online]. [cit. 2020-04-18]. Dostupné z: www.generator-knihy-jizd.cz

[6] ČMIEL, Vladislav. JE KNIHA JÍZD POVINNÁ ?*Daňový portál: Vedení účetnictví a účetní služby* [online]. 2011, 21. 12. 2011 [cit. 2020-06-02]. Dostupné z: www.cmiel.cz/blog/je-kniha-jizd-povinna/

[7] Pokyn GFŘ D-22. In:*Finanční zpráva* [online]. Praha, 2015, 6. 2. 2015 [cit. 2020-07-20]. Dostupné z: www.financnisprava.cz/assets/cs/prilohy/d-zakony/Pokyn_GFR_D-22.pdf

[8] HEJKRLÍK, Lukáš. *Kniha jízd z pohledu daně z příjmu a DPH.* Vedení účetnictví České Budějovice, daňová evidence [online]. 2020, 23.01.2020 [cit. 2020-05-04]. Dostupné z: www.benecb.cz/novinky/kniha-jizd/

[9] Školení řidičů z povolání a řidičů referentů. Jaké jsou rozdíly?*Školení BOZP* [online]. 2018, 19. 1. 2018 [cit. 2020-04-20]. Dostupné z: www.skolenibozp.cz/aktuality/rozdil-ve-skoleni-ridicu-z-povolani-a-ridicu-referentu/

[10] ČESKO. fragment f2312577 nařízení vlády č. 168/2002 Sb., nařízení vlády, kterým se stanoví způsob organizace práce a pracovních postupů, které je zaměstnavatel povinen zajistit při provozování dopravy dopravními prostředky. In: Zákony pro lidi.cz [online]. © AION CS 2010-2020 [cit. 4. 8. 2020]. Dostupné z:www.zakonyprolidi.cz/cs/2002-168f2312577

[11] Co je GDPR a jak bude aplikováno v Česku.*GDPR: Obecné nařízení o ochraně osobních údajů - prakticky* [online]. [cit. 2020-06-22]. Dostupné z: www.gdpr.cz/gdpr/co-je-gdpr/

[12] KOMÍNKOVÁ, Magda. Jak vznikalo nařízení o ochraně osobních údajů (GDPR)?*Euroskop.cz: Zpravodajství* [online]. 2018, 27. 3. 2018 [cit. 2020-04-19]. Dostupné z: www.euroskop.cz/9047/30715/clanek/jak-vznikalo-narizeni-o-ochrane-osobnich-udaju-gdpr/

[13] Osobní údaje.*GDPR: Obecné nařízení o ochraně osobních údajů - prakticky* [online]. 2017, 20. 2. 2017 [cit. 2020-04-20]. Dostupné z: www.gdpr.cz/gdpr/heslo/osobni-udaje/

[14] Citlivé osobní údaje.*GDPR: Obecné nařízení o ochraně osobních údajů - prakticky* [online]. 2017, 20. 2. 2017 [cit. 2020-04-20]. Dostupné z: www.gdpr.cz/gdpr/heslo/citlive-osobni-udaje/

[15] Jaké sankce hrozí firmám, které budou GDPR ignorovat. GDPR: Obecné nařízení o ochraně osobních údajů - prakticky [online]. [cit. 2020-05-16]. Dostupné z: www.gdpr.cz/gdpr/sankce/

[16] ECall based on 112 in Europe: HeERO 2 project final event.*HeERO* [online]. [cit. 2020-04-20]. Dostupné z: www.heero-pilot.eu/view/ecs/media/news/20141121.html

[17] BARTÁK, Petr. Jak funguje nouzové volání eCall: Pozor na čudlík!*Auto.cz* [online]. 2019, 19. 5. 2019 [cit. 2020-04-19]. Dostupné z: www.auto.cz/jak-funguje-nouzove-volani-ecall-pozor-na-cudlik-129228

[18] GIBSON, Dean. What is eCall?: Automated emergency call technology explained.*Auto Express: New and Used Car Reviews, News Advice* [online]. 2020, 25. 3. 2020 [cit. 2020-04-21]. Dostupné z: www.autoexpress.co.uk/car-tech/101706/what-is-ecall-automated-emergency-call-technology-explained

[19] ČSN EN 15722. *Inteligentní dopravní systémy - eSafety - Minimální soubor dat pro eCall*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2015, Třídící znak 01 8461.

[20] What is ERA-GLONASS?*Everything RF: RF Components Test Equipment* [online]. 2018, 19. 9. 2018 [cit. 2020-04-25]. Dostupné z: www.everythingrf.com/community/what-is-era-glonass

[21] O projektu HeERO.*HeERO* [online]. [cit. 2020-05-10]. Dostupné z: www.heero-pilot.eu/view/cs/heero/vision.html

[22] BOWMAN, Courtney. Data Localization Laws: an Emerging Global Trend.*JURIST: Legal News Commentary* [online]. 2017, 6. 1. 2017 [cit. 2020-05-07]. Dostupné z: www.jurist.org/commentary/2017/01/Courtney-Bowman-data-localization/

[23] CHANDER, Anupam a Uyên P. LÊ. Data Nationalism.*Emory Law Journal* [online]. 2015 [cit. 2020-05-07]. Dostupné z: law.emory.edu/elj/content/volume-64/issue-3/articles/data-nationalism.html

[24] BOWMAN, Courtney. A Primer on Russia's New Data Localization Law.*Proskauer Rose LLP: Privacy Law Blog* [online]. 2015, 27. 8. 2015 [cit. 2020-05-07]. Dostupné z: privacylaw.proskauer.com/2015/08/articles/international/a-primer-on-russias-new-data-localization-law/

[25] SHAFTAN, Vera. Russian Data Localization law: now with monetary penalties.*Norton Rose Fulbright: Global law firm* [online]. 2019, 20. 12. 2019 [cit. 2020-05-08]. Dostupné z: www.dataprotectionreport.com/2019/12/russian-data-localization-law-now-with-monetary-penalties/

[26] EU to ban data localisation restrictions as ambassadors approve deal on free flow of data.*The European Council: Consilium* [online]. 2018, 29. 6. 2018 [cit. 2020-05-08]. Dostupné z: www.consilium.europa.eu/en/press/press-releases/2018/06/29/eu-to-ban-data-localisation-restrictions-as-ambassadors-approve-deal-on-free-flow-of-data/

[27] STUPP, Catherine. Negotiators reach quick agreement on law banning data localisation.*EURACTIV: EU news and policy debates across languages* [online]. 2018, 20. 6. 2018 [cit. 2020-05-09]. Dostupné z: www.euractiv.com/section/data-protection/news/negotiators-reach-quick-agreement-on-law-banning-data-localisation/

[28] *17 Popular Java Frameworks [2019 edition]: Pros, cons, and more* In: Raygun [online]. 2018-04-14 [cit. 2020-03-01]. Dostupný z WWW: `www.raygun.com/blog/popular-java-frameworks/`.

[29] *Overview of Spring Framework* [online]. 2017 [cit. 2020-05-22]. Dostupné z: docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/overview.html

[30] RITCHIE, Christopher. *WildFly Configuration, Deployment, and Administration*. 2. Birmingham: Packt Publishing, 2014. ISBN 1783286237.

[31] MASSÉ, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. 1. Sebastopol, USA: O'Reilly Media, 2011. ISBN 1449319904.

[32] MARTIN, Daly a Douglas CROCKFORD. The application/json Media Type for JavaScript Object Notation (JSON). *Internet Engineering Task Force (IETF)* [online]. [cit. 2020-06-11]. ISSN 2070-1721. Dostupné z: www.ietf.org/rfc/rfc4627.txt

[33] *Angular* [online]. [cit. 2020-05-10]. Dostupné z: angular.io

[34] Version Support Status.*AngularJS: Superheroic JavaScript MVW Framework* [online]. [cit. 2020-05-11]. Dostupné z: docs.angularjs.org/misc/version-support-status

[35] DARWIN, Pete Bacon. Stable AngularJS and Long Term Support.*Angular Blog* [online]. 2018, 26. 1. 2018 [cit. 2020-05-11]. Dostupné z: blog.angular.io/stable-angularjs-and-long-term-support-7e077635ee9c

[36] Angular versioning and releases.*Angular* [online]. [cit. 2020-05-10]. Dostupné z: angular.io/guide/releases

[37] Upgrading from AngularJS to Angular.*Angular* [online]. [cit. 2020-05-11]. Dostupné z: angular.io/guide/upgrade

[38] Deprecated APIs and features.*Angular* [online]. [cit. 2020-05-11]. Dostupné z: angular.io/guide/deprecations

[39] Introduction to Angular concepts.*Angular* [online]. [cit. 2020-05-11]. Dostupné z: angular.io/guide/architecture

[40] DAITYARI, Shaumik. *Angular vs React vs Vue: Which Framework to Choose in 2020.* CodeinWP [online]. 2020, 7. července 2020 [cit. 2020-07-11]. Dostupné z: www.codeinwp.com/blog/angular-vs-vue-vs-react

[41] May 2020 Web Server Survey.*Netcraft: Internet Research, Cybercrime Disruption and PCI Security* Services. [online]. 2020, 26. 5. 2020 [cit. 2020-06-03]. Dostupné z: news.netcraft.com/archives/2020/05/26/may-2020-web-server-survey.html

[42] *Apache: HTTP Server Project* [online]. [cit. 2020-05-13]. Dostupné z: httpd.apache.org/

[43] What is XAMPP? and How to Install XAMPP on Local Computer? *WPBlogX* [online]. 2017, 16. 10. 2017 [cit. 2020-07-20]. Dostupné z: www.wpblogx.com/what-is-xampp/

[44] SNIDER, Ed. *Mastering Xamarin.Forms: Build Rich, Maintainable, Multi-platform, Native Mobile Apps with Xamarin.Forms*. 2. Birmingham: Packt Publishing, 2018. ISBN 1788297342.

[45] VANIUKOV, Slava. Top Frameworks for Android App Development: Overview Of Top.*Level Up Coding* [online]. 2020, 24. 3. 2020 [cit. 2020-06-11]. Dostupné z: levelup.gitconnected.com/top-frameworks-for-android-app-development-overview-of-top-dc95fcde75a0

[46] BRITCH, David, Craig DUNN a Justin JOHNSON. What is Xamarin? *Microsoft* [online]. 2020, 28. 5. 2020 [cit. 2020-06-10]. Dostupné z: docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[47] FISCHER, Roman. Xamarin: Úvod do Xamarin Forms.*Skeleton Software: Informační systémy, webové aplikace, mobilní aplikace* [online]. 2017, 1. 3. 2017 [cit. 2020-07-13]. Dostupné z: www.skeleton.cz/uvod-do-xamarin-forms

[48] BRITCH, David, Craig DUNN, Nick SCHONNING a John OSBORNE. The Model-View-ViewModel Pattern.*Microsoft* [online]. 2017, 7. 8. 2017 [cit. 2020-06-15]. Dostupné z: docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[49] HERMES, Dan a Nima MAZLOUMI. *Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals.* 1. New York City: Apress, 2019. ISBN 978-1-4842-4030-4.

[50] Základy kartografie: Kartografie se zabývá tvorbou map.*Maturitní otázky: Přes 360 kvalitně vypracovaných maturitních otázek* [online]. [cit. 2020-06-14]. Dostupné z: www.vysokeskoly.cz/maturitniotazky/zemepis/zaklady-kartografie

[51] Shapefiles.*Esri: GIS Mapping Software, Location Intelligence Spatial Analytics* [online]. [cit. 2020-06-20]. Dostupné z: doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm

[52] *GeoServer* [online]. [cit. 2020-03-08]. Dostupné z: www.geoserver.com

[53] YOUNGBLOOD, Brian a Stefano IACOVELLA. *Geoserver: Beginner's Guide*. 1. Birmingham: Packt Publishing, 2003. ISBN 9781849516686.

[54] *OpenStreetMap* [online]. [cit. 2020-05-15]. Dostupné z: www.openstreetmap.org

[55] History of OpenStreetMap. *OpenStreetMap Wiki* [online]. [cit. 2020-07-11]. Dostupné z: wiki.openstreetmap.org/wiki/History_of_OpenStreetMap

[56] Web Map Service.*OGC: The Home of Location Technology Innovation and Collaboration* [online]. [cit. 2020-06-24]. Dostupné z: www.ogc.org/standards/wms

[57] FARKAS, Gabor. *Mastering OpenLayers 3*. 1. Birmingham: Packt Publishing, 2016. ISBN 1785281003.

[58] SQL Lite [online]. [cit. 2020-06-25]. Dostupné z: sqlite.org/index.html

[59] Distinctive Features Of SQLite. *SQLite* [online]. [cit. 2020-06-25]. Dostupné z: www.sqlite.org/different.html

[60] Xamarin.FormsMessagingCenter. *Microsoft Docs* [online]. 2019, 08. 11. 2019 [cit. 2020-06-16]. Dostupné z: docs.microsoft.com/cs-cz/xamarin/xamarin-forms/app-fundamentals/messaging-center

[61] SmartAdmin - Responsive WebApp. *Bootstrap Templates on WrapBootstrap* [online]. [cit. 2020-03-14]. Dostupné z: wrapbootstrap.com/theme/smartadmin-responsive-webapp-WB0573SK0

[62] *MapTiler: Mapping platform for quick publishing of zoomable maps online*,2020 [online]. [cit. 2020-05-19]. Dostupné z: www.maptiler.com/

[63] MARTIN, Daly, Howard BUTLER, Allan DOYLE, Sean GILLIES, Stefan HAGEN a Tim SCHAUB. The GeoJSON Format. *Internet Engineering Task Force (IETF)* [online]. [cit. 2020-06-03]. ISSN 2070-1721. Dostupné z: tools.ietf.org/html/rfc7946

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
DB	Databáze
DPH	Daň z přidané hodnoty
DOM	Document Object Model
GDPR	General Data Protection Regulation
GIS	Geographic Information System
GKJ	Generátor Knihy Jízd
GNSS	Global Navigation Satellite System
GPRS	General Packet Radio Service
GUI	Graphical user interface
HTTP	Hypertext Transfer Protocol
IoC	Inversion of Control
JS	JavaScript
JSON	JavaScript Object Notation
MVVM	Model-View-ViewModel
OGC	Open Geospatial Consortium
POM	Project Object Model
PHM	Pohonné hmoty a mazadla
REST	Representational State Transfer
SIM	Subscriber Identity Module
SLD	Styled Layer Descriptor
SOAP	Simple Object Access Protocol
SPZ	Státní Poznávací Značka
VIN	Vehicle Identification Number
XAML	Extensible Application Markup Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I. PŘÍKLAD STYLOVACÍHO SOUBORU SLD

Níže je zobrazen jeden z použitých souborů ke stylování vrstev na GeoServeru.

Zobrazované ikony, které je nutné umístit do využívaného GeoServeru, jsou umístěny v elektronické příloze na přiloženém médiu.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
            xsi:schemaLocation="http://www.opengis.net/sld
                StyledLayerDescriptor.xsd"
            xmlns="http://www.opengis.net/sld"
            xmlns:ogc="http://www.opengis.net/ogc"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>default_point</Name>
    <UserStyle>
      <Title>Default Point</Title>
      <Abstract>A sample style that draws a point</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <Name>CarOnline</Name>
          <ogc:Filter>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>status</ogc:PropertyName>
              <ogc:Literal>active</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Filter>
          <MaxScaleDenominator>70000000</MaxScaleDenominator>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                    xlink:type="simple"
                    xlink:href="car_green_real.svg"/>
                <Format>image/svg+xml</Format>
              </ExternalGraphic>
              <Size>
                <ogc:Literal>25</ogc:Literal>
              </Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
        <Rule>
          <Name>CarOffline</Name>
          <ogc:Filter>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>status</ogc:PropertyName>
              <ogc:Literal>stopped</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Filter>
          <MaxScaleDenominator>70000000</MaxScaleDenominator>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                    xlink:type="simple"
                    xlink:href="car_brown_real.svg"/>
                <Format>image/svg+xml</Format>
              </ExternalGraphic>
              <Size>
                <ogc:Literal>25</ogc:Literal>
              </Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
```

```
        <Rule>
          <Name>Carbreak</Name>
          <ogc:Filter>
            <ogc:And>
              <ogc:PropertyIsEqualTo>
                <ogc:PropertyName>status</ogc:PropertyName>
                <ogc:Literal>break</ogc:Literal>
              </ogc:PropertyIsEqualTo>
            </ogc:And>
          </ogc:Filter>
          <MaxScaleDenominator>70000000</MaxScaleDenominator>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                    xlink:type="simple"
                    xlink:href="car_gray_real.svg"/>
                <Format>image/svg+xml</Format>
              </ExternalGraphic>
              <Size>
                <ogc:Literal>25</ogc:Literal>
              </Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
        <Rule>
          <Name>CarNoData</Name>
          <ogc:Filter>
            <ogc:And>
              <ogc:PropertyIsEqualTo>
                <ogc:PropertyName>status</ogc:PropertyName>
                <ogc:Literal>inactive</ogc:Literal>
              </ogc:PropertyIsEqualTo>
            </ogc:And>
          </ogc:Filter>
          <MaxScaleDenominator>70000000</MaxScaleDenominator>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource
                    xlink:type="simple"
                    xlink:href="car_red_real.svg"/>
                <Format>image/svg+xml</Format>
              </ExternalGraphic>
              <Size>
                <ogc:Literal>25</ogc:Literal>
              </Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

PŘÍLOHA P II. UKÁZKA DOKUMENTU PRO SOUHLAS S GDPR

Zde je zobrazena pouze ukázka dokumentu, se kterým musí souhlasit uživatel aplikace před jeho registrací.

Souhlas s poskytnutím osobních údajů

Správcem osobních údajů je: ……, se sídlem …….

Zpracovatelem osobních údajů je: ……, se sídlem …….

Osobní údaje je oprávněn také zpracovávat kterýkoliv zaměstnanec správce.

Já, níže podepsaný, ……, datum narození ……, trvale bytem …….

v souladu s čl. 6 odst. 1 písm. a) nařízení Evropského Parlamentu a Rady (EU) 2016/679 ze dne 27. dubna 2016, o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů a o zrušení směrnice 95/46/ES (dále jen „nařízení GDPR“), které nabylo účinnosti dne 25. 5. 2018, **souhlasím se shromažďováním, uchováním a zpracováním mých osobních údajů mnou poskytnutých správci.**

Osobní údaje, kterými jsou:

- **jméno a příjmení,**
- **emailová adresa**

bude správce shromažďovat, uchovávat a zpracovávat pro následující účely:

- pro oznámení správce a identifikační údaje

Správce bude uchovávat osobní údaje po dobu nezbytně nutnou. Nejdéle bude správce uchovávat osobní údaje po dobu 3 let.

Osobní údaje budou zpracovány strojově (automatizovaně) prostřednictvím počítačů a počítačových programů.

Jako subjekt údajů mám právo na přístup k mým osobním údajům, mám právo na jejich opravu nebo vymazání, případně omezení zpracování. Mám právo požadovat informaci, jaké osobní údaje jsou zpracovávány a mám právo požadovat vysvětlení ohledně zpracování osobních údajů.

Dále mám právo vznést námitku proti zpracování, jakož i práva na přenositelnost údajů. Mám **právo souhlas kdykoli odvolat**, aniž by tím byla dotčena zákonnost zpracování založená na souhlasu uděleném před jeho odvoláním, pokud je zpracování založeno na čl. 6 odst. 1 písm. a) nebo čl. 9 odst. 2 písm. a) nařízení GDPR. To znamená, že takové právo nemám zejména tehdy, pokud je zpracování nezbytné pro splnění právní povinnosti, která se na správce vztahuje (zejména povinnosti vztahující se k LVTČ). Mám právo podat stížnost u Úřadu pro ochranu osobních údajů nebo se obrátit na soud.

Jako subjekt údajů jsem informován o tom, že souhlas mohu odvolat odesláním emailu na adresu: …….

Jako subjekt údajů mám právo požádat o informaci o zpracování mých osobních údajů, přičemž tuto informaci je správce povinen mi bez zbytečného odkladu předat. Obsah informace je dán ustanovením článku 15 nařízení GDPR. Správce má právo za poskytnutí informace požadovat přiměřenou úhradu nepřevyšující náklady nezbytné na poskytnutí informace.

Správce prohlašuje, že nedochází při zpracovávání k automatizovanému rozhodování, včetně profilování, uvedenému v čl. 22 odst. 1 a 4 nařízení GDPR.

Správce prohlašuje, že osobní údaje nejsou zpracovávány pro účely vědeckého či historického výzkumu.

Jako subjekt údajů prohlašuji, že jsem si vědom(a) svých práv podle kapitoly III nařízení GDPR.

Prohlašuji, že všechny poskytnuté údaje jsou přesné a pravdivé a jsou poskytovány dobrovolně.

Správce prohlašuje, že bude shromažďovat osobní údaje v rozsahu nezbytném pro naplnění účelu a zpracovávat je bude pouze v souladu s účelem, k němuž byly shromážděny.

Tento souhlas je svobodný a vědomý projev vůle subjektu údajů, jehož obsahem je svolení subjektu údajů se zpracováním osobních údajů.

V …… dne ……

……..