UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2017 Jan Holešínský

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Návrh a implementace systému pro efektivní plánování nákladní přepravy (web API, Databáze, Mobilní aplikace)

Jan Holešínský

Diplomová práce

2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 15. 5. 2017

Jan Holešínský

PODĚKOVÁNÍ

Chtěl bych poděkovat panu Ing. Janu Fikejzovi, Ph.D. za odborné konzultace a vedení při zpracování diplomové práce.

V neposlední řadě bych chtěl poděkovat rodině a přítelkyni, kteří mi byli nápomocni v průběhu tvorby této práce.

ANOTACE

Diplomová práce se zabývá návrhem a implementací systému pro efektivní plánování nákladní přepravy. Mezi stěžejní části práce se řadí web API, databáze a mobilní aplikace.

V první části práce jsou popsány současné trendy v oblasti nákladní přepravy. V následujících částech je práce zaměřena na konkrétní návrh a implementaci softwarového systému na podporu plánováni nákladní přepravy s využitím platformy .NET.

KLÍČOVÁ SLOVA

.NET, C#, doprava, API

TITLE

Design and implementation of a system for the effective planning of the freight transport (Web API, database, mobile applications)

ANNOTATION

Diploma thesis deals with design and implementation of system for effective planning freight transport. Main parts of diploma thesis are web API, database and mobile application.

In the first part of the thesis are described actual trends of freight transport. The other parts of the thesis are focused on design and implementation software system for support of freight transport planning based on .NET platform.

KEYWORDS

.NET, C#, transport, API

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK

API	Application programming interface
ASP	Active Server Pages
CRUD	Create, Read, Update, and Delete
EF	Entity Framework
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IIS	Internet Information Services
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
LINQ	Language-Integrated Query
POCO	Plain Old C# Object
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
XML	Extensible Markup Language

ÚVOD

Plánování nákladní dopravy je problematika, která je lidstvem řešena od pradávna, v počátcích člověk využíval zvířata, postupem času a objevováním nových technologií se doprava diametrálně změnila, začaly se používat lodě, železnice, nákladních automobily a letadla. Jedna věc se ale nezměnila od dob počátků nákladní dopravy, a to potřeba dopravu naplánovat co nejefektivněji.

Cílem diplomové práce je vytvoření komplexního systému pro efektivní plánování nákladní dopravy, s jejímž využitím bude umožněno usnadnění komunikace mezi dispečerem a řidičem. Aplikace je implementována s využitím nejmodernějších technologií a navržena tak, aby byla snadno ovladatelná a uživatelsky přívětivá.

V teoretické části práce jsou popsány moderní trendy v nákladní dopravě se zaměřením na silniční nákladní dopravu. Důraz je kladen na Průmysl 4.0 neboli čtvrtou průmyslovou revoluci, která zavádí do nákladní dopravy moderní informační technologie jako jsou cloud, strojové učení, internet věcí atd., zejména pak na nasazení těchto technologických vymožeností v reálném světě.

Podle provedených analýz je v současné době plánování nákladní dopravy ve většině malých a středních firem působících v tomto oboru řešeno využitím tužky a papíru, popřípadě nástroje Excel, nasazení pokročilých informačních technologií je naprosto minimální. Analýzou mezi řidiči a dispečery je ověřeno, že v současné době je komunikace řešena pomocí častého telefonování, kdy nezřídka dochází k chybám a následně je téměř nereálné dohledat podklady pro určení chybujícího. Je zde velký potenciál pro inovace, automatizace a optimalizace s využitím nejmodernějších informačních technologií.

Praktická část se zabývá návrhem a implementací systému pro efektivní plánování nákladní dopravy s využitím technologií popsaných v části teoretické. Účelem systému je efektivní plánování dopravy s využitím informačních technologií pro nákladní automobily a automatizace komunikace mezi dispečerem a řidičem. Systém pro efektivní plánování dopravy pojmenovaný Smart cargo planning je implementován ve vysokoúrovňovém objektově orientovaném programovacím jazyce C#, který využívá platformy .NET Framework. Cloudový poskytovatel Microsoft a jeho služba Azure je využita pro komplexní zastřešení systému, datové úložiště Smart cargo planning je postaveno na relační databází MS SQL s využitím

objektově relačního mapování, je zajištěna vysoká rychlost vyřízení jednotlivých požadavků a snadné úpravy databázového schématu.

1 TRENDY V NÁKLADNÍ DOPRAVĚ

Schopnost přesunu zboží bezpečně, rychle a hospodárně na jednotlivé trhy je klíčová pro mezinárodní obchod a ekonomický rozvoj. Rapidní nárůst světového obchodu spolu s prohloubením integrace rozšířené Evropské unie a řadou ekonomických praktik, jako je koncentrace výroby v menších místech s cílem dosáhnout velkých úspor, mají vliv na relativně rychlý nárůst nákladní dopravy po celé EU. Na zkvalitnění a zrychlení dopravy mají značný podíl i moderní technologie a s nimi související trend digitalizace. Stejně jako celý průmysl, tak i nákladní doprava prochází čtvrtou průmyslovou revolucí, jejíž koncept byl představen na veletrhu v Hannoveru v roce 2013, která spolu s výše uvedenou digitalizací přináší automatizaci opakujících se úkonů a změny na trhu práce.

Nákladní doprava se skládá z následujících kategorií:

- Letecká doprava
- Železniční doprava
- Námořní doprava
- Říční doprava
- Automobilová doprava

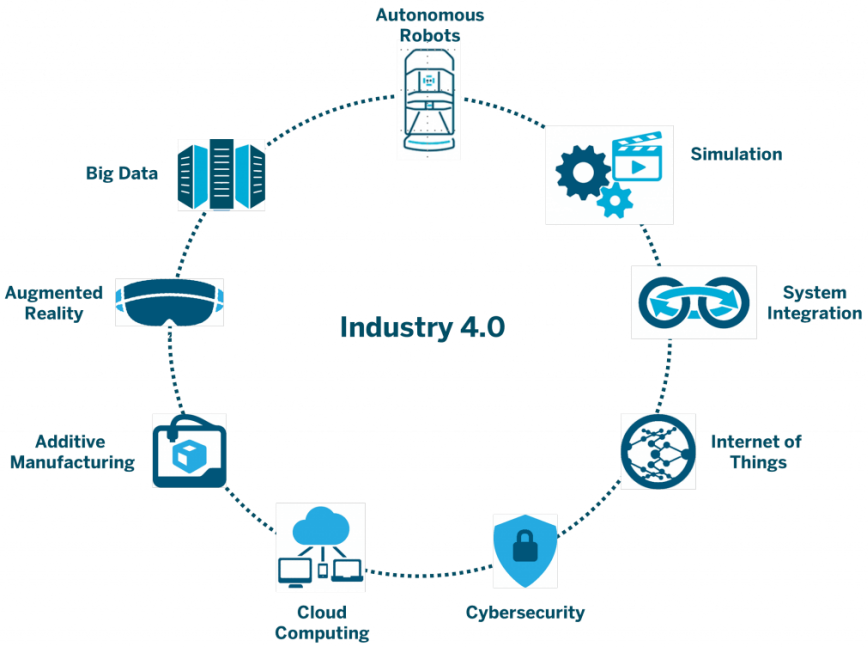
V každém z těchto odvětví se projevuje digitalizace, která úzce souvisí s trendem Big Data [1], v rámci tohoto trendu jsou uchovávána veškerá možná data o nákladní přepravě na rozsáhlých serverových uložištích umístěných po celém světě. Množství perzistentně uložených informací má velmi rozsáhlé využití, ať již co se týče statistických analýz, tak následného použití umělé inteligence.

Všechny výše zmíněné kategorie jsou podstatnou součástí světového obchodu a napomáhají k jeho neustálému růstu. Například výše zmíněná letecká doprava v roce 2014 přepravila 14,4 milionu tun zboží což je 27% nárůst oproti roku 2009. Je tedy jasně patrné, že poptávka po nákladní dopravě trvale roste a předpokládá se, že tomu bude i v budoucnu [2].

1.1 Průmysl 4.0

Současný trend Průmysl 4.0, který je také nazýván čtvrtou průmyslovou revolucí, přináší moderní informační technologie do světa průmyslu, zahrnuje internet věcí neboli IoT, Cloud computing, další kyberneticko-fyzikální systémy a všeobecnou digitalizaci. Koncept je

vytvořen na základě myšlenek z konference v Hannoveru konané v roce 2013, a předpokládá, že v budoucnu vzniknou chytré továrny, které za využití kyberneticko-fyzikálních systémů převezmou opakující se jednoduché činnosti, které v dřívějších dobách vykonávali lidé [3]. To povede ke změně pracovního trhu a zapříčiní vznik velkého množství nových pracovních míst se zaměřením na informační technologie.



Obrázek 1 - Složky průmyslu 4.0, zdroj: [22]

Do průmyslu budou zaváděny moderní technologie v podobě strojového učení, cloudových úložišť, datových center, robotů, rozšířené a virtuální reality, internetu věcí a simulací, které jsou zobrazeny na obrázku 1. Za pomoci těchto technologií bude docházet k úsporám peněz, času a navýšení flexibility firem. Průmysl 4.0 s sebou nese i související rizika, mezi které patří hlavně útoky hackerů a možnost zneužití získaných dat. Předpokládá se, že důkladné využití digitalizace zvýší v budoucnu produktivitu firem o 15 až 25 procent [4].

1.2 Průmysl 4.0 v nákladní dopravě

Z výše uvedených kapitol je jasně patrné, že hlavním trendem nákladní dopravy je začleňování moderních informačních technologií spolu v souladu se čtvrtou průmyslovou revolucí za účelem zvýšení flexibility dopravních společností.

Moderní informační technologie budou využívány v každém odvětví nákladní dopravy, od letecké, až po automobilovou, v této práci bude kladen důraz na využití současných trendů

v nákladní automobilové dopravě. Využití Průmyslu 4.0 v nákladní dopravě disponuje velkým potenciálem snížení nákladů dopravních firem doprovázeným zvýšením celkové efektivity nákladní dopravy.

1.2.1 Rozšířená a virtuální realita

Dopravní společnosti získávají objevením rozšířené a virtuální reality zcela nové možnosti, mezi které patří trénování klíčových úkonů řidičů za využitích těchto technologií. V praxi to znamená, že zaměstnanec dopravní společnosti, která rozšířenou realitu využívá, si může v testovací místnosti vyzkoušet běžné dopravní krizové situace, do kterých se může dostat při výkonu pracovních povinností, a předcházet tak škodám na životech a majetku.

Mezi zmíněné činnosti patří například couvání s přípojným vozidlem v prostorech společnosti dlouhodobého zákazníka firmy. Řidič je v testovací místnosti opatřen brýlemi a ovládáním volantu, v brýlích je nahraná aplikace, která simuluje například couvání s přípojným vozidlem mezi dva stojící stroje, jakmile řidič narazí nebo jinak chybně vyhodnotí situaci, aplikace ho nejprve upozorní a poté se akce opakuje, dokud řidič zadaný úkol nesplní. Výsledkem je, že řidič má daný úkon odzkoušen, takže při řešení reálného problému, již bude vědět, jak danou situaci řešit a jakým způsobem zareagovat. Přidanou hodnotou je, že při nácviku standartních situací nezpůsobí řidič žádné hmotné škody firmě a nepřivodí sobě, ani druhým osobám žádnou újmu na zdraví.

Mezi další výhody rozšířené a virtuální reality patří simulace prostředí. V případě, že řidič bude vykonávat svou práci například v hlučném prostředí stavby, je možné mu do sluchátek zapnout hlasitý ruch, který simuluje prostředí stavby a zaměstnavatel zjistí, jakým způsobem řidič reaguje, pokud je při výkonu povolání vystaven nadměrnému hluku.

1.2.2 Big Data

Jedná se o technologie, které umožňují ukládání velkého množství dat generovaných informačními systémy. V současné době jsou Big Data obvykle ukládána v datových skladech, většinou se jedná o data ve strukturované podobě, je tedy možné nad těmito daty provádět různé analýzy a výpočty. Problémem provádění analýz a výpočtů je, že je nutný velký výpočetní výkon pro zpracování z důvodu velkého množství dat. Tento problém byl odstraněn s příchodem cloud computingu.

V nákladní dopravě mají Big Data široké uplatnění, mezi hlavní pozitiva patří možnost naplánování trasy a optimalizace chodu automobilu. S využitím informací uložených

v datových skladech je například možné optimalizovat trasy nákladních aut. Obsahují-li datové sklady data z předchozích cest, je téměř triviální provést nad těmito daty analýzu a na základě jejího výsledku vybrat optimální trasu pro nákladní automobil. Dopravní společnost si při správně navrženém datovém skladu může dokonce vybírat z jednotlivých aspektů, na které chce klást důraz s ohledem na aktuální potřeby [5]. V případě zpoždění zvolí časovou optimalizaci trasy, v případě snížení nákladů zvolí optimalizaci na základě úspory nákladů za projeté pohonné hmoty a odpracované hodiny zaměstnanců.

Big data dávají dopravním firmám téměř neomezené možnosti analýz a výpočtů, z velkého množství dat vyplývá, že perzistentně uložená data by měla být snadno přístupná. Ze získaných výsledků mohou firmy vysledovat problematické oblasti a snížit náklady, které musí vynaložit na zpracování výsledků analýz a výpočtů prováděných specializovaným týmem.

1.2.3 Cloud computing

Zjednodušeně řečeno je cloud computing dodání výpočetních služeb, v podobě propůjčení výpočetního výkonu serverů, úložišť, databází, analytických nástrojů atd. V podstatě se jedná o přesun programů a dat z osobního počítače na korporátní servery [6]. Cloud computing s sebou nese mnohé výhody, mezi které se řádí škálovatelnost a snadná udržitelnost aplikací. Rozvržení přesunu technologií na cloudové poskytovatele je znázorněno na obrázku 2

Obrázek 2 - Cloud computing schéma, zdroj: [23]

Výše zmíněné specifikace Cloud computingu vnáší do nákladní dopravy nové možnosti v podobě využití serverových úložišť provozovatelů cloudu. Mezi největší výhody patří [7]:

- Nízké náklady na provoz

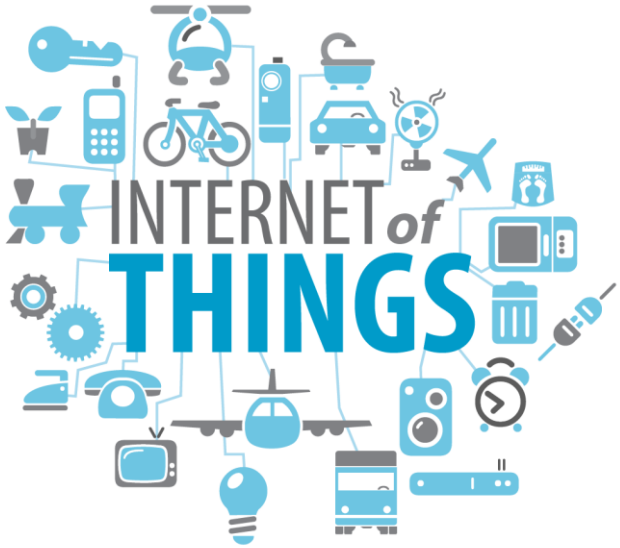
    - Eliminace nákupu hardware
    - Odpadá potřeba IT pracovníků spravujících infrastrukturu
- Vysoká rychlost pořízení
    - Značné výpočetní množství lze zajistit během pár minut
    - Snížení tlaku na plánování kapacit
- Elastická škálovatelnost
    - Snadné navýšení a snížení výpočetní síly
- Zvýšení produktivity
    - Nevyžaduje nastavování hardware a pořizování software
- Enormní výkon
    - Nejmodernější hardware a software
    - Celosvětová síť zabezpečených datových center
- Spolehlivost
    - Automatické zálohování dat
    - Snadné zotavení po havárii

Využití možností cloud computingu v nákladní dopravě s sebou přináší také nevýhody, mezi které se řadí [8]:

- Zjednodušení kontroly a monitoring osobních dat
- Bezpečnost dat
    - Data jsou přenášena přes internet
    - Uložení dat v cizí infrastruktuře
- Potřeba stálého internetového připojení

1.2.4 Internet věcí

Zkratka IoT pochází z anglického Internet of Things. Zjednodušená definice IoT vyjadřuje komunikaci zařízení připojených k internetu, která posílají informace k cloudově založeným aplikacím. Tato zařízení mohou být libovolná, např. mobilní telefony, motory, lékařské vybavení a další, jak je ukázáno na obrázku 3 [9].



Obrázek 3 - Zařízení používající IoT, zdroj: [24]

V současné době je IoT využíváno firmou Rolls-Royce k zaznamenávání gigabytů dat z jejich leteckých motorů, které pohánějí letadla více než 85 leteckých společností. Pro analýzu dat je využíváno cloudové služby Azure od společnosti Microsoft. Analýza je využívána například pro sledování spotřeby paliva a plánování oprav. Hlavním záměrem využití IoT v dopravě je přechod z pouhého monitorování stavu motorů na aktivní úpravu účinnosti motorů, aby se předcházelo poruchám a dalším možným problémům [10]. Poznatky firmy Rolls-Royce, získané ze zkušeností při používání leteckých motorů, jsou dnes využívány při řešení problémů s motory používanými v nákladních automobilech.

Další případová studie se zabývá zvýšením bezpečnosti silničního provozu za špatných klimatických podmínek. Ročně nebezpečné klimatické podmínky na silnici zapříčiní 7 400 úmrtí, 700 000 zranění a finanční výdaje jednotlivých států USA spojené s nehodovostí při zhoršených klimatických podmínkách činí okolo 2.3 miliardy dolarů. S využitím výhod IoT jsou jednotlivé státy schopné získávat informace o klimatických podmínkách v jednotlivých mikroregionech a mohou predikovat krizové situace a zvýšit bezpečnost silničního provozu [11].

Implementace technologie IoT v nákladní dopravě umožňuje provozovatelům udržovat vozidla v provozu pomocí predikce a monitorování jejich stavu, sledovat chování řidičů, používat aktuální data k optimalizaci dopravních cest a tím zabraňovat tvorbě dopravních kolapsů.

1.2.5 Strojové učení

Data mohou uchovávat skryté informace, obzvláště pokud je dat velké množství. Dostatečné množství dat je možné inteligentním způsobem prozkoumat a je možné objevit vzorce, které jsou příliš komplexní, aby byly odhaleny člověkem. Na tomto principu funguje strojové učení. Dochází k prověření velkého množství dat, v nichž jsou nalezeny vzorce a vygenerovány zdrojové kódy, které umožní predikce do budoucna [12].

Pro využití strojového učení je potřeba velké množství dat, z tohoto důvodu se často strojové učení využívá v kombinací s technologií Big Data. V případě, že provozovatel nákladní společnosti potřebuje v současné době snížit poruchovost motorů, najme pravděpodobně tým specialistů, který přijde s adekvátním řešením, je velmi pravděpodobné, že dodané řešení bude fungovat, ale náklady na zpracování zadání jsou velmi vysoké. Za předpokladu využití technologie strojového učení v kombinaci s technologií big data disponuje provozovatel velkým počtem záznamů z motoru, informace jsou ve formě databázových tabulek, v řádech milionů záznamů s velkým množstvím sloupců. Velké množství dat je téměř nemožné analyzovat bez použití technologie strojového učení, která zjistí, v jakých případech dochází k poruchovosti motorů a provozovateli ušetří náklady spojené s opravami automobilů a další výdaje jako penále za pozdní dodání. Výsledkem strojového učení je vzorec, který definuje situace, v nichž dochází k poruše motorů. Pro příklad může jít o vzorec, který signalizuje, že motory při nájezdu 150 000 km bez výměny filtrů budou mít poruchu. V reálném světě jsou vzorce daleko komplexnější.

Použití strojového učení kombinuje většinu výše zmíněných technologií. Konkrétně IoT k získávání potřebných dat, big data k jejich perzistentnímu uchování a cloud computing za účelem získání dostatečného výpočetního výkonu. S vhodnými datovými podklady disponuje strojové učení velkým potenciálem stát se klíčovou technologií v budoucnosti.

1.2.6 Použití Průmyslu 4.0 v systému plánování dopravy

V Průmyslu 4.0 byly představeny technologie, jejichž využíváním je vývojářům poskytnuto mnoho nových možností, jak vyvíjet moderní informační systém za relativně nízkých nákladů. V tomto duchu se nese vývoj systému pro efektivní plánování dopravy, který je komplexním řešením pro dopravní společnosti. Aplikace umožňuje naplánování dopravy pro společnosti s využitím cloudu a následnou synchronizaci do mobilních telefonů.

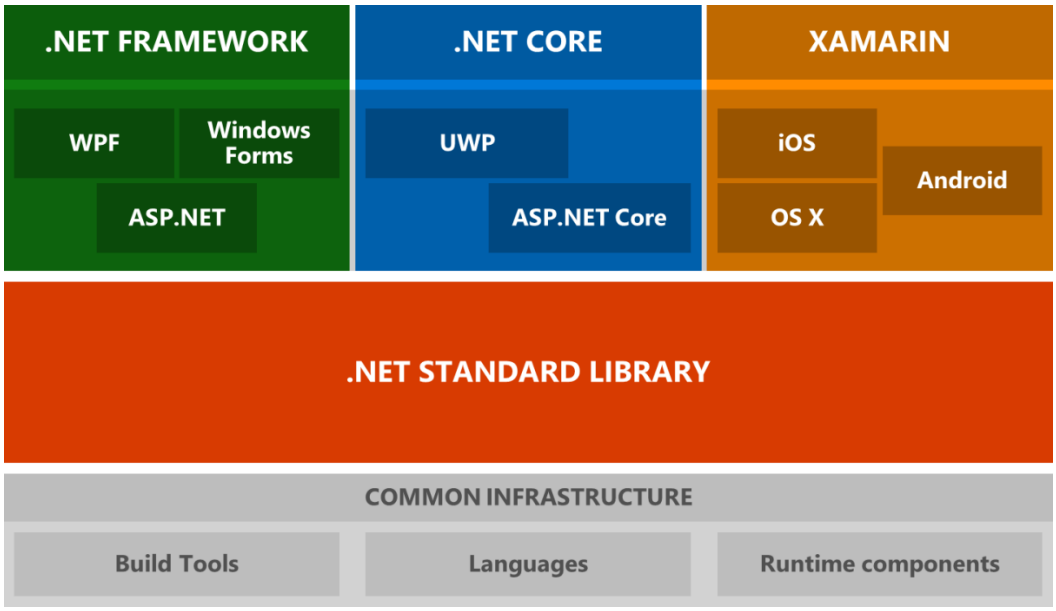
2 POUŽITÉ TECHNOLOGIE

Pro tvorbu aplikace byla zvolena platforma .NET, která byla vybrána z důvodů dlouhodobého využívání na trhu, integrace cloudových nástrojů a multiplatformní podpory. Mobilní aplikace byla vytvořena pomocí technologie Xamarin.Forms z důvodu multiplatformního využití na platformě Android a iOS.

2.1 ASP.NET Core

.NET Core je všeobecná vývojová platforma spravovaná společností Microsoft a komunitou .NET na GitHubu. .NET Core je multiplatformní, podporuje Windows, MacOS a Linux, dále může být použito jako zařízení, cloud nebo IoT. Více o .NET Core a .NET Standart je uvedeno v diplomové práci Jana Matičky.

ASP.NET Core je nový open source a multiplatformní framework určený pro vytváření moderních cloudově založených internetových aplikací, jakými jsou webové aplikace a backendy pro mobilní aplikace. ASP.NET Core aplikace mohou být spuštěny na .NET Core nebo na úplném .NET Frameworku. Kompletní struktura .NET frameworku je ukázána na obrázku 4. Architektura byla navržena tak, aby byl poskytnut optimalizovaný vývojový framework pro aplikace, které jsou nasazeny na cloudu nebo běží lokálně. Framework je složen z modulárních komponentů, které mají minimální režii. Umožňuje| vývojáři získat flexibilitu v průběhu vytváření řešení. ASP.NET Core jsou multiplatformní aplikace, které mohou být nasazeny na Windows, Linux a Mac [13].



Obrázek 4 - Struktura .NET, zdroj: [25]

ASP.NET Core a .NET framework jsou v této práci popsány pouze informativně se zaměřením na tvorbu API a její zabezpečení, detaily implementace webové aplikace a zavádění klientských frameworků jsou popsány v diplomové práci Jana Matičky.

2.1.1 Motivace k použití ASP.NET Core

První preview ASP.NET bylo vydáno před více než 15ti lety jako část .NET Frameworku, od té doby používají ASP.NET miliony vývojářů po celém světě, s nárůstem použití byly přidány další funkcionality. Modulární architektura je založená na NuGet balíčcích, které reprezentují jednotlivé rozšíření a funkcionality, minimalizuje aplikace a umožňuje snadnou optimalizaci. Výhody menší aplikace jsou [13]:

- Vyšší zabezpečení
- Snížení údržby
- Navýšení výkonu
- Snížení nákladů

2.1.2 Modulárnost ASP.NET Core

Modulárnost ASP.NET Core je zajištěna pomocí technologie NuGet, jedná se o správce balíčků pro vývoj na platformě Microsoft. Nástroje pro klienty NuGet poskytují schopnost vytvářet a konzumovat balíčky, které jsou uložené na centrálním balíčkovém repositáři, který využívají všichni tvůrci a konzumenti balíčků [14].

2.1.3 Klíčové vlastnosti ASP.NET Core

Klíčové vlastnosti ASP.NET Core plynou z jeho restrukturalizace a modulárnosti za pomoci využití balíčků NuGet. Použitím ASP.NET Core vývojář získá [13]:

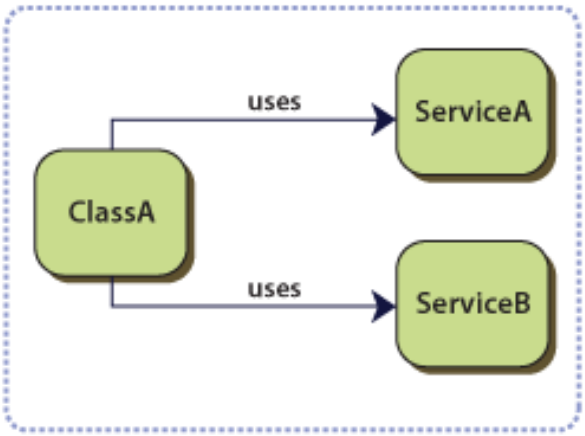
- Jednotný přístup k vytváření webového UI a API
- Snadnou integraci moderních klientově orientovaných frameworků
- Systém připravený na nasazení do cloudu
- Vestavěnou Dependency injection
- Možnost hostování aplikace na IIS nebo vlastních serverech
- Multiplatformní aplikaci s možností nasazení na Windows, Mac a Linux

Detailnější popis klientských frameworků, jejich použití a integrace v ASP.NET Core jsou charakterizovány v diplomové práci Jana Matičky.

Dependency injection je technika pro dosažení volného spojení mezi objekty a jejich spolupracovníky nebo závislostmi, namísto přímého spouštění spolupracujícími objekty nebo pomocí statických referencí. Nejčastěji budou třídy deklarovat jejich závislosti prostřednictvím svého konstruktoru.

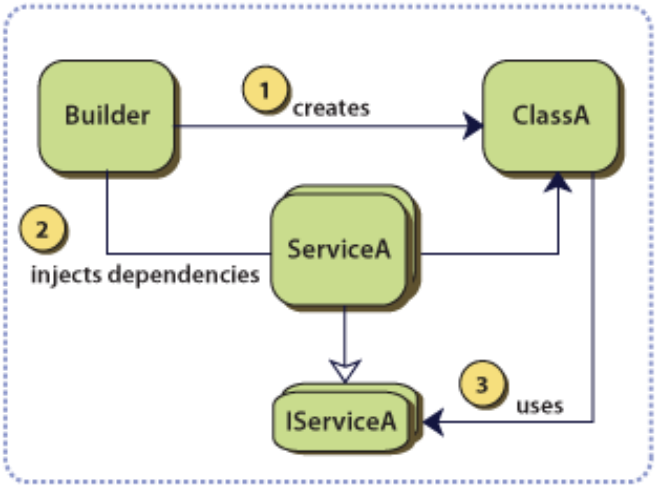
ASP.NET Core je přizpůsobeno k využití výhod vestavěného dependency injection tím, že jednotlivé služby jsou vloženy přímo do metod. Výchozí kontejner obsahuje pouze minimální sadu funkcí.

S využitím dependency injection je vyřešen problém přímé vazby na objekty, přímá vazba je vyobrazena na obrázku 5.



Obrázek 5 - Přímá vazba objektů, zdroj: [26]

Schéma tvorby objektu s využitím dependency injection je znázorněno na obrázku 6. Objekt je vytvořen pomocným objektem Builder, který vloží v konstruktoru potřebné závislosti.



Obrázek 6 - Využití dependency injection, zdroj: [26]

2.1.4 REST API

REST API definuje množinu architektonických principů, kterými je řízen návrh webové služby. Webové služby splňující REST architekturu jsou přenášeny pomocí protokolu HTTP. REST byl poprvé představen v roce 2000 Royem Fieldingem na univerzitě v Kalifornii. Disertační práce Roye Fieldinga navrhuje, aby webová služba splňovala následující body:

- Používat http metody explicitně
- Bezstavovost
- Sestavit adresářovou strukturu podle URI
- Přenášet data ve formátu XML, JavaScript Object Notation (JSON), nebo obojí.

V důsledku rozšíření architektury REST téměř vymizela rozhraní založená na službě SOAP a WSDL, jedním z důvodů bylo, že REST je jednodušší na implementaci a použití [17].

2.1.5 Tvorba API v ASP.NET Core

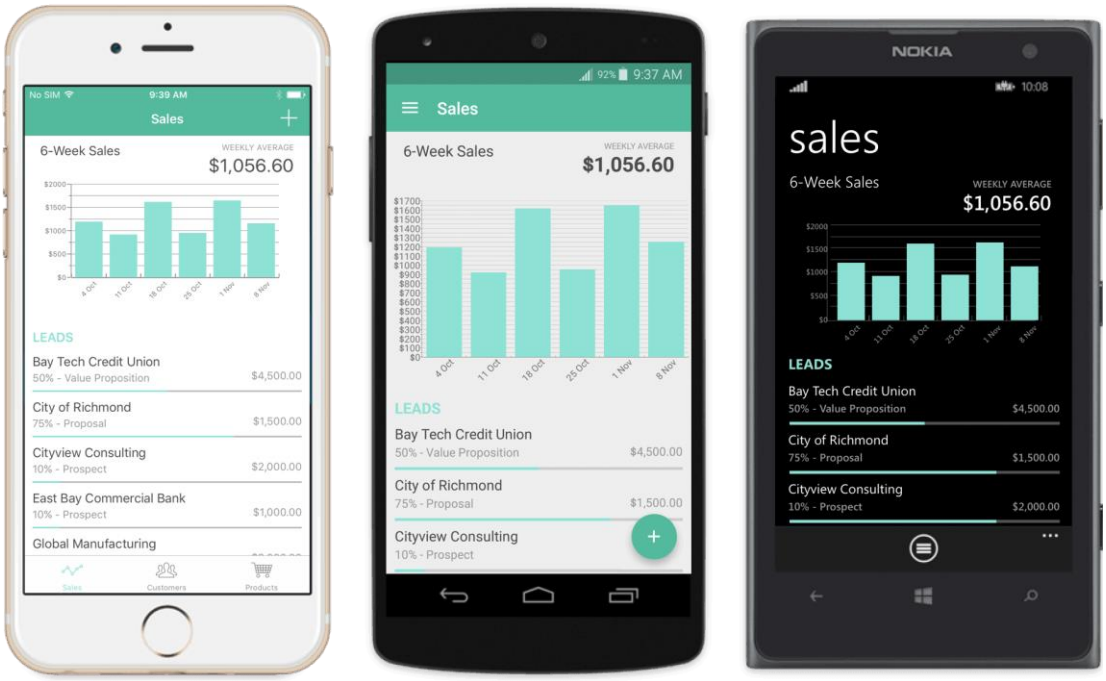
ASP.NET Core disponuje vestavěnou podporou pro tvorbu API, která je založena na HTTP protokolu, jenž neslouží pouze k zobrazování webových stránek, ale je také silným nástrojem pro vytvoření API. HTTP protokol je jednoduchý, flexibilní a všudypřítomný. Téměř každé zařízení má vestavěnou podporu tohoto protokolu, HTTP služba může dosáhnout na široké

množství klientů, ať již prostřednictvím webových prohlížečů, mobilních zařízení nebo tradičních desktopových aplikací [16]. ASP.NET Core vytváří API jako webovou službu implementující standardy REST.

2.2 Technologie Xamarin.Forms

Vývoj mobilních aplikací je různorodé prostředí, které vyžaduje specifické programovací jazyky a nástroje, z tohoto důvodu je vývoj multiplatformních aplikací finančně náročný. Pro systém efektivního plánování dopravy byla po prověřování všech technologií, které umožňují multiplatformní vývoj, vybrána technologie Xamarin.Forms. V rámci Xamarin.Forms je programátorovi umožněno vytvářet mobilní aplikace na platformy iOS, Android a Windows v jediném programovacím jazyce, kterým je C#, odpadá tedy potřeba psaní zdrojového kódu v Objective-C pro iOS a Javě pro Android. Výstupem jsou nativní aplikace pro každou výše zmíněnou platformu, které je snadné publikovat na oficiálních kanálech společností Apple, Google a Microsoft [21]. Na obrázku 7 je ukázáno porovnání zobrazení aplikace na všech třech platformách.

Sdílení kódu, ale není možné vždy, každá platforma má specifické volání určitých funkcí, zejména funkcí telefonu, jako jsou telefonický hovor, zobrazení na mapě, přístup k fotoaparátu atd. Pro tyto funkcionality je potřeba napsat specifický kód, který bývá většinou poměrně složitý. Je to daň za multiplatformní přístup.



Obrázek 7 - Výstup ze Xamarin.Forms, zdroj: [27]

3 ZABEZPEČENÍ CLOUDOVÝCH APLIKACÍ

Svět se nyní nachází v době, ve které jsou uchovávána data o všem a o všech a tuto skutečnost potvrzuje výše uvedený trend informačních technologií big data. Pod pojmem bezpečnost nejsou myšleny pouze útoky a možnosti úniku dat, ale zároveň i bezproblémový přístup k datům a provoz bez výpadků. Důležité bezpečnostní aspekty, které je potřeba zohlednit při vytváření zabezpečené cloudové aplikace a jejich řešení jsou popsány níže [18].

3.1 Zálohování dat

Data jsou ve většině případů zálohována automaticky poskytovatelem cloudové služby, záloha probíhá několikrát denně, popřípadě podle požadavků uživatelů. Veškerá data jsou zálohována na několik disků, které jsou zrcadlené. Ztráta dat je tedy nepravděpodobná.

3.2 Zneužití dat

Zneužití dat se ze své podstaty dělí na dvě části, zneužití obsahu dat nebo fyzické odcizení dat. Fyzické odcizení dat, popřípadě celého serveru můžeme téměř vyloučit, valná většina poskytovatelů data center je vybavena vysoce kvalitním zabezpečením. Zneužití dat je možné minimalizovat za využití vhodných šifrovacích algoritmů a zabezpečením celkové komunikace aplikace protokolem HTTPS.

3.3 Vnější útoky

V současné době je zabezpečení dat v cloudu proti vnějším útokům bráno jako standard a je zahrnuto v poplatku za odebírané služby, ale neznamená to, že úspěšný útok je možné vyloučit. Mnohem komplexnějším problémem jsou takzvané DDoS útoky. Zkratka DDos pochází z anglického distributed denial of service, jedná se o odepření služby. DDos útok je v podstatě přehlcení serveru fiktivními požadavky a z toho vyplývající neschopnost odpovídat na reálné požadavky od uživatelů. Problémem je, že požadavky jsou vysílány z celého světa a není tedy jednoduché tomuto typu útoku zabránit. Cloudové služby jsou vybaveny ochranou před DDoS útoky automaticky od provozovatele, zároveň je nabízena podpora při zjištění útoku.

3.4 Zastaralý software a hardware

Aktuální software, stejně tak jako hardware je poskytován provozovateli cloudových služeb. Aktualizace jsou připravovány a nahrávány centralizovaně na všechny servery. V datových centrech probíhá neustálý monitoring hardwarových komponent a v případě potřeby jsou součástky okamžitě vyměněny. Aktuálnost hardware a software je tedy zaručena.

3.5 Stabilita, dostupnost a spolehlivost

Stabilita, dostupnost a spolehlivost je jednou z největších výhod cloudových aplikací. Poskytovateli cloudových služeb je umožněno přizpůsobování zdrojů aktuální potřebě, tím je zamezeno přetížení a zpomalení aplikace. U cloudu bývá zpravidla garantováno téměř 100 % dostupnosti.

3.6 Zabezpečení ASP.Net Core

Výše zmíněné aspekty zabezpečení cloudové aplikace jsou v případě ASP.NET Core řešeny firmou Microsoft. Společnost Microsoft nabízí komplexní cloudové řešení Azure, které je automaticky zabezpečeno proti všem výše zmíněným bezpečnostním aspektům cloudových aplikací. Tato část bude tedy zaměřena na zabezpečení samotné aplikace.

3.6.1 ASP.NET Identity

ASP.NET Identity je systém spravující uživatele, který byl představen v roce 2005. V ASP.NET 2.0 od této doby došlo k mnoha změnám a vylepšením, které se týkají především autentizace a autorizace. ASP.NET Identity představuje standard, jak by měla vypadat správa uživatelů v moderních aplikacích pro web, či mobilních zařízení (mobil nebo tablet) [19].

ASP.NET Identity bylo vyvíjeno několik let na základě podnětů od zákazníků. Předpoklad, že uživatelé se budou přihlašovat do aplikací pouze uživatelským jménem a heslem, už delší dobu neplatí. Internet se stal více sociální, uživatelé mezi sebou komunikují v reálném čase skrz kanály sociálních sítí jako je Facebook, Twitter a další sociální sítě. Na vývojáře je vytvářen tlak, aby umožnili uživatelům přihlásit se prostřednictvím těchto účtů, moderní správa uživatelů by tedy měla být schopna přesměrovat přihlášení na poskytovatele, jakými jsou Facebook, Twitter atd. Při uvážení všech změn, bylo ASP.NET vyvíjeno pro dosažení následujících cílů: [19].

- Jeden komplexní systém spravující identitu uživatelů
- Snadné přidání dalších informací o uživateli
    - Kontrola nad schématem uživatele
- Kontrola perzistentního uložení dat
    - Standardně ASP.NET Identity ukládá informace o uživatelích do SQL databáze
    - Snadná změna názvů tabulek
    - Jednoduchá změna úložiště na NoSQL databáze nebo SharePoint
- Testovatelnost pomocí Unit testů
- Podpora uživatelských rolí a oprávnění
- Možnost přihlášení účtem ze sociální sítě
- Začlenění Active Directory
- Integrace služby OWIN
- Složení z jednotlivých NuGet balíčků

3.7 Objektově relační mapování

Objektově relační mapování (ORM) je využíváno pro převod dat mezi relační databází a objektově orientovanými jazyky. Aplikace je tedy schopna pracovat s daty v podobě objektů a jejich uchování realizovat pomocí relačních databází. Je možné objekty ukládat i do objektových databází, ale tyto databáze v současné době ještě nejsou tak propracované a rychlé. Výhodou použití ORM je zjednodušení implementace, ke kterému dochází díky možnosti pracovat přímo s objektovým modelem. Aplikace jsou snadno přenositelné mezi různými databázovými systémy a potlačují potenciální chyby v SQL, které se projeví až za běhu aplikace. Mezi nevýhody je možné zařadit menší výkon, kvůli režii převodu dat a nemožnosti použití některé funkcionality relačních databází. V platformě .NET je využíváno pro ORM mapování frameworku Entity Framework.

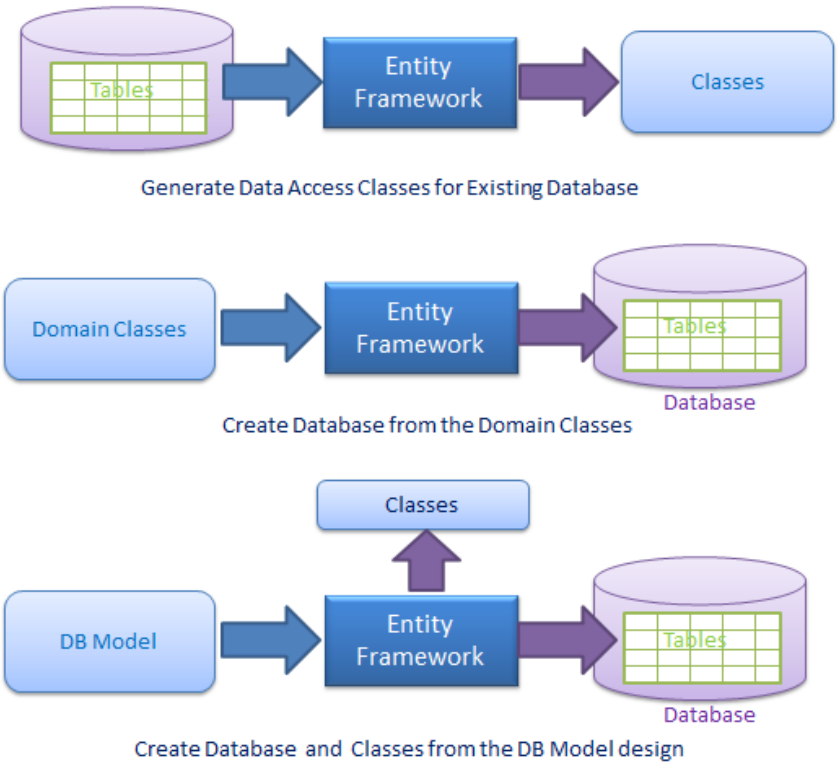
3.7.1 Použití Entity Framework Core

Entity Framework (EF) použitý při tvorbě systému pro efektivní plánování dopravy je odlehčená, lehce rozšiřitelná multiplatformní verze populární technologie přístupu k datům Entity Framework.

EF Core je objektově relační mapper, který umožňuje vývojářům na platformě .NET práci s databází pomocí .NET objektů. Eliminuje potřebu psaní kódu, jenž přistupuje k datům, který

obvykle vývojáři píší. EF podporuje řadu databázových strojů, jako jsou Microsoft SQL, PostgreSQL, MySQL atd. V blízké budoucnosti je připravováno napojení na databázi Oracle. EF Core je pro ASP.NET Core dostupný jako NuGet balíček [20].

Entity framework podporuje dva přístupy vytváření databáze. První možnost je vygenerovat datovou vrstvu aplikace z již existující databáze, druhou možností je z jednotlivých tříd vygenerovat databázové tabulky, obě možnosti jsou ukázány na obrázku číslo 8.



Obrázek 8 - Možnosti využití EF, zdroj: [28]

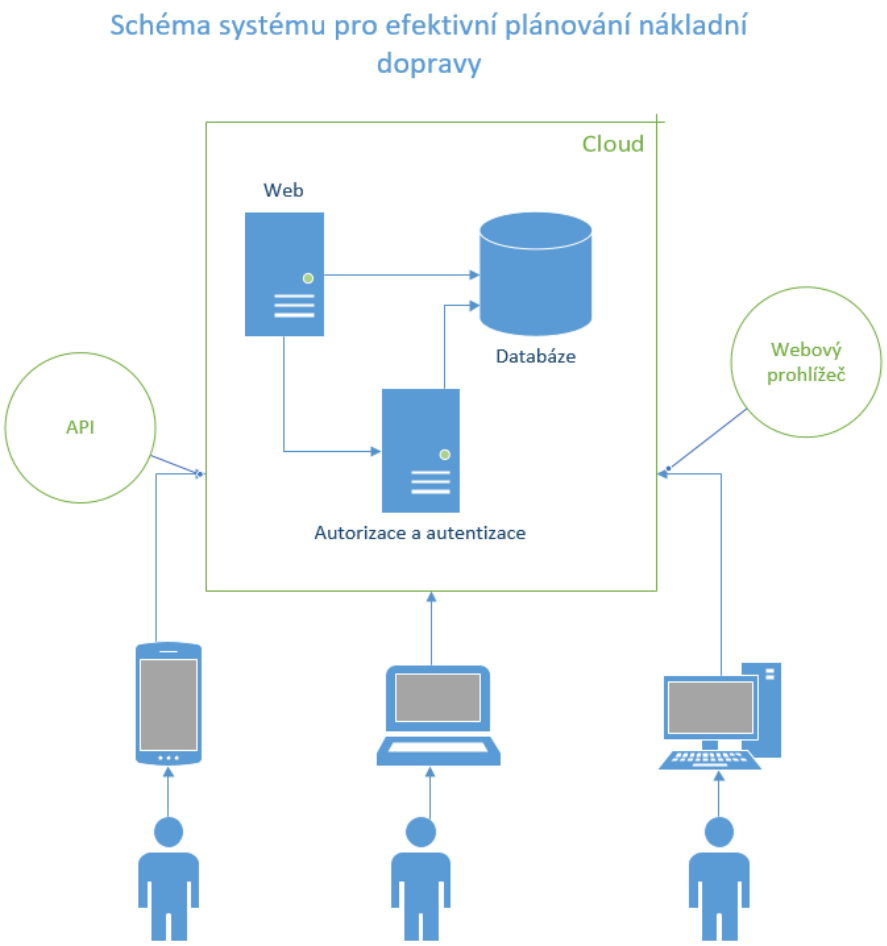
4 NÁVRH A IMPLEMENTACE

Cílem práce bylo vytvořit systém pro efektivní plánování nákladní dopravy, který bude založen na platformě .NET ovládané vysokoúrovňovým objektově orientovaným programovacím jazykem C#. Systém pojmenovaný Smart cargo planning je složen z jednotlivých modulů, mezi které se řadí mobilní aplikace vytvořená s použitím technologie Xamarin.Forms, cloudová webová aplikace postavená na ASP.NET Core a databáze využívající ORM mapování. Systém, který vznikl v rámci této diplomové práce je velmi komplexní, stavba serverové části je znázorněna na diagramu tříd v příloze A, stavba datové vrstvy v příloze B.

Technologie, které byly využity pro systém efektivního plánování dopravy, byly hluboce analyzovány s důrazem kladeným na dostupnost, multiplatformní využití a následnou rozšiřitelnost řešení. Vybrána byla platforma .NET, protože poskytuje komplexní pokrytí všech částí vývoje moderních aplikací.

S použitím nejmodernějších informačních technologií představených v rámci čtvrté průmyslové revoluce bylo dosaženo cloudového řešení, které umožňuje plánování nákladní dopravy a automatizaci komunikace mezi dispečery a řidiči.

Cloudová webová aplikace byla vybrána z důvodu snadné dostupnosti po celém světě, uživatelé aplikace mohou plánovat dopravu na služební cestě, v kanceláři nebo z pohodlí domova. Automatizace kontaktování řidičů a oznamování pracovních výkonů je prováděna přes zabezpečené API přímo do mobilního telefonu vlastněného řidiči, je tak zabráněno možnosti zneužití dat. Schéma systému pro efektivní plánování nákladní dopravy je zobrazeno na obrázku 9.



Obrázek 9 - Schéma systému, zdroj: vlastní

4.1 Provoz systému

Pro provoz systému bylo využito cloudových služeb od společnosti Microsoft, konkrétně je použito řešení Azure, které má rozmístěno datová centra po celém světě a umožňuje vytvořit komplexní infrastrukturu systému v krátkém časovém horizontu.

4.1.1 Azure

Azure je kompletní cloudová platforma, která je schopna hostovat existující aplikace, poskytovat služby vytvořené na míru nebo rozšířit aplikace uložené lokálně. Hlavní výhodou použití Azure je možnost snadného výběru virtuálních strojů podle potřeb uživatelů, vysoká spolehlivost, která je garantována na více než 99 % a nulová počáteční investice [29].

Zdroje na Azuru jsou spravovány pomocí portálu, což je webová aplikace, ve které můžete vytvářet, upravovat nebo mazat zdroje a služby. Portál Azure je dostupný na adrese

https://portal.azure.com. Azure disponuje velmi pokročilými nástroji pro monitorování aplikace, odpadá tak potřeba implementovat své vlastní.

Použití Azure je zpoplatněno v rámci použití předplatných, která jsou navázána na jednotlivé Microsoft účty. Poplatky jsou hrazeny každý měsíc a jejich výše se liší podle množství dat a zdrojů.

4.2 Databáze

Po provedení analýz všech požadavků na systém byl zvolen návrh datovou vrstvu vytvořit pomocí ORM mapování, vzhledem k dobré integraci byl vybrán framework Entity Framework. Model byl vytvořen s využitím návrhu CodeFirst, kdy je z jednotlivých POCO tříd (Plain Old C# Object) vygenerováno databázové schéma, přístup CodeFirst byl vybrán s ohledem na pravděpodobné časté budoucí změny v databázovém modelu.

4.2.1 POCO třída

POCO třída se skládá z konstruktoru a jednotlivých vlastností, které s využitím ORM mapování budou transformovány na názvy sloupců v tabulce a její cizí klíče. Příklad POCO třídy je ukázán na obrázku číslo 10

```
28 references | Jan Holešínský, 23 days ago | 3 authors, 6 changes
public class Trailer : CarBase
{
    [Key]
    3 references | JanHolešínský, 112 days ago | 1 author, 2 changes
    public int TrailerId { get; set; }
    8 references | Jan Holešínský, 27 days ago | 2 authors, 3 changes
    public virtual TrailerType TrailerType { get; set; }
    1 reference | Jan Holešínský, 27 days ago | 1 author, 1 change
    public virtual int TrailerTypeId { get; set; }
    3 references | Jan Holešínský, 23 days ago | 1 author, 1 change
    public ICollection<Ride> TrailerRides { get; set; }
}
```

Obrázek 10 - POCO třída, zdroj: vlastní

Z výše uvedeného obrázku je patrné, že ORM mapování za využití technologie Entity framework podporuje dědičnost, není potřeba psát redundantní kód. V této práci je dědičnost využita pro oddělení auta a přívěsu, obě entity mají hodně společných vlastností, které není potřeba uchovávat na dvou místech a sledovat, který uživatel upravil danou entitu. Je vytvořena třída Trackable zaštiťující tuto funkcionalitu pro všechny třídy, u kterých je to vyžadováno. EF rozpozná, že v každé tabulce má vytvořit dané sloupce pro sledování aktivity.

4.2.2 Tvorba databázového modelu

Databázové řešení bylo vybráno od společnosti Microsoft z důvodu snadné integrace a vysoké kompatibility se všemi použitými technologiemi. Model relační databáze byl vygenerován pomocí nástroje EF.

EF používá množinu konvencí pro tvorbu modelu na základě jednotlivých POCO tříd. POCO třídy jsou reprezentovány proměnnými datového typu DataSet, které se nacházejí ve třídě spravující databázový kontext a reprezentují data v tabulce. Jedinou podmínkou je, aby daná třída dědila od třídy IdentityDbContext. IdentityDbContext je využit pro správu uživatelů, EF automaticky přidá potřebné tabulky, ve kterých budou uživatelé perzistentně uchováni. Proměnné typu DataSet budou následně odzrcadleny do jednotlivých tabulek. Pro tvorbu modelu je možné využít dvou způsobů konfigurace.

Prvním způsobem je přepsání chování metody OnModelCreating, tato možnost je nejpoužívanější a umožňuje velmi specifickou konfiguraci. Tento způsob se nazývá programování pomocí fluentního API a má nejvyšší prioritu. Na obrázku 11 je zobrazena ukázka modelování pomocí fluentního API, kdy jsou nastavovány cizí klíče pro konkrétní tabulky řidičů.

```
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    // Customize the ASP.NET Identity model and override the defaults if needed.
    // For example, you can rename the ASP.NET Identity table names and more.
    // Add your customizations after calling base.OnModelCreating(builder);

    //MxM relation https://docs.microsoft.com/en-us/ef/core/modeling/relationships
    builder.Entity<DriverLicenseApplicationUser>().HasKey(t => new {t.Id, t.DriverLicenseId});

    builder.Entity<DriverLicenseApplicationUser>()
        .HasOne(x => x.ApplicationUser)
        .WithMany(p => p.DriverLicenses)
        .HasForeignKey(x => x.Id);
```

Obrázek 11 - Ukázka fluentního API, zdroj: vlastní

Druhým způsobem je použití data anotací, které jsou zobrazeny na obrázku číslo 9, avšak použití anotací není příliš užíváno, protože fluentní API má vyšší prioritu. Výsledný databázový model se skládá z osmnácti tabulek.

4.2.3 Úprava databázového modelu

Framework EntityFramework (EF) disponuje schopností snadno upravovat databázové modely, je označována jako migrace. Pro povolení úprav databáze neboli migrací je potřeba pouze zadat do příkazové řádky ve vývojovém prostředí Visual Studio následující příkaz Enable-Migrations, dojde k vytvoření nové složky v projektu, kde budou všechny migrace uloženy.

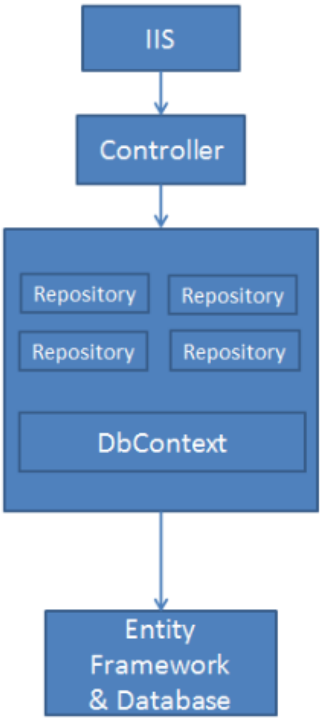
Pro vytvoření konkrétní nové migrace stačí udělat úpravu v POCO třídách, které ve zdrojovém kódu zachycují model databáze, a použít následující sekvenci příkazů:

1. Add-Migration
    - Slouží k vytvoření konkrétní migrace (změny v databázi)
2. Update-Database
    - Provede změny do databáze

Po vytvoření a provedení změn v databázi aplikace automaticky začne používat nový databázový model bez jakékoliv ztráty dat.

4.3 Přístup k datům

Pro přístup k datům byl zvolen návrhový vzor repositáře, který vytváří vrstvu abstrakce mezi datovou a business vrstvou aplikace. Implementace tohoto návrhového vzoru zapříčiní izolaci aplikace před změnami v datovém uložišti a usnadní automatizované testování aplikace. Systém efektivního plánování dopravy disponuje repositářem pro každou entitu, neboli POCO třídu, struktura s využitím repositářů je ukázána na obrázku 12.



Obrázek 12 - Struktura aplikace s repositáři, zdroj: vlastní

Každá entita disponuje interfacem a jeho následnou implementací v konkrétní třídě, tato vrstva abstrakce umožňuje snadnou manipulaci s datovou vrstvou, v extrémním případě je například možné vyměnit relační databázi za objektovou bez nutnosti, jakkoliv měnit logiku aplikace.

Všechny repositáře jsou zaregistrované jako služba v kontejneru aplikace ve třídě Startup, která je spuštěna jako první při zapnutí aplikace. Registrace je provedena pomocí dependency injection, která umožňuje kontrolovat životní cyklus objektů. V ASP.NET Core jsou standardně podporovány 3 typy [15]:

- Scoped
    - Vytvořeny jednou dle požadavku
- Singleton
    - Vytvořeny jednou za běh celé aplikace
- Transient
    - Vytvořeny pokaždé, když jsou potřeba

```
//add repositories
services.AddScoped<ICarRepository, CarRepository>();
services.AddScoped<ICustomerRepository, CustomerRepository>();
services.AddScoped<IDrivigLicenseRepository, DrivigLicenseRepository>();
services.AddScoped<IRideRepository, RideRepository>();
services.AddScoped<ITrailerRepository, TrailerRepository>();
services.AddScoped<ITrailerTypeRepository, TrailerTypeRepository>();
services.AddScoped<ICarTypeRepository, CarTypeRepository>();
```

Obrázek 13 - Registrace repositářů zdroj: vlastní

Repositáře, jak je vidět na obrázku 13, jsou z důvodu zachování integrity dat vytvořeny v režimu Scoped, při vytvoření v jiném režimu dochází k nebezpečí duplikace dat. Funkcionality repositáře jsou pevně stanoveny interfacem, například repositář pro auta je složen z metod zobrazených na obrázku 14.

```
6 references | Jan Matička, 14 days ago | 2 authors, 3 changes
public interface ICarRepository
{
    2 references | JanHolešínský, 112 days ago | 1 author, 1 change
    Car CreateCar(Car car);
    2 references | JanHolešínský, 112 days ago | 1 author, 1 change
    Car UpdateCar(Car car);
    2 references | JanHolešínský, 112 days ago | 1 author, 1 change
    Car DeleteCar(int carId);
    2 references | JanHolešínský, 112 days ago | 1 author, 1 change
    Car GetCarById(int carId);
    2 references | Jan Matička, 62 days ago | 1 author, 1 change
    ICollection<Car> GetCarsByOwner(string userId);
    2 references | JanHolešínský, 112 days ago | 1 author, 1 change
    ICollection<Car> GetAllCars();
    2 references | Jan Matička, 14 days ago | 1 author, 1 change
    ICollection<Car> GetAllAvailableCars(DateTime dateTimeFrom, DateTime dateTimeTo, int? rideId);
    4 references | JanHolešínský, 112 days ago | 1 author, 1 change
    Task<bool> SaveChangesAsync();
}
```

Obrázek 14 - Metody repositáře, zdroj: vlastní

Konkrétní implementace jednotlivých metod je závislá na umístění dat, pro systém efektivního plánování dopravy byla zvolena relační databáze, ve které jsou za využití LINQ to SQL data ukládána do jednotlivých tabulek. Konkrétní příklad je znázorněn na obrázku 15 a znázorňuje způsob získávání aut pro jednotlivé dispečery.

```
public ICollection<Car> GetCarsByOwner(string userId)
{
    var results =
        _context.Cars.Where(x => x.Owner.Id == userId && !x.IsDeleted)
        .ToList();
    return results;
}
```

Obrázek 15 - Ukázka LINQ to SQL, zdroj: vlastní

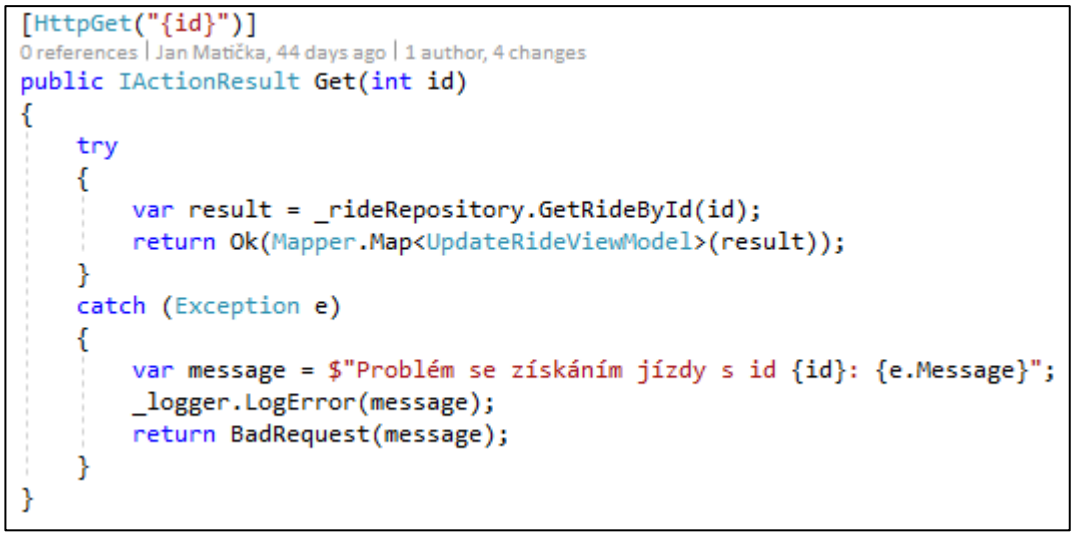
Repositáře jsou využívány v široké míře API rozhraním Smart cargo planning, Každé volání API je zabezpečeno pomocí ASP.NET Identity a jeho dalších rozšíření.

4.3.1 API rozhraní aplikace

Rozhraní API slouží k distribuci dat, v současné době pro mobilní telefony a Javascriptové frameworky, ale samotné API je možné kontaktovat s jakýmkoliv zařízením, které má přístup k internetu. API v systému pro efektivní plánování dopravy dodržuje standardy RESTful API, které jsou popsány v teoretické části. V rozhraní API jsou implementovány jednotlivé CRUD (Create, Read, Update, Delete) operace, při implementaci byly dodrženy následující mezinárodní standardy REST:

- POST
  - Vytváří nové zdroje
- GET
  - Získává zdroje
- PUT
  - Upravuje zdroje
- DELETE
  - Maže zdroje

Na následujícím obrázku 16 je ukázána implementace operace GET, která získává z databáze auto podle jeho unikátního identifikátoru id.



```
[HttpGet("{id}")]
0 references | Jan Matička, 44 days ago | 1 author, 4 changes
public IActionResult Get(int id)
{
    try
    {
        var result = _rideRepository.GetRideById(id);
        return Ok(Mapper.Map<UpdateRideViewModel>(result));
    }
    catch (Exception e)
    {
        var message = $"Problém se získáním jízdy s id {id}: {e.Message}";
        _logger.LogError(message);
        return BadRequest(message);
    }
}
```

Obrázek 16 - Ukázka operace GET, zdroj: vlastní

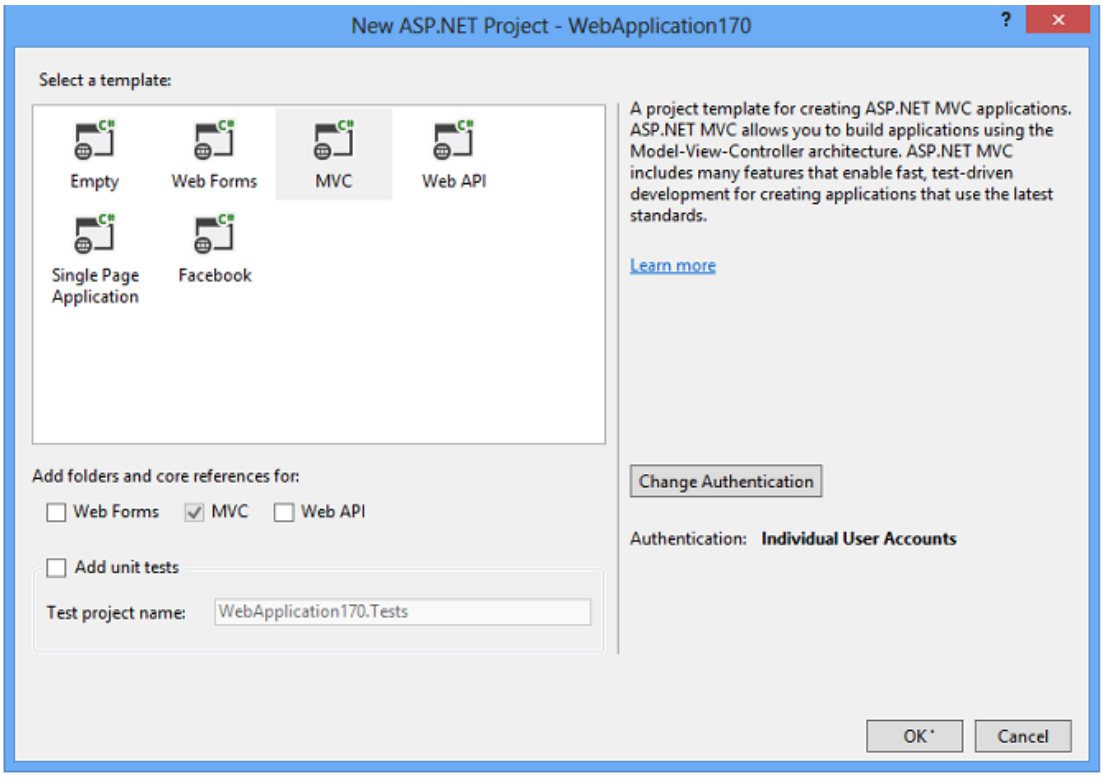
4.4 Zabezpečení systému

Zabezpečení aplikace komplexního systému je náročný úkol, je potřeba, aby bylo zabezpečení jednotné, od jednoho poskytovatele a bylo schopné pokrýt všechny platformy, na kterých se systém vyskytuje i s ohledem na budoucí rozšíření. Bezpečnost Smart cargo planning je zajištěna pomocí ASP.NET Identity, které je popsáno v teoretické části práce.

4.4.1 Integrace ASP.NET Identity

Integrace ASP.NET Identity do projektu je velice snadná, při vytváření aplikace stačí vybrat možnost autorizace uživatelů založenou na jednotlivých účtech, jak je zachyceno na obrázku 17. Do nově vytvořeného projektu jsou přidány tři balíčky:

- Microsoft.AspNet.Identity.EntityFramework
  - Pro persistentní uchování dat o uživatelích
- Microsoft.AspNet.Identity.Core
  - K výběru typu úložiště
- Microsoft.AspNet.Identity.OWIN
  - Na tvorbu cookies



Obrázek 17 - Integrace ASP.NET Identity do projektu, zdroj: [19]

V systému pro efektivní plánování dopravy je použito ASP.NET Identity k autorizaci a autentizaci uživatelů z webového prohlížeče a mobilní aplikace. Pro uživatele využívající webový prohlížeč je jejich ověřování implementováno pomocí Cookies, uživatelé, kteří využívají mobilní aplikaci, jsou ověřeni s využitím JSON Web Tokenů (JWT) tokenů.

Zabezpečení je realizováno pomocí anotací před jednotlivými třídami systému. Každá třída, která je anotována atributem Authorize je zabezpečena a nelze se k ní dostat bez autentizace uživatele, použití anotací je ukázáno na obrázku 18.

```
[Route("api/[controller]")]
[Authorize]
11 references | Jan Holešínský, 13 days ago | 2 authors, 28 changes
public class RidesController : Controller
{
```

Obrázek 18 - Použití anotace při autorizaci, zdroj: vlastní

4.4.2 Použití JSON Web tokenů

JSON Web Token (JWT) je otevřený standard (RFC7519), který definuje kompaktní a samostatný způsob bezpečného přenosu informací mezi účastníky jako objekt JSON. Informace lze ověřit a zároveň jim důvěřovat, protože jsou digitálně podepsány. JWT mohou být podepsány pomocí algoritmu HMAC s využitím tajného klíče nebo páru veřejného soukromého klíče pomocí RSA [30].

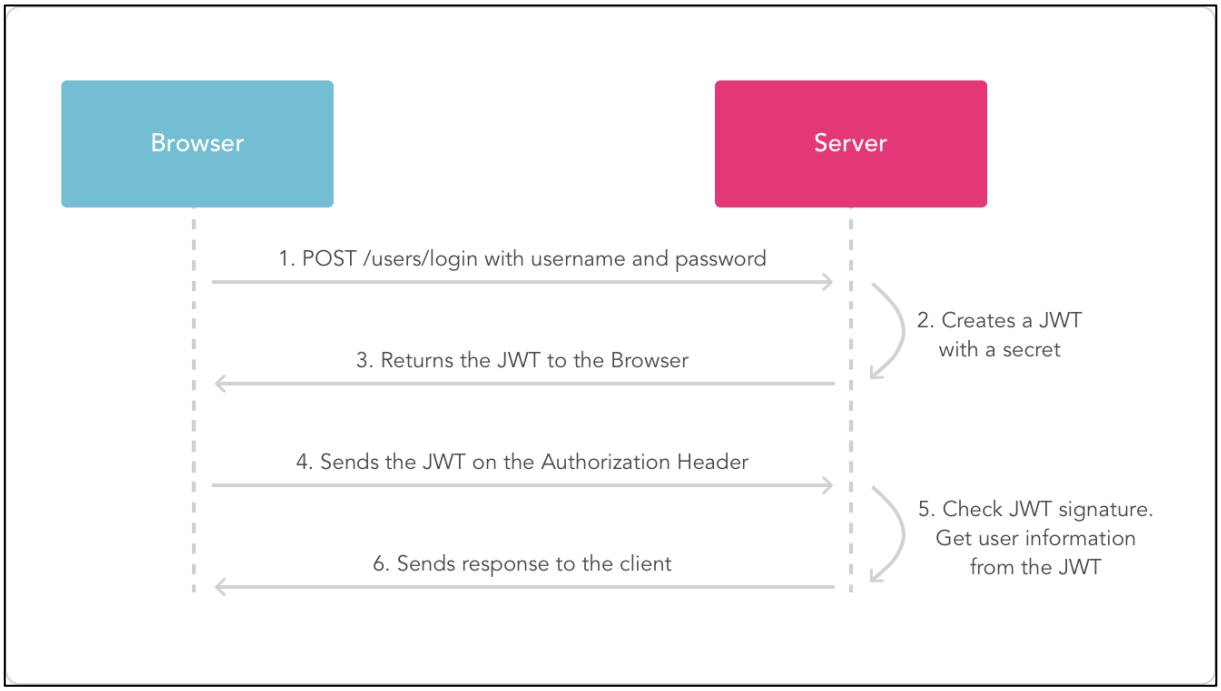
Komunikace mobilních telefonů se systémem Smart cargo planning je zabezpečena implementací JWT. V autentizaci, když se uživatel přihlašuje do systému, pošle své přihlašovací údaje, které systém zkontroluje a následně vrátí token, který je potřeba uložit lokálně na zařízení. Kdykoliv uživatel potřebuje přístup k chráněným datům, systém ho ověří podle tohoto tokenu, který je přiložen v hlavičce žádosti. Konkrétní implementace generování tokenu je ukázána na obrázku 19.

```
var claims = new[]
{
    new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
    new Claim(ClaimTypes.Name,user.UserName)
};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Tokens:Key"]));
var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(
    issuer: _config["Tokens:Issuer"],
    audience: _config["Tokens:Audience"],
    claims: claims,
    expires: DateTime.UtcNow.AddMinutes(15),
    signingCredentials: creds
    );
return Ok(new TokenDto
{
    Token = new JwtSecurityTokenHandler().WriteToken(token),
    Expiration = token.ValidTo
});
```

Obrázek 19 - Implementace generování tokenu, zdroj: vlastní

Jedná se o bezstavový způsob autentizace, protože stav uživatele není uložen na serveru, server pouze kontroluje, jestli je token přítomen, případně validní, a poté uživateli umožní přístup k zabezpečeným informacím, nebo mu vrátí kód 401 Unauthorized. JWT tokeny jsou

samostatné a obsahují veškeré potřebné informace, dochází tedy k redukci dotazů do databáze. Následující diagram na obrázku 20 ukazuje proces použití tokenu.



Obrázek 20 - Diagram použití JWT tokenu, zdroj: [30]

4.5 Mobilní aplikace

Mobilní aplikace systému efektivního plánování dopravy je implementována na technologii Xamarin.Forms, která je popsána v teoretické části práce. Účelem aplikace je usnadnit komunikaci mezi řidičem a dispečerem, konkrétně aplikace zobrazuje řidiči naplánované a dispečerem schválené jízdy na jednotlivé dny. Mobilní aplikace je vytvořena jako nativní aplikace na platformy Android a iOS.

Implementace zdrojového kódu je založena na návrhovém vzoru MVVM (Model–View–ViewModel), je tedy snadno rozšiřitelná a neobsahuje zbytečné závislosti mezi jednotlivými třídami.

Mobilní aplikace umožňuje řidiči zpětně dohledávat vykonanou práci, v případě, že bude potřeba dohledat informace z historie, například na kterém místě vyzvedával daný náklad a zároveň může sloužit jako podklad pro vyplňování docházky.

4.5.1 Funkcionality závislé na platformě

Xamarin.Forms umožňuje sdílet vysoké procento kódu mezi jednotlivými platformami, není ale možné mít vše napsané pouze jednou. Klíčovou vlastností mobilní aplikace Smart cargo planning je možnost kontaktovat přímo z telefonu místa naložení a vyložení nákladu. Tato funkcionalita musí být naprogramována přímo pro každou platformu. Ve sdílené části aplikace je potřeba vytvořit interface, znázorněný na obrázku 21, který bude sloužit jako abstrakce a který bude obsahovat metodu pro volání telefonního čísla.

```
2 references | Jan Holešínský, 25 days ago | 1 author, 1 change
public interface IDialer
{
    1 reference | Jan Holešínský, 25 days ago | 1 author, 1 change
    Task<bool> DialAsync(string number);
}
```

Obrázek 21 - Interface pro volání z telefonu, zdroj: vlastní

V jednotlivých částech, které jsou specifické pro každou platformu, je potřeba tento interface implementovat konkrétním voláním systémových funkcí pro uskutečnění telefonického hovoru. Na platformu Android je kód popsán v příloze C. Jako kontrast je přidán obrázek 22, který ukazuje porovnání, jak je funkcionalita implementována na platformě iOS.

```
1 reference | Jan Holešínský, 25 days ago | 1 author, 1 change
public class PhoneDialer : IDialer
{
    0 references | Jan Holešínský, 25 days ago | 1 author, 1 change
    public Task<bool> DialAsync(string number)
    {
        return Task.FromResult(
            UIApplication.SharedApplication.OpenUrl(
            new NSUrl("tel:" + number))
        );
    }
}
```

Obrázek 22 - Implementace telefonického hovoru iOS, zdroj: vlastní

V následujícím kroku je potřeba dát vše dohromady ve sdílené části kódu, kde je potřeba rozlišit, kdy zavolat metody pro Android a kdy pro iOS. Tuto funkcionalitu umožňuje třída DependencyService, která rozlišuje volání z jednotlivých platforem a vyhledá potřebné

implementace konkrétního interface na jednotlivých platformách. Na následujícím obrázku 23 je ukázáno použití pro telefonický hovor.

```
var dialer = DependencyService.Get<IDialer>();
if (dialer != null)
{
    await dialer.DialAsync(number.ToString());
}
```

Obrázek 23 - Použití DependencyService, zdroj: vlastní

Funkcionality, které je možné sdílet mezi platformami, jsou implementovány ve sdílené části projektu, v příloze D je ukázána třída, která umožňuje přihlášení uživatele a následné zpracování JWT tokenu.

Sdílené funkcionality jsou v průběhu nasazení aplikace na konkrétní mobilní telefon se specifickým operačním systémem zkonvertovány do nativní aplikace, která je snadno distributovatelná v síti oficiálních obchodů. V blízké budoucnosti bude aplikace nasazena na obchod Play a App Store pro dosažení optimálního šíření mezi koncovými zákazníky systému pro efektivní plánování dopravy.

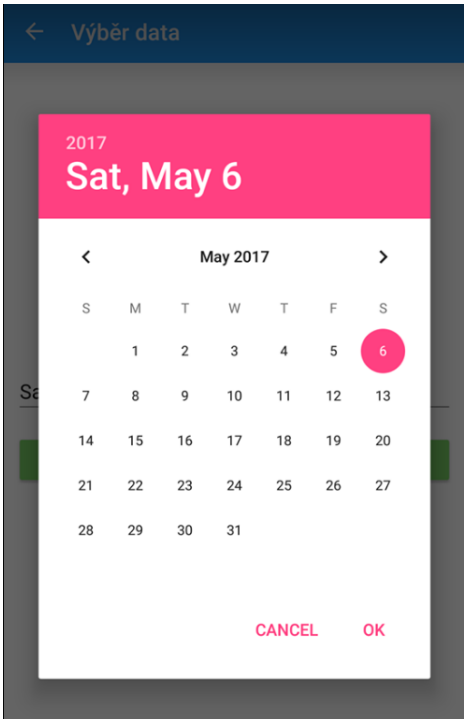
4.5.2 Funkcionality mobilní aplikace

V předchozí části práce je zmíněno, že aplikace je vytvořena na platformy iOS a Android, co se týče funkcionality, je uživatelům dodána aplikace, na které není znatelný rozdíl mezi těmito platformami, a je zde kladen důraz na vysokou uživatelskou přívětivost.

Pro použití aplikace je nutné, aby se uživatelé, kterými nejspíše budou jednotliví řidiči, nejdříve přihlásili. Přihlášení je velice jednoduché, je požadováno pouze vyplnění přihlašovacích údajů, které se skládají z uživatelského jména a hesla. V případě chybného zadání přihlašovacích údajů je uživatel okamžitě upozorněn, v aplikaci se neodehrává žádná akce bez vědomí uživatele. Po úspěšném zadání přihlašovacích údajů je v aplikaci zobrazen ukazatel v podobě grafické animace, zobrazen na obrázku 24, upozorňující na fakt, že přihlášení probíhá.

Obrázek 24 - Přihlašování uživatele, zdroj: vlastní

Jakmile je uživatel přihlášen, zobrazí se obrazovka s výběrem data. Zobrazení je důležité z důvodu možnosti kontroly práce provedené v minulosti a také umožňuje získání podkladů pro výkon práce, která řidiče teprve čeká. Vše je usnadněno emailovým upozorněním, které přichází okamžitě se zpřístupněním nových jízd pro jednotlivé řidiče. Výběr data je prováděn přes systémové komponenty platforem Android a iOS., na obrázku 25, pro příklad je uvedeno znázornění výběru data na platformě Android.



Obrázek 25 - Výběr data platforma Android, zdroj: vlastní

Po potvrzení výběru data následuje opět zobrazení stavu aplikace, které, jak je vidět na obrázku 26, plynule přechází na výpis jednotlivých úkonů řidiče ve zvoleném datu.

Obrázek 26 - Výpis činností řidiče, zdroj: vlastní

Řidiči je umožněno na každou jednotlivou činnost kliknout, aby se zobrazily detaily jízdy. Tento krok je velice podstatný, protože v detailu jízdy jsou zachyceny veškeré informace, které řidič potřebuje o konkrétní jízdě znát, pro ilustraci přidávám obrázek 27. Zároveň je zde využita výše popsaná funkcionalita telefonického hovoru, aby řidič mohl přímo kontaktovat klíčová místa jeho trasy.

Obrázek 27 - Detaily jízdy, zdroj: vlastní

5 ZÁVĚR

Cílem práce bylo vytvořit systém pro efektivní plánování nákladní přepravy s důrazem na silniční nákladní dopravu. S využitím programovacího jazyka C# a technologie Xamarin.Forms byly vytvořeny nativní mobilní aplikace na platformy iOS a Android, databázové řešení bylo implementováno s využitím technologie objektově relačního mapování a Entity Framework na serveru MS SQL. K celé aplikaci bylo vytvořeno rozhraní umožňující komunikaci s externími zařízeními a programy pomocí standardu REST.

Před zahájením implementace aplikace byla provedena analýza jednotlivých technologií, které by bylo reálné použít, s důrazem kladeným na dostupnost a zabezpečení aplikace. Zvoleno bylo komplexní řešení od firmy Microsoft a služby Azure, která je postavena na cloudových technologiích a využívá nejmodernější technologie 21. století.

Návrh databázového řešení byl proveden v souladu s normálními formami určenými pro tvorbu databází, konkrétně byla použita třetí normální forma, ve které neexistuje závislost mezi dvěma neklíčovými atributy.

Pro vývoj aplikace bylo nejprve nutné vybrat vhodné technologie a vyzkoušet, zda jsou všechny mezi sebou kompatibilní. Po ukončení tohoto kroku bylo již možné začít tvořit systém jako celek. V průběhu implementace systému se vyskytly problémy s výkonem, aplikace tedy musela být značně optimalizována. V současné době je aplikace testována v reálném prostředí firmou, která se specializuje na nákladní silniční dopravu.

Systém efektivního plánování nákladní dopravy umožňuje komplexní plánování jednotlivých cest, správu vozového parku, jednotlivých zákazníků a uživatelských účtů. Smart cargo planning optimalizuje a automatizuje komunikaci mezi dispečerem a řidičem s využitím nativních mobilních aplikací.

Systém je složen z dispečerské aplikace, která je implementována jako webová aplikace na platformě ASP.NET a slouží jednotlivým dispečerům pro efektivní plánování výkonů jednotlivých nákladních automobilů. Další část systému zahrnuje mobilní aplikaci, která je určena pro řidiče a umožňuje jim získávat informace nejen o aktuální dopravě, ale také o již dříve uskutečněných dopravách. Mobilní aplikace je vytvořena na platformy Android a iOS s využitím technologie Xamarin.Forms.

Aplikace je automaticky synchronizována, aby vždy zobrazovala aktuální data. Synchronizace je dosaženo s využitím aplikačního rozhraní, které slouží k administraci a zpracování dat.

Aplikační rozhraní je naprogramováno podle architektonických principů RESTful API s implementací CRUD operací.

Při implementaci systému byla zohledněna současná bezpečnostní rizika, současně byl brán ohled na vnější i vnitřní útoky, proto je celý systém důkladně zabezpečen s využitím technologie ASP.NET Identity. Bezpečnostní rizika jsou pokryta v mobilní aplikaci, javascriptových frameworcích i webové aplikaci.

V současné době systém disponuje základními funkcionalitami a je k dispozici vybranému komerčnímu subjektu v oblasti nákladní dopravy na testování. V budoucnu je plánováno systém dále rozšířit a navázat spolupráci s dalšími firmami z oblasti nákladní dopravy. Systém pro efektivní plánování nákladní dopravy je zcela univerzální, je možné aplikovat rozšíření i na jiné druhy nákladní dopravy.

Systém je připraven na připojení internetu věcí neboli IoT, poté bude možné detailně sledovat chování jednotlivých automobilů i řidičů na cestách a veškeré informace použít jako podklad pro strojové učení, které zajistí dokonalejší optimalizaci plánování jednotlivých tras a chování automobilů. Zároveň bude možné předcházet opotřebení a poruchám dopravních prostředků.

V rámci diplomové práce byly splněny všechny zadané cíle, byl vytvořen plně funkční optimalizovaný a otestovaný systém složený z databázové části, mobilní aplikace, komunikačního rozhraní a komplexní serverové části, který disponuje značným tržním potenciálem.

LITERÁRNÍ ZDROJE

[1] Oracle Big Data: The Road to Big Data. *Oralce: Integrated Cloud Applications & Platform Services* [online]. California: Oracle, 2017 [cit. 2017-04-14]. Dostupné z: https://www.oracle.com/big-data/index.html

[2] Freight transport statistics. *Eurostat: Statistics Explained* [online]. Lucembursko: European Commission, 2017 [cit. 2017-04-11]. Dostupné z: http://ec.europa.eu/eurostat/statistics-explained/index.php/Freight_transport_statistics#Road_freight

[3] LEE, Jay, Berhard BAGHERI a Hung-An KAO. *Proceeding of Int. Conference on Industrial Informatics* (INDIN) 2014 Recent Advances and Trends of Cyber-Physical Systems and Big Data Analytics in Industrial Informatics Recent Advances and Trends of Cyber-Physical Systems and Big Data Analytics in Industrial Informatics. Conference on Industrial Informatics [online]. Porot Alegre, 2014, , 1-5 [cit. 2017-04-14]. Dostupné z: https://www.researchgate.net/publication/266375284_Recent_Advances_and_Trends_of_Cyber-Physical_Systems_and_Big_Data_Analytics_in_Industrial_Informatics

[4] RÜßMANN, Michael, Markus LORENZ, Philipp GERBERT, Manuela WALDNER, Pascal ENGEL a Michael HARNISCH. *Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries* [online]. 2015, 5 [cit. 2017-04-14]. Dostupné z: http://www.inovasyon.org/pdf/bcg.perspectives_Industry.4.0_2015.pdf

[5] PROVOST, Foster a Tom FAWCETT. Data Science and its Relationship to Big Data and Data-Driven Decision Making. *Big Data* [online]. 2013, 1(1), 51-59 [cit. 2017-04-14]. DOI: 10.1089/big.2013.1508. ISSN 2167-6461. Dostupné z: http://online.liebertpub.com/doi/abs/10.1089/big.2013.1508

[6] HAYES, Brian. Cloud computing. *Communications of the ACM* [online]. 2008, 51(7), 9- [cit. 2017-04-16]. DOI: 10.1145/1364782.1364786. ISSN 00010782. Dostupné z: http://portal.acm.org/citation.cfm?doid=1364782.1364786

[7] Co je cloud computing? *Microsoft Azure* [online]. Seattle: Microsoft, 2017 [cit. 2017-04-16]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-cloud-computing/

[8] STEHLÍKOVÁ, Blanka. *Cloud Computing* [online]. Praha, 2010 [cit. 2017-04-16]. Dostupné z: https://support.dce.felk.cvut.cz/mediawiki/images/c/c8/Dp_2010_setikovska_blanka.pdf. Diplomová práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Otto Cerman.

[9] What is the Internet of Things? Everything you need to know about IoT: With countless companies investing in the Internet of Things, we ask: what exactly is it? What are the risks? And how will it impact our lives? Here is everything you need to know. In: *Techworld* [online]. London: IDG Communications, 2015 [cit. 2017-04-16]. Dostupné z: http://www.techworld.com/big-data/what-is-internet-of-things-3631109/

[10] IoT reshapes transportation, whether driving down the street or flying at 30,000 feet Read more at https://blogs.microsoft.com/iot/2016/04/27/iot-reshapes-transportation-whether-driving-down-the-street-or-flying-at-30000-feet/#dr1LoamC8zr2wOHv.99. In: *Internet of Things* [online]. Seattle: Microsoft, 2016 [cit. 2017-04-16]. Dostupné z: https://blogs.microsoft.com/iot/2016/04/27/iot-reshapes-transportation-whether-driving-down-the-street-or-flying-at-30000-feet/

[11] Fathym's IoT-enabled WeatherCloud enhances driver safety during inclement weather Read more at https://blogs.microsoft.com/iot/2016/12/09/fathyms-iot-enabled-weathercloud-enhances-driver-safety-during-inclement-weather/#PomAHuz0cRUFd3lE.99. In: *Internet of Things* [online]. Seattle: Microsoft, 2016 [cit. 2017-04-16]. Dostupné z: https://blogs.microsoft.com/iot/2016/12/09/fathyms-iot-enabled-weathercloud-enhances-driver-safety-during-inclement-weather/

[12] CHAPPELL, David. INTRODUCING AZURE MACHINE LEARNING: A GUIDE FOR TECHNICAL PROFESSIONALS. In: *Microsoft Azure* [online]. Seattle: Microsoft, 2017 [cit. 2017-04-16]. Dostupné z: http://download.microsoft.com/download/3/B/9/3B9FBA69-8AAD-4707-830F-6C70A545C389/Introducing_Azure_Machine_Learning.pdf

[13] Introduction to ASP.NET Core. *Microsoft Docs* [online]. Seattle: Microsoft, 2017 [cit. 2017-04-17]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/

[14] What is NuGet? *Nuget* [online]. Seattle: .NET foundation, 2017 [cit. 2017-04-17]. Dostupné z: https://www.nuget.org/

[15] Introduction to Dependency Injection in ASP.NET Core. *Microsoft Docs* [online]. Seattle: Microsoft, 2017 [cit. 2017-04-17]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection

[16] Building Your First Web API with ASP.NET Core MVC and Visual Studio. *Microsoft Docs* [online]. Seattle: Microsoft, 2017 [cit. 2017-04-17]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api

[17] RESTful Web services: The basics. *IBM* [online]. New York: IBM developerWorks, 2008 [cit. 2017-04-17]. Dostupné z: https://www.ibm.com/developerworks/webservices/library/ws-restful/

[18] ŠEFRÁNEK, Jan. *Bezpečnostní specifika provozu aplikací v prostředí cloudu* [online]. Praha, 2013 [cit. 2017-04-18]. Dostupné z: https://dip.felk.cvut.cz/browse/pdfcache/sefraja1_2013bach.pdf. Bakalářská práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Pavel Náplava.

[19] GALLOWAY, Roy, Pranav RASTOGI, Rick ANDERSON a Tom DYKSTRA. Introduction to ASP.NET Identity. In: *Microsoft Docs* [online]. Seattle: Microsoft, 2015 [cit. 2017-04-18]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

[20] MILLER, Rowan a Martin MILAN. Entity Framework Core. In: *Microsoft Docs* [online]. Seattle: Microsoft, 2016 [cit. 2017-04-20]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/

[21] Dickson, Jared, "Xamarin Mobile Development" (2013). *Technical Library.* 167. http://scholarworks.gvsu.edu/cistechlib/167

[22] What Industry 4.0 Means for Manufacturers. In: *AETHON* [online]. Pittsburgh: AETHON, 2015 [cit. 2017-04-20]. Dostupné z: http://www.aethon.com/industry-4-0-means-manufacturers/

[23] What is cloud computing? In: *IBM* [online]. New York: IBM, 2014 [cit. 2017-04-20]. Dostupné z: https://www.ibm.com/blogs/cloud-computing/2014/03/what-is-cloud-computing-2/

[24] IoT – Internet Of Things. In: *Meccanismo Complesso* [online]. Meccanismo Complesso, 2016 [cit. 2017-04-20]. Dostupné z: https://www.meccanismocomplesso.org/en/iot-internet-of-things/

[25] WENZEL, Maira a Philip CARTER. .NET Architectural Components. In: *Microsoft .NET* [online]. Seattle: Microsoft, 2016 [cit. 2017-04-20]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/articles/standard/components

[26] Dependency Injection. In: *Microsoft Developer Network* [online]. Seattle: Microsoft [cit. 2017-04-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff921152.aspx

[27] Xamarin.Forms: Build native UIs for iOS, Android and Windows from a single, shared C# codebase. In: *Xamarin* [online]. San Francisco: Xamarin, 2017 [cit. 2017-04-20]. Dostupné z: https://www.xamarin.com/forms

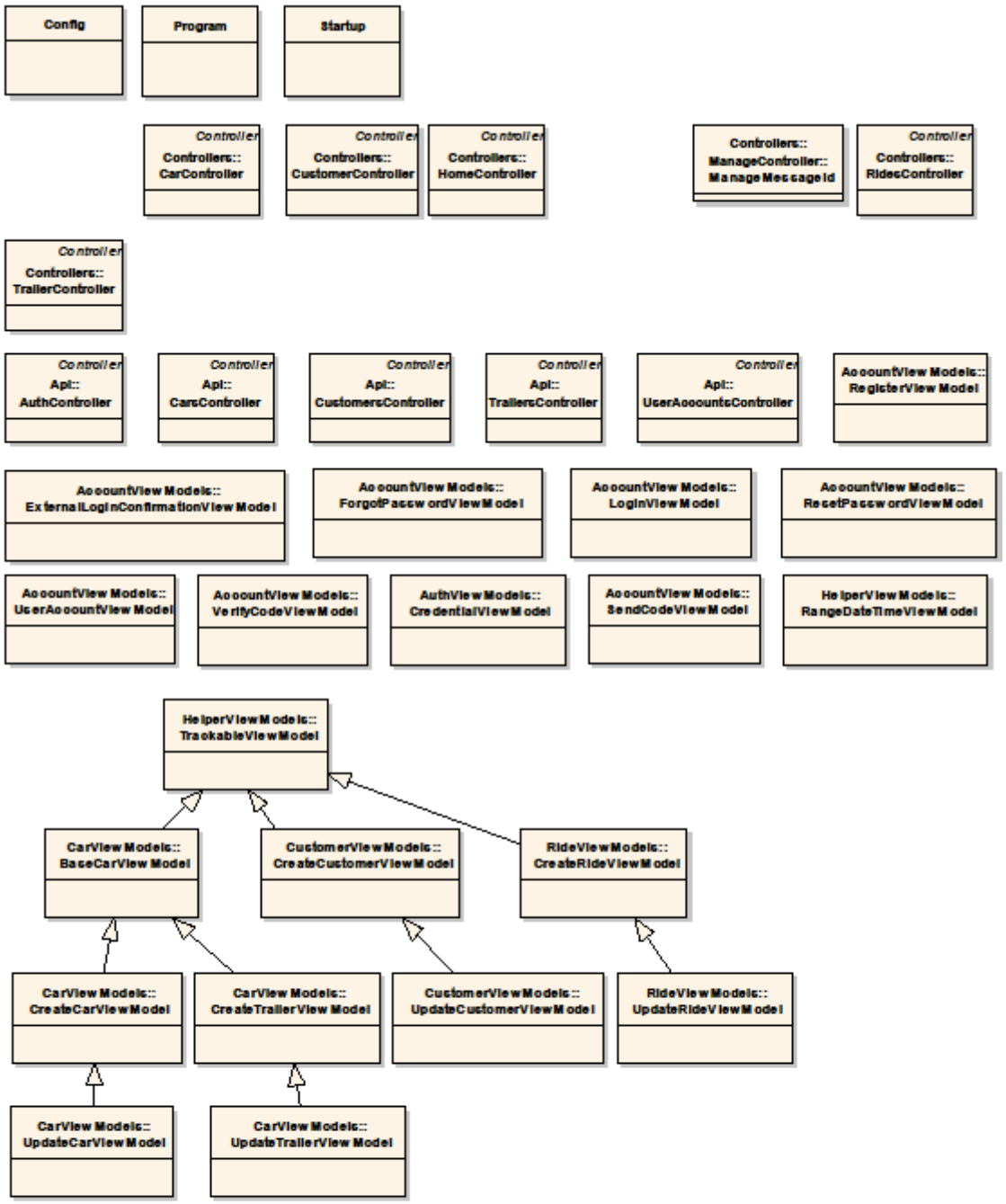
[28] What is Entity Framework? In: *Entity Framework Tutorial* [online]. 2016 [cit. 2017-04-20]. Dostupné z: http://www.entityframeworktutorial.net/what-is-entityframework.aspx

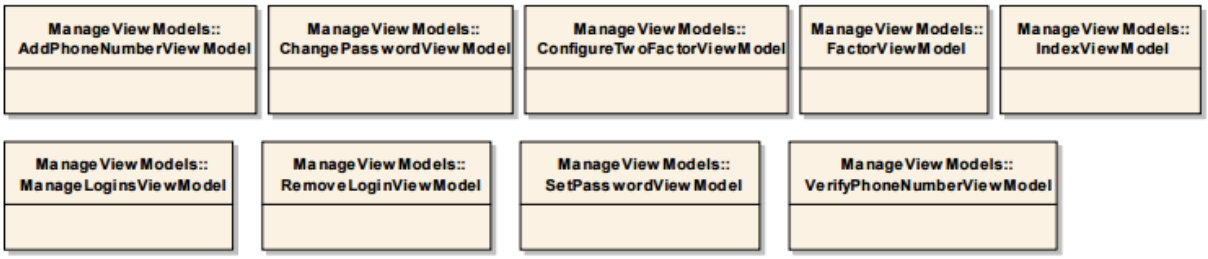
[29] GAILEY, Glenn. Get started guide for Azure developers. In: *Microsoft Docs* [online]. Seattle: Microsoft, 2016 [cit. 2017-04-20]. Dostupné z: https://docsmsftpdfs.blob.core.windows.net/guides/azure/azure-developer-guide.pdf

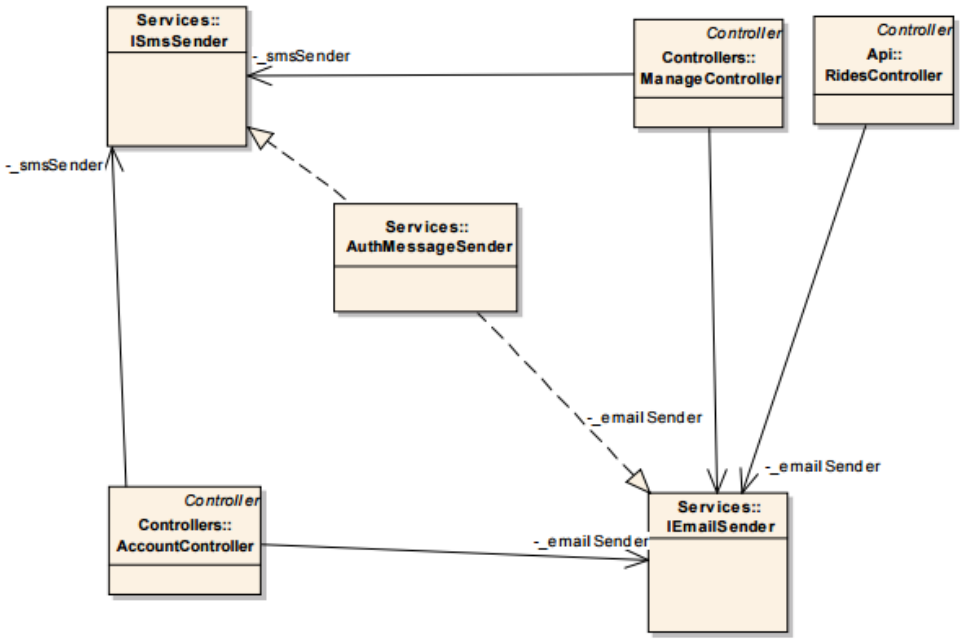
[30] Introduction to JSON Web Tokens. *JWT* [online]. Bellevue: Auth0, 2013 [cit. 2017-04-26]. Dostupné z: https://jwt.io/introduction/

PŘÍLOHY

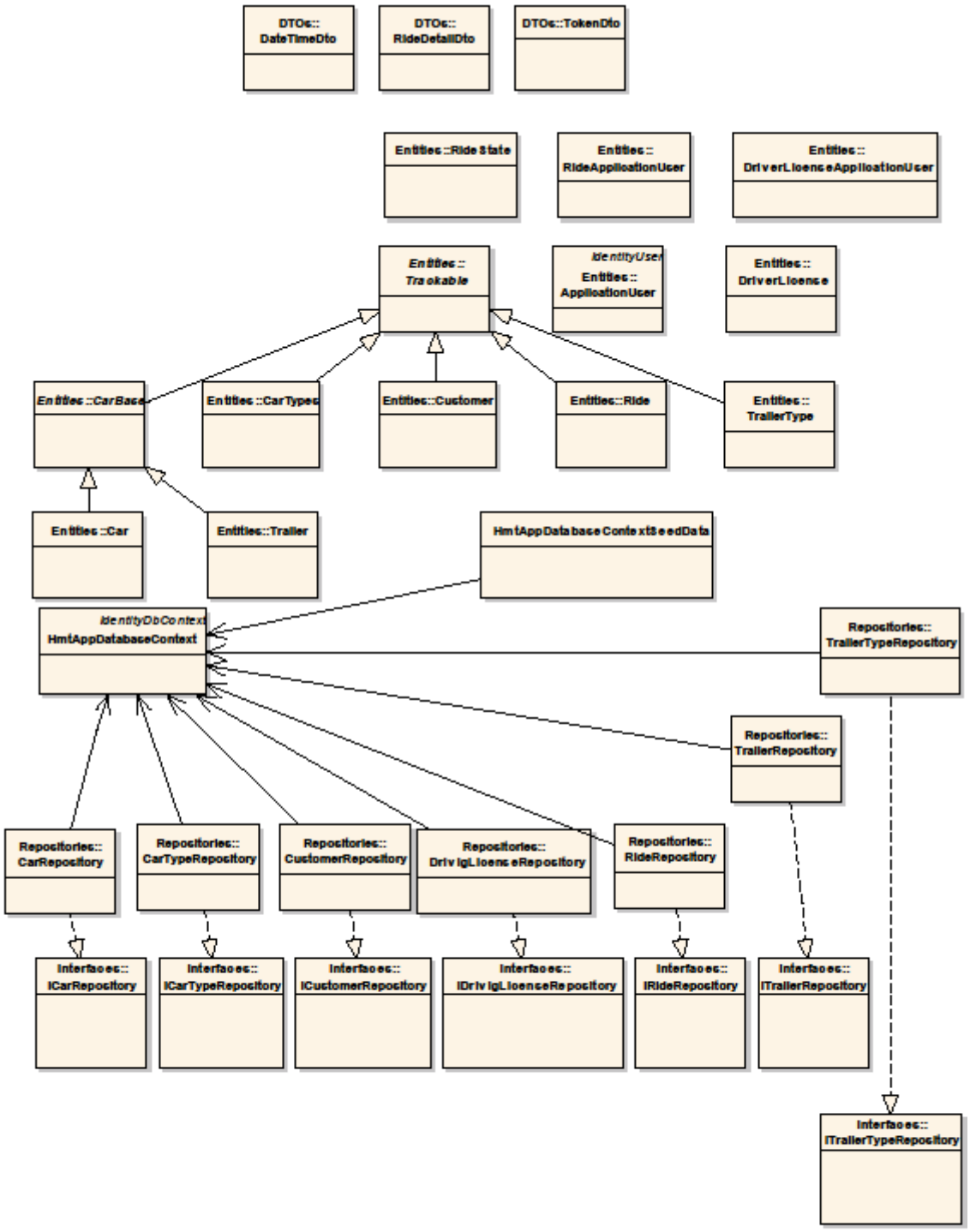
Příloha A – *Diagram tříd serverové části*




ManageViewModels::
AddPhoneNumberViewModel
ManageViewModels::
ChangePasswordViewModel
ManageViewModels::
ConfigureTwoFactorViewModel
ManageViewModels::
FactorViewModel
ManageViewModels::
IndexViewModel
ManageViewModels::
ManageLoginsViewModel
ManageViewModels::
RemoveLoginViewModel
ManageViewModels::
SetPasswordViewModel
ManageViewModels::
VerifyPhoneNumberViewModel


Services::
ISmsSender
-_smsSender
Controller
Controllers::
ManageController
Controller
Api::
RidesController
-_smsSender
Services::
AuthMessageSender
-_emailSender
-_emailSender
Controller
Controllers::
AccountController
-_emailSender
Services::
IEmailSender

Příloha B – *Diagram tříd datové vrstvy*



Příloha C – *Zdrojový kód pro telefonický hovor na platformě Android*

```
[assembly: Dependency(typeof(PhoneDialer))]
namespace HmtMobile.Droid
{
    public class PhoneDialer : IDialer
    {
        /// <summary>
        /// Dial the phone
        /// </summary>
        public Task<bool> DialAsync(string number)
        {
            var context = Forms.Context;
            if (context != null)
            {
                var intent = new Intent(Intent.ActionCall);
                intent.SetData(Android.Net.Uri.Parse("tel:" + number));

                if (IsIntentAvailable(context, intent))
                {
                    context.StartActivity(intent);
                    return Task.FromResult(true);
                }
            }

            return Task.FromResult(false);
        }

        /// <summary>
        /// Checks if an intent can be handled.
        /// </summary>
              public static bool IsIntentAvailable(Context context, Intent intent)
        {
            var packageManager = context.PackageManager;

            var list = packageManager.QueryIntentServices(intent, 0)
                .Union(packageManager.QueryIntentActivities(intent, 0));
            if (list.Any())
                return true;

            var mgr = TelephonyManager.FromContext(context);
            return mgr.PhoneType != PhoneType.None;
        }
    }
}
```

Příloha D – *Zdrojový kód třídy LoginHelper*

```
public class LoginHelper
    {
        public async Task<TokenModel> Login(CredentialDto credentials)
        {
            using (HttpClient client = new HttpClient())
            {

                var jsonObject = JsonConvert.SerializeObject(new CredentialDto {
UserName = credentials.UserName, Password = credentials.Password });
                var content = new StringContent(jsonObject, Encoding.UTF8,
"application/json");


                HttpResponseMessage response = await client.PostAsync(new
Uri("http://smartcargoplanning.azurewebsites.net/api/auth/token"), content);

                if (response.IsSuccessStatusCode)
                {


                    var result = await response.Content.ReadAsStringAsync();
                    var token = JsonConvert.DeserializeObject<TokenModel>(result);
                    SaveCredentials(credentials,token);
                    return token;
                }
            }
            return null;
        }
        private void SaveCredentials(CredentialDto credentials,TokenModel token)
        {
            var properties = new Dictionary<string, string>
            {

                    { "password", credentials.Password },
                    { "token", token.Token },
                    { "expiration", token.Expiration.ToString() }

            };

            var account = new Account(credentials.UserName, properties);
            AccountStore.Create().Save(account, Constants.AppName);

            var accountReaded =
AccountStore.Create().FindAccountsForService(Constants.AppName).FirstOrDefault();
        }

        public void DeleteCredentails()
        {
            var account =
AccountStore.Create().FindAccountsForService(Constants.AppName).FirstOrDefault();
            if (account != null)
            {
                AccountStore.Create().Delete(account, Constants.AppName);
            }
        }

    }
```