FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Optimalizace rozhraní a přiřazování zásilek v kurýrní společnosti |
| **Student:** | Bc. Jakub Lacný |
| **Vedoucí:** | Ing. Jaroslav Kuchař, Ph.D. |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Webové a softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2020/21 |

## Pokyny pro vypracování

Ve společnosti Messenger, která se zabývá primárně expresní přepravou zásilek, řeší operátoři každodenně problém s přiřazováním zásilek mezi pracující kurýry. Cílem práce je analyzovat současný stav a zaměřit se na automatizaci či optimalizaci problému.

- Proveďte průzkum aktuálního stavu uživatelského rozhraní webové aplikace pro operátory.
- Zpracujte rešerši používaných technologií a principů v oblasti přiřazování zásilek mezi více kurýrů.
- Na základě výstupů průzkumu a rešerše navrhněte podpůrný optimalizační systém pro operátory zahrnující:
  - způsob navrhování přiřazování zásilek kurýrům např. na základě polohy, historických dat o trasách apod.
  - změny uživatelského rozhraní s cílem rychlejší a intuitivnější práce.
- Implementujte prototyp obsahující navržené prvky.
- Proveďte řádné testování zahrnující reálná data a reálné uživatele dle možností společnosti.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 15. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

# Optimalizace rozhraní a přiřazování zásilek v kurýrní společnosti

***Bc. Jakub Lacný***

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jaroslav Kuchař, PhD.

29. července 2020

# Poděkování

Chtěl bych tímto poděkovat vedoucímu mé diplomové práce, panu Ing. Jaroslavu Kuchařovi, Ph.D. za zodpovědné vedení a časté konzultace.

Dále chci poděkovat všem pracovníkům společnosti MESSENGER za možnost zpracovat tohle téma a za mnoho rozhovorů a rad.

Velké poděkování patří také mým rodičům a přátelům za podporu během celých pěti let na vysoké škole.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. července 2020 ……

### Odkaz na tuto práci

Lacný, Jakub. *Optimalizace rozhraní a přiřazování zásilek v kurýrní společnosti*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

# Abstrakt

Cílem práce je zanalyzovat interní procesy v kurýrní společnosti MESSENGER a na jejich základě navrhnout, implementovat a otestovat serverovou část modulu, který bude optimalizovat proces přiřazování objednávek kurýrům. Výsledkem je otestovaný optimalizační modul připravený na postupné zařazení do produkčního prostředí a návrhy možného začlenění daných optimalizací do již existujícího uživatelského rozhraní. K optimalizaci přiřazování jsou využity analýzy míst vyzvednutí a doručení, umístění kurýrů a aktuální zatížení kurýrů. Implementace probíhá v programovacím jazyku Java s použitím frameworků Spring a Hibernate. Využita je databáze MSSQL a fulltextový vyhledávač Elasticsearch.

**Klíčová slova** Kurýrní společnost, Přiřazování zásilek, Optimalizace, Návrh uživatelského rozhraní

# Abstract

The purpose of this thesis is to analyse the internal processes in the MESSENGER courier company, and then to design, implement and evaluate the server part of a new optimization module. This module optimizes the process of assigning couriers to consignments. The results are a tested optimization module ready for gradual integration into the production environment, and the designs for the possible integration of the optimizations into the existing user interface. The current real-time load of couriers and analysis of pickup and delivery places are used to optimize assigning. The module is implemented using Java programming language with the help of the Spring and Hibernate frameworks. The data is stored and queried with MSSQL and ElasticSearch.

**Keywords** Courier company, Consignment assignment, Optimization, User interface design

# Obsah

**Úvod** **1**
Seznámení s problematikou . . . . . 1
Teoretický základ . . . . . 1
Cíl práce . . . . . 2
Struktura práce . . . . . 3

**1 Analytická část** **5**
1.1 Reálná řešení . . . . . 5
1.2 Analýza informačního systému ve společnosti MESSENGER . . 7
1.3 Architektura informačního systému . . . . . 12
1.4 Dostupná data vhodná k použití . . . . . 13
1.5 Problémy specifické pro společnost MESSENGER . . . . . 14
1.6 Proces doručení zásilky . . . . . 15
1.7 Analýza možných metod optimalizace . . . . . 19
1.8 Nielsenova heuristická analýza . . . . . 21
1.9 Heuristická analýza modulu operátora . . . . . 22
1.10 Analýza modulu operátora provedená s pracovníky . . . . . 24
1.11 Problémy v aktuálním rozhraní operátorů . . . . . 24
1.12 Možná zdokonalení vyplývající z analýzy . . . . . 25

**2 Návrhová část** **27**
2.1 Návrh optimalizačního modulu . . . . . 27
2.2 Návrh architektury . . . . . 28
2.3 Komparátory . . . . . 29
2.4 Návrh zakomponování doporučovacího modulu do rozhraní operátora . . . . . 31
2.5 Vyhodnocení návrhů integrace s pracovníky . . . . . 42

**3 Implementace prototypu optimalizátoru** **45**

3.1 Struktura modulu . . . . . 45
3.2 Veřejné rozhraní modulu . . . . . 45
3.3 Přístup k datům . . . . . 46
3.4 Práce s geolokačními údaji . . . . . 47
3.5 Implementace komparátorů . . . . . 48
3.6 Zpracování požadavku . . . . . 48
3.7 Předávání závislostí v rámci modulu . . . . . 48

**4 Vývoj a testování komparátorů** **51**
4.1 Úvod . . . . . 51
4.2 Popis zvolených testovacích dat . . . . . 52
4.3 Otestování funkčnosti kostry modulu na komparátoru s náhodným výsledkem . . . . . 54
4.4 Základní komparátor založený na vzdálenosti . . . . . 54
4.5 Komparátor rozšířený o analýzu místa vyzvednutí a doručení . 55
4.6 Komparátor s penalizací . . . . . 58
4.7 Pokročilý komparátor . . . . . 59
4.8 Zjednodušení komparátoru . . . . . 61
4.9 Zhodnocení vývoje komparátoru . . . . . 61

**5 Testovací část** **63**
5.1 Jednotkové testy . . . . . 63
5.2 Výkonnostní testování . . . . . 64

**Závěr** **67**
Zhodnocení vývoje . . . . . 67
Přínos pro autora . . . . . 67
Možnosti budoucího vývoje . . . . . 68

**Literatura** **69**

**A Seznam použitých zkratek** **73**

**B Zprovoznění zdrojových kódů** **75**

**C Obsah přiloženého CD** **77**

# Seznam obrázků


1.1 Nástroj Routific . . . . . 6
1.2 Nástroj Route4Me . . . . . 6
1.3 Architektura informačního systému . . . . . 13
1.4 Rozhraní operátora . . . . . 17
1.5 Mapa s polohami kurýrů . . . . . 18

2.1 Návrh architektury doporučovacího modulu . . . . . 28
2.2 Analýza již přiřazených zásilek - nepříznivá situace . . . . . 30
2.3 Analýza již přiřazených zásilek - příznivá situace . . . . . 30
2.4 Ukázka notifikace v informačním systému . . . . . 31
2.5 Zobrazování doporučení pomocí notifikací . . . . . 32
2.6 Ukázka modálu pro přiřazení kurýra . . . . . 34
2.7 Zobrazování zakomponované do modulu operátora . . . . . 35
2.8 Zobrazování zakomponované do modulu operátora – modální okno 35
2.9 Integrace do rozhraní mapy kurýrů . . . . . 37
2.10 Integrace do rozhraní mapy kurýrů – zvýraznění . . . . . 38
2.11 Zobrazování na separátní stránce . . . . . 40
2.12 Zobrazování na separátní stránce – modální okno . . . . . 41
2.13 Úprava rozhraní na základě konzultace . . . . . 43
2.14 Přidání směru kurýra na základě konzultace . . . . . 44

3.1 Sekvenční diagram zpracování požadavku na doporučení kurýra . . 49

4.1 Vizualizace bodů všech objednávek v testovací sadě . . . . . 53

# Seznam tabulek

4.1 Souhrnné výsledky vývoje komparátoru . . . . . 61

5.1 Výsledky výkonnostního testování . . . . . 65

# Seznam ukázek kódu


3.1 Veřejné rozhraní modulu . . . . . 46
3.2 Ukázka dotazu do Elasticsearch . . . . . 47
3.3 Ukázka použití Dependency Injection . . . . . 49
3.4 Ukázka třídy sloužící k předání závislostí . . . . . 50
4.1 Dotaz do Elasticsearch na přiřazené zásilky . . . . . 59
5.1 Ukázka jednotkového testu . . . . . 64

# Seznam pseudokódů

1 Kombinace vzdálenosti a shody místa vyzvednutí a doručení . . 56
2 Výpočet penalizace kurýrů za počet přiřazených zásilek . . . . . 58

# Úvod

## Seznámení s problematikou

V roce 1991 byla v Praze založena první kurýrní služba, a to akciová společnost MESSENGER. Jednalo se v České republice o zcela novou službu, a i proto byla ze začátku využívána primárně zahraničními společnostmi. Postupně se však stabilizovala a nyní představuje v Praze největší kurýrní společnost, která postupně proniká i do celostátní a mezinárodní přepravy (na Slovensku již také funguje jedna pobočka).

Přestože se již podařilo postupně rozšířit působnost i na dálkovou přepravu, největší podíl objednávek tvoří stále expresní přeprava po Praze. Právě v této oblasti je široký prostor na automatizaci v oblasti přiřazování zásilek mezi kurýry.

Za téměř třicet let existence je dostupná spousta dat a informací o trendech v přiřazování a následném doručování. Jelikož autor v MESSENGERu pracuje jako vývojář, tak má již poměrně široké znalosti o interních procesech a možných využitelných trendech. Práce bude také často diskutována s ostatními pracovníky.

## Teoretický základ

O algoritmickou optimalizaci se akademičtí pracovníci pokoušejí již od poloviny minulého století. Jelikož se ale jedná o problém z třídy NP[1], není aktuálně možné vyvinout algoritmus, který by řešil tento problém v polynomiálním čase. Formálně se tento problém označuje jako *Dynamic Vehicle Routing Problem* (dále jako D-VRP) a jedná se o rozšíření problému obchodního cestujícího (TSP[2], tedy NP-těžký problém [1]). [2] [3]

[1] nondeterministic polynomial time

[2] Dostupné z: `https://en.wikipedia.org/wiki/Travelling_salesman_problem`

Problémem je také skutečnost, že pro každou kurýrní společnost platí jiné požadavky na doručování. U MESSENGERu se například řeší produkt (ten určuje rychlost dodání, v nabídce jsou dodání do druhého dne, ale také do dvou hodin po celé Praze) nebo velikost zásilky a s ní spojený potřebný dopravní prostředek (od jízdního kola až po dodávku). Pravděpodobnost existence obecného algoritmu, který by řešil optimalizaci ve všech kurýrních službách je tedy velmi nízká.

Jedním z mála reálně navržených algoritmů řešící problém D-VRP je algoritmus MORSS[3] vyvinutý profesorem Jimem Orlinem pro MSC[4], tedy pro organizaci Námořnictva USA realizující většinu jejího námořního zásobování. [4]

Algoritmus měl sloužit k řízení pohybu nákladních lodí při mimořádných událostech, kdy by bylo nutné co nejefektivněji plánovat pohyby mnoha lodí mezi mnoha přístavy. Byl také otestován a MSC s ním byla spokojena. Jestli je stále využíván však není známo kvůli armádnímu utajení. [4]

K řešení podobných optimalizačních NP problémů jsou často využívány aukce (jejich mechanismus je používán především v ekonomickém prostředí). Nejčastěji se v IT prostředí využívá mechanismus Vicreyovy aukce (dle [5]), kdy vítěz aukce platí druhou nejvyšší nabídku. Tento typ aukce je také obecně považován za jeden z nejspravedlivějších a možná i z toho důvodu se moc nevyužívá v ekonomickém prostředí. [6]

## Cíl práce

Cílem práce je provést analýzu firemních procesů ve společnosti MESSENGER, a to primárně z pohledu přiřazování objednávek. Následně bude navržena vhodná forma optimalizace tohoto procesu.

Navržená metoda bude otestována na reálných historických datech a bude navrženo její začlenění do existujícího rozhraní. Kroky budou konzultovány s pracovníky společnosti a budou jim předkládány důležité výstupy z jednotlivých částí práce.

[3] MIT Ocean Routing and Scheduling System
[4] Military Sealift Command

## Struktura práce

První kapitola popisuje analýzu reálných řešení a především analýzu interních procesů ve společnosti MESSENGER spojené s procesem přiřazování objednávek. Kapitola obsahuje také nástin architektury používaného informačního systému a popis důležitých technologií v něm použitých.

Druhá kapitola se zaměřuje na návrh optimalizačního modulu a na návrh zakomponování tohoto modulu do existujícího prostředí.

Třetí kapitola popisuje postupy při implementaci navrženého modulu. Jsou zde popsány metody přístupu k datům, práce s geolokačními údaji a také proces zpracování požadavku na doporučení v modulu.

Ve čtvrté kapitole je popsán vývoj modulu spojený s jeho postupným testováním z pohledu přesnosti optimalizace. Na konci kapitoly je vývoj zhodnocen.

V poslední kapitole jsou popsány postupy využité při testování. Jedná se o jednotkové testy navržených tříd a výkonnostní testování.

Kapitola 1

# Analytická část

## 1.1 Reálná řešení

### 1.1.1 Úvod

Jelikož MESSENGER není jediná kurýrní společnost, existuje již mnoho řešení této problematiky, avšak jedná se především o komerční služby. V této kapitole jsou představeny vybrané nástroje umožňující optimalizaci rozdělení zásilek mezi kurýry.

U těchto nástrojů je však většinou nabízena pouze komerční licence a možnosti analýzy jsou tak omezené. Je většinou možné zjistit, co vše daný nástroj zvládá, avšak není již možné zjistit, jaké postupy a technologie k tomu používá.

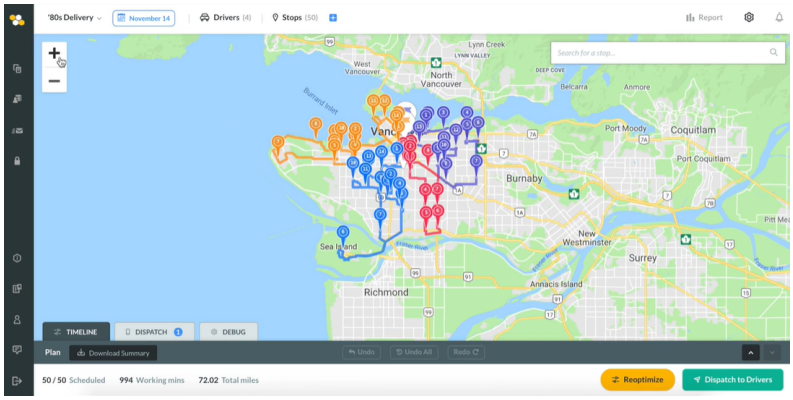
### 1.1.2 Routific[5]

Routific představuje kompletní platformu pro potřeby plánování a kontrolování tras. Na svém webu udávají časovou úsporu oproti ručnímu plánování 95 % a snížení průměrné délky cesty o 40 %. [7]

Nástroj nabízí jak webové rozhraní pro operátora, tak i aplikaci pro řidiče. Webové rozhraní nabízí přehled všech cest a jejich zobrazení na mapě. Umožňuje import z různých formátů souborů a okamžité přiřazování naimportovaných zásilek. Je možné také přeřazovat zásilky mezi řidiči. Mobilní aplikace zobrazuje řidiči trasu a seznam jeho přiřazených zásilek.

Další analýzy funkcionality a především metodik řešení D-VRP problému není možná z důvodu nemožnosti nástroj vyzkoušet bez zakoupení licence.

[5]Dostupné z: `https://routific.com`

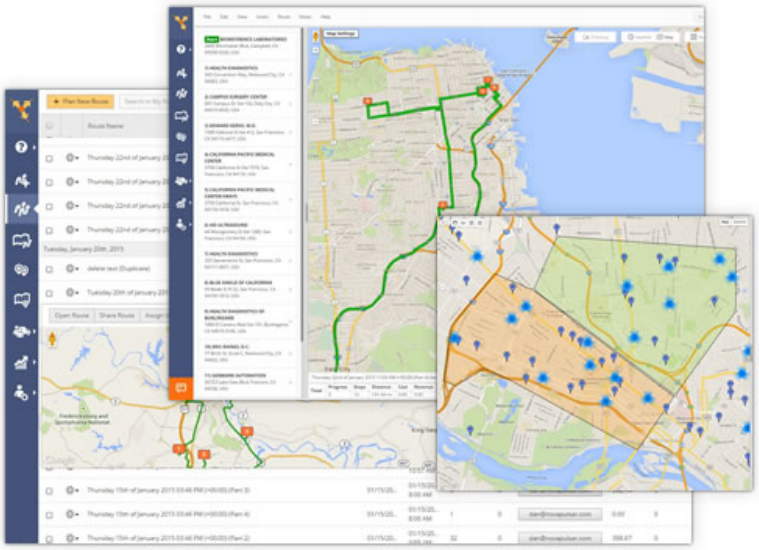



Obrázek 1.1: Nástroj Routific

### 1.1.3 Route4Me

Služba Route4me[6] nabízí podobné funkconality jako Routific, ovšem informace o samotné aplikaci jsou ještě skromnější.

Služba uvádí úspory při použití až ve výši 30 % a chlubí se jednoduchým rozhraním, ke kterému není potřeba žádné školení. [8] Stejně jako u Routific však není možné službu vyzkoušet a analyzovat tak přímo použité metody. Oficiální ukázka rozhraní se nachází na obrázku 1.2.



Obrázek 1.2: Nástroj Route4Me

[6]Dostupné z: `https://www.route4me.com`

## 1.2 Analýza informačního systému ve společnosti MESSENGER

### 1.2.1 Úvod

Akciová společnost MESSENGER byla založena v roce 1991 a za dobu své existence se stabilizovala na poli expresní přepravy. Nabízí vysoce přizpůsobitelné služby pro zákazníky v rámci celé České republiky a části Slovenska. Provozuje 18 poboček po Česku a jednu v Bratislavě.

Přeprava na delší vzdálenosti se podobá stylu běžných kurýrních společností (například PPL, DHL, DPD a další), nicméně ve větších městech je v provozu také expresní přeprava. Nejrychlejším dostupným typem přepravy je produkt EXTREME, u kterého je garantováno doručení do půl hodiny od vyzvednutí (do vzdálenosti 15km). Běžná přeprava po Praze operuje v rámci jednotek hodin.

### 1.2.2 Základní popis informačního systému

Veškeré pracovní pozice ve společnosti podporuje převážně monolitická webová aplikace nazývaná *Kolos*. Ta je napojena na celkem tři databáze, přičemž každá z nich má svůj důvod, proč je použita. Program obstarává jak provoz na osobních počítačích v callcentru, u operátorů a v depu, tak i provoz na mobilních přístrojích v depu a u samotných kurýrů.

Historicky *Kolos* vychází z aplikace napsané v jazyce Delphi[7] ve verzi 6. Jednalo se o desktopovou aplikaci na které společnost fungovala několik let a během provozu nebyla nijak upravována.

Jakmile začala tato původní aplikace společnost spíše omezovat, byl zahájen vývoj *Kolosu*. Ten byl již vyvíjen přímo programátory v rámci MESSENGERu a po několika letech vývoje plně nahradil původní aplikaci.

### 1.2.3 Technologie používané v programu (Backend)

Program využívá velké množství různorodých technologií. V této kapitole jsou tyto technologie postupně rozebrány. V první části se jedná o technologie použité na backendu aplikace (tedy technologie běžící na serveru).

#### Java

I když je poslední dobou spíše na ústupu, tak se Java stále drží jako druhý nejpoužívanější programovací jazyk světa. Na první místo se až letos dostal jazyk C. [9]

Jazyk Java byl vydán firmou Sun Microsystems 23. května 1995 (v roce 2009 byla společnost Sun Microsystems odkoupena společností Oracle[8]). Jedná

[7] Dostupné z: `https://delphi.cz`

[8] Dostupné z: `https://www.oracle.com/`

se o objektově orientovaný jazyk, který narozdíl od jazyka C není kompilovaný, ale interpretovaný. Zdrojové kódy se tedy nepřekládají přímo do strojového kódu, ale do takzvaného „bajtkódu“, který je následně spuštěn pomocí JVM[9].

### Spring Framework

Jedná se o jeden z nejpoužívanějších MVC[10] frameworků pro jazyk Java. [10] Na frameworku Spring běží například VOD[11] služba Netflix[12]. Podporuje všechny potřebné technologie na backendu a zjednodušuje implementaci služeb a závislostí v rámci celého projektu.

### SOAP, WSDL

SOAP[13] je častým protokolem, který se v podnikové sféře využívá ke komunikaci mezi více aplikacemi. Základem jsou zprávy ve formátu XML[14], které se posílají standardně protokolem HTTP (je však možné použít i jiné protokoly).

WSDL[15] je jazykem pro popis webových služeb. V praxi to znamená, že server zveřejní WSDL soubor a klient si s jeho pomocí vygeneruje potřebné třídy, pomocí kterých bude volat dané API.

V aplikaci se využívá především u starších modulů pro komunikaci mezi frontendem a backendem. Novější moduly komunikují spíše pomocí REST.

### REST

REST[16] je aktuálně nejpoužívanější technologií v oblasti webových služeb. [11] Narozdíl od SOAP je REST orientován na zdroje. Každý zdroj je identifikován pomocí URI[17] a operace nad nimi jsou definované pomocí HTTP metod (například GET, PUT, POST).

Oproti SOAP je jednodušší na implementaci a zprávy (požadavky a odpovědi) jsou oproti XML daleko menší zátěží pro přenosové linky díky nižší velikosti.

V aplikaci je REST využíván u všech novějších modulů. Navíc je v provozu také veřejné REST API určené pro zákazníky.

[9] Java Virtual Machine
[10] Model View Controller
[11] Video On Demand
[12] Dostupné z: `www.netflix.com/`
[13] Simple Object Access Protocol
[14] Extensible Markup Language
[15] Web Services Description Language
[16] Representational State Transfer
[17] Universal Resource Identificator

### 1.2.4 Technologie používané v programu (Frontend)

V této kapitole jsou rozebrány nejdůležitější technologie využité na Frontendu aplikace. Jedná se o separátně běžící Java aplikaci, využívá se však i framework AngularJS v jazyce javascript.

#### Play Framework

Jedná se o framework běžící nad JVM. Je postaven na principech technologie REST a umožňuje jednoduše propojovat programovou logiku psanou v jazyce Java (případně je zde i podpora jazyku Scala) a šablony propojující přímo HTML šablony a skripty v jazyce Groovy.

U starších modulů Play obstarává veškerou komunikaci mezi uživatelem a Backendem. U nových modulů přesměrovává požadavky do komponent napsaných v AngularJS.

#### Javascript

JS je jazyk, kterému za posledních několik let prudce vzrostla popularita a podle TIOBE indexu se nachází již na sedmém místě[9]. Vznikl v roce 1995 a standardizován byl v roce 1997 jako ECMASript 1. [12]

Jedná se o skriptovací jazyk interpretovaný přímo v prohlížeči u klienta[13] (technologie Node.js však umožňuje i využití na serveru). Podporují jej všechny používané prohlížeče, a to ve standardizaci ECMAScript 6 (některé i ve standardizaci ECMAScript 7, konkrétně Google Chrome od verze 68 a Opera od verze 55). [12]

JS umožňuje vytvářet interaktivní webové aplikace tak, jak je známe. Pomocí technologie AJAX[18] umožňuje volat služby na serveru a získávat aktuální data i bez nutnosti načíst celou stránku znovu. Je to tedy základ pro interaktivní formuláře (například modul telefonistky) či dynamické aplikace (modul operátora a mapa kurýrů).

#### jQuery

Jedná se o knihovnu zjednodušující práci s čistým JS. První verze byla vydána už v roce 2006, ale stále je udržována a vycházejí nové verze. [14]

Zjednodušuje jak práci s DOM[19], tak i například práci s AJAX požadavky. Pro svou jednoduchost byl základem pro web takový, jaký ho známe dnes. Nyní je už však tato knihovna zastaralá a v nových projektech se nevyužívá.

V informačním systému se nachází především v těch nejstarších modulech (modul telefonistky a operátora), protože byly vyvíjeny v době, kdy pokročilejší frameworky teprve vznikaly. To aktuálně značně znepříjemňuje možnosti rozvoje těchto modulů především z toho důvodu, že se jedná i o soubory

[18] Asynchronous Javascript And XML
[19] Document Object Model

s délkou několik tisíc řádek, ke kterým je dostupná minimální dokumentace. Postupně se však tyto moduly přepisují do modernějších a především udržitelnějších technologií.

#### AngularJS

Je to jeden z několika aktuálně používaných frameworků pro jazyk javascript. Framework byl vyvíjen už od roku 2009 a stojí za ním dva vývojáři Misko Hevery a Adam Abrons. Následně však byl celý projekt přesunut do Googlu, kde je vyvíjen dodnes. [15]

Nejnovější verze, nazvaná Angular 4 byla vydána v roce 2016 a přinesla spoustu změn. Především již není postavený nad javascriptem, ale Typescriptem (standardizace v prohlížečích se jmenuje ECMAScript 6). V informačním systému je však využita první verze frameworku, tedy AngularJS.

AngularJS umožňuje narozdíl od jQuery vyvíjet strukturovanější a udržitelnější aplikace. Framework se sám stará o propojení šablony (většinou v HTML) a javascriptových skriptů. Podobně jako jQuery také zjednodušuje zpracování AJAX požadavků.

### 1.2.5 Technologie používané v programu (databáze a infrastruktura)

Důležité jsou také samotné serverové a databázové technologie, pomocí kterých je udržován provoz vyvíjených aplikací. V této kapitole jsou postupně popsány tři druhy použitých databází a následně také některé důležité technologie běžící na serverech.

#### Microsoft SQL Server

MSSQL je relační databáze od společnosti Microsoft[20] vyvíjená už přes 30 let (první verze byla vydána v roce 1989). Představuje spolehlivé a robustní řešení jako relační databáze pro uchovávání perzistentních dat.

V podnikové infrastruktuře představuje perzistentní úložiště dat. Data jsou chráněna transakcemi a také jsou nejčastěji (aktuálně každý den) vytvářeny zálohy pro případ ztráty dat. V databázi se také nachází velká část podnikové logiky (především počítání cen za doručení), což není už aktuálně příliš vhodné řešení a udržitelnost těchto databázových funkcí je omezená.

#### Elasticsearch

Elasticsearch[21] je distribuovaný Fulltextový vyhledávač nad uloženými daty. Na rozdíl od běžných relačních databází má velmi omezené použití. Nepodporuje transakce, dotazy nad více tabulkami (v terminologii ES nad více indexy)

[20] Dostupné z: `www.microsoft.com/`

[21] Dostupné z: `https://www.elastic.co`

a propojování dat mezi indexy. Umožňuje pouze vkládat sloupcová data a poté nad nimi provádět dotazy či agregace.

Data v ES tedy představují v podstatě pouze jednu velkou tabulku (jeden řádek na každou uloženou zásilku a přibližně 400 sloupců). ES následně umožňuje mezi těmito daty velmi rychle vyhledávat a vytvářet tak rozhraní, která by nad standardním relačním schématem byla velmi pomalá a zbytečně by zatěžovala databázový server.

Data zde uchovaná je vždy možné ztratit a následně „naindexovat“ z databáze. Obsahují také mnoho předpočítaných hodnot, které v databázi uložené nejsou, ale vždy jsou dopočitatelné.

Komunikace s ES probíhá pomocí REST rozhraní, které automaticky tato databáze poskytuje. Pro Javu je také dostupná knihovna, která umožňuje interaktivně vytvářet dotazy přímo pomocí objektů.

### MongoDB

Stejně jako ES se jedná o představitele (tentokrát dokonce toho nejpopulárnějšího dle [16]) představitele NoSQL databáze. Jedná se o takzvanou dokumentovou databázi, protože k datům přistupuje jako k dokumentům (ukládá ve formátu BSON, který je podobný standardnímu formátu JSON[22]). Jedná se o distribuovanou databázi s možností horizontálního škálování.

Narozdíl od MSSQL umožňuje MongoDB také ukládat soubory. Jedno z použití je tedy ukládání všemožných PDF souborů. Druhým použitím je ukládání historie objednávek (ukládají se všechny editace). Stejně jako u ES však ztráta dat nepředstavuje blokující problém.

### Apache Tomcat

Backend aplikace je provozována na aplikačních serverech Apache Tomcat ve verzi 8[23]. Je vyvíjen jako otevřený software a představuje spolehlivé řešení pro i velké aplikace. [17]

Apache Tomcat se může chlubit více než 100 000 instalacemi z oficiálních stránek. Používá se také na spoustě hostingových služeb. [18]

### HAProxy

HAProxy je velmi oblíbeným nástrojem sloužícím k vyvažování zátěže na serverech (anglicky load balancing). Je napsán v jazyce C a i díky tomu se pyšní vysokým výkonem a také vysokou stabilitou. [19]

Umožňuje i například dočasně přesouvat zátěže na jednotlivé servery, čímž umožňuje bezvýpadkově přenasazovat moduly běžící na aplikačních serverech.

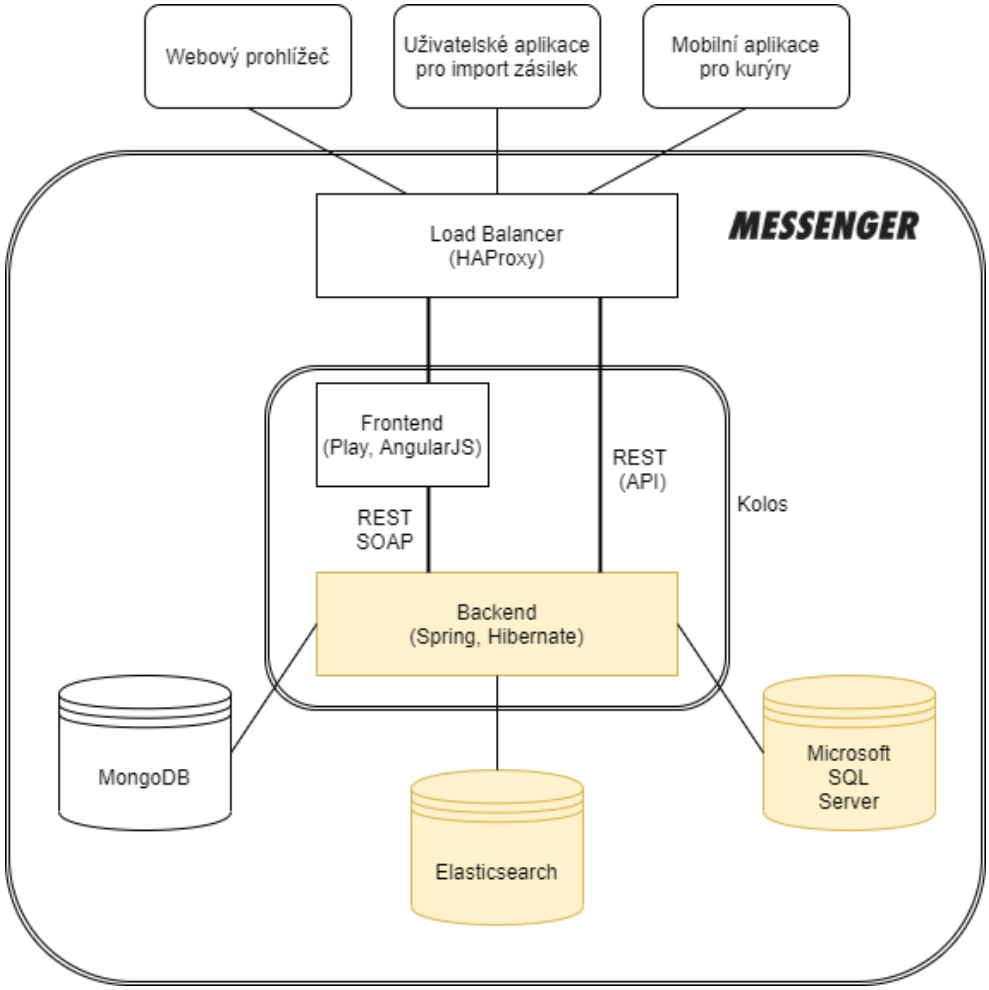
[22] Javascript Object Notation

[23] Dostupné z: `http://tomcat.apache.org/tomcat-8.5-doc/`

## 1.3 Architektura informačního systému

Na obrázku 1.3 je znázorněna zjednodušená architektura celého informačního systému společnosti MESSENGER. Následuje seznam důležitých zjednodušení:

- *Kolosů* (tedy virtuální server, na kterém běží Backend a Frontend) běží souběžně několik. Má to několik důvodů. Nejdůležitější je takzvaný *failover*, tedy schopnost jednoho *Kolosu* obsluhovat veškeré příchozí požadavky (odolnost vůči výpadku). Další výhodou více instancí je také možnost nasazovat nové verze bez výpadkově.
- Elasticsearch i MongoDB jsou v reálu clustery více databázových strojů (z důvodu výkonu i *failover*)
- Stejná architektura je také využita pro testovací prostředí. Na rozdíl od produkčního prostředí se ale jedná většinou o samostatné instance jak *Kolosu*, tak i databázových serverů.
- Existuje i několik menších aplikací, které obstarávají vždy pouze jednu úlohu. Ty v diagramu zakreslené nejsou. Jedná se především o různé monitorovací nástroje a o některé specializované importéry zásilek od náročnějších zákazníků.




Obrázek 1.3: Architektura informačního systému

Implementační část diplomové práce se zaobírá žlutě zvýrazněnou částí (na obrázku 1.3). Bude se tedy jednat o backendovou část za použití databáze MSSQL a vyhledávače Elasticsearch.

## 1.4 Dostupná data vhodná k použití

V datovém úložišti se nachází velké množství historických dat, na kterých je možné testovat prototyp doporučovacího systému. V rámci aktuálních dat jsou také implementovány některé optimalizační prvky (například ukládání aktuálních poloh kurýrů do databáze Elasticsearch (1.2.5) z důvodu rychlosti přístupu).

### 1.4.1 Dostupná data u objednávek

Data objednávek jsou perzistentně uložena v MSSQL (1.2.5) databázi, ale jsou také indexována do vyhledávače Elasticsearch (1.2.5). Během indexace jsou také ukládána některá předzpracovaná data z důvodu jejich rychlejšího načítání.

Pro účely testování prototypu jsou pro nás vhodná data o místech vyzvednutí i doručení, časy vytvoření, přiřazení, vyzvednutí i doručení a také identifikátor kurýra, který zásilku doručil.

### 1.4.2 Dostupná data o kurýrech

Od zavedení nové aplikace na chytré mobilní telefony jsou každou minutu ukládány polohy všech aktuálně pracujících kurýrů. Pro rychlost načítání jsou aktuální data uchovávána ve vyhledávači Elasticsearch (1.2.5). Historická data jsou však uložena v databázi MSSQL (1.2.5) a je tedy velmi pomalé mezi mini vyhledávat (tabulka obsahuje za půl roku cca deset milionů záznamů). Znamená to tedy, že výkonnostní testování bude muset být provedeno na aktuálních datech, abychom získali přehled o výkonnosti algoritmu v produkčním prostředí.

## 1.5 Problémy specifické pro společnost MESSENGER

Jako každá společnost, i MESSENGER má svůj systém přizpůsoben prostředí, ve kterém podniká. Jelikož se omezujeme na doručování expresních zásilek po Praze, budeme uvažovat i zde pouze tuto možnost. Znalosti těchto problémů vyplývají z autorových zkušeností a z několika rozhovorů s pracovníky.

### 1.5.1 Nemožnost jistého určení směru kurýra

Problémem je, že až na výjimky (objednávky s vysokou prioritou apod.) nevíme, ke které zásilce z těch, které již kurýr přiřazené má, zrovna jede. Nemůžeme tedy jistě počítat se směrem od poslední akce a bude nutné tedy směr nějakým způsobem odhadovat.

### 1.5.2 Obecná nepřesnost GPS ve městě

Ve větších městech je GPS obecně méně přesná a v Praze hrají velkou roli i tunely, které způsobují i několikaminutové výpadky polohy.

### 1.5.3 Solidarita v oblasti přiřazování zásilek

Operátoři nejenže přiřazují podle potencionální efektivity doručení (rychlost doručení především), ale také přiřazují mezi kurýry v určitém slova smyslu

„rovnoměrně“ (aby nejrychlejší kurýr nejezdil i třikrát více zásilek za den oproti ostatním) a zohledňují také schopnosti kurýrů (začátečníkům nedávají příliš velké množství zásilek najednou).

Tento vliv je velice těžce přenositelný do algoritmu a je tedy i z tohoto důvodu nutné počítat s nepřesnostmi.

## 1.6 Proces doručení zásilky

Doručování v rámci celé republiky funguje podobně jako u běžných společností. Kurýr den předem ví, co a kam bude doručovat a sám si naplánuje trasu (jelikož jezdí každý den stejnou lokalitu, není v tomto směru žádný problém). V této oblasti aktuálně není moc prostoru pro optimalizaci, a proto jsou v rámci diplomové práce tyto typy zásilky (takzvané OVERy) ignorovány.

Zajímavější systém vyzvedávání a doručování je v rámci městské expresní dopravy (především po Praze a Brně). V těchto místech velká část zásilek vzniká až v průběhu dne a je tedy na operátorech[24], aby tyto zásilky efektivně přiřadili vhodným kurýrům. Zde se nabízí oblast, v níž by bylo možno naimplementovat automatický přiřazovací systém, nebo případně pouze doporučovací systém.

### 1.6.1 Popis životního cyklu objednávky

Proces doručování je zjednodušeně popsán v následujících krocích:

1. Zákazník vytvoří objednávku. To lze provést několika způsoby, může ji importovat přes svůj personalizovaný importér, případně API (tohle jsou dvě nejčastější metody u společností objednávající vyšší množství zásilek).

   Další metodou je objednání přes formulář na webových stránkách a poslední možností je zavolání do call centra, kde objednávku zákazník objedná s aktuálně dostupnou telefonistkou.

2. Objednávka se následně v určitý čas od vyžadovaného vyzvednutí objeví v rozhraní operátora (u produktu STANDARD je garantováno vyzvednutí do půl hodiny, objeví se tedy okamžitě). Operátor vidí všechny parametry objednávky (místa vyzvednutí a doručení, cenu, případnou dobírku nebo speciální požadavky) a začne se rozhodovat komu takovou zásilku přiřadit.

3. Operátor pomocí rozhraní (případně i pomocí vysílačky) tedy určí kurýra a tomu ji přiřadí. Kurýr zásilku co nejdříve vyzvedne a následně má určitou dobu na doručení do cílové lokality (u produktu STANDARD

[24] Operátor je člověk, který v rozhraní přiřazuje zásilky kurýrům

se jedná o 2 hodiny od vyzvednutí, ale jsou i produkty s rychlejším doručením).

### 1.6.2 Rozhraní operátora

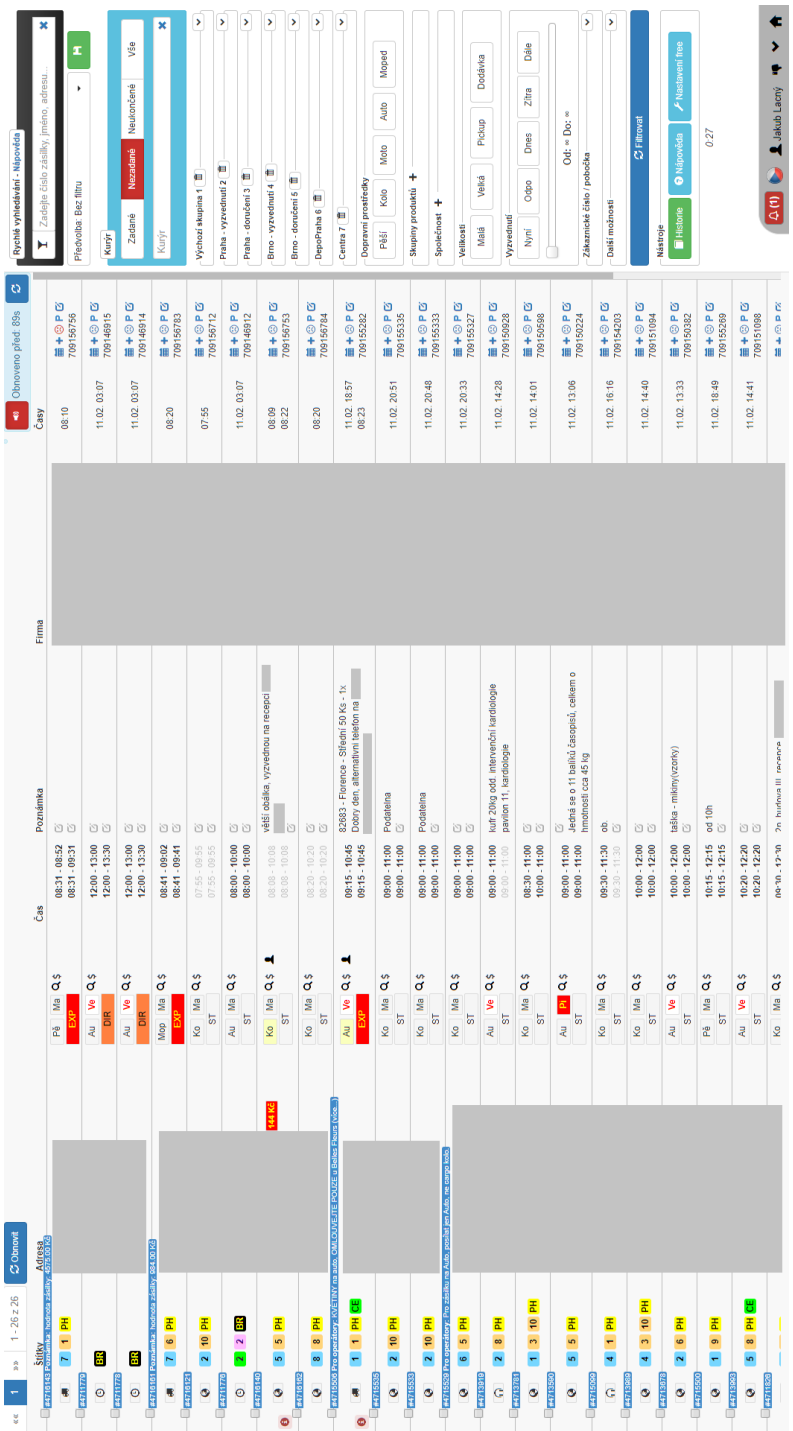
V této kapitole jsou postupně rozebrány dvě aplikace, které operátoři využívají ke své práci. Jedná se o modul „Operátor“, který zobrazuje veškerá potřebná data v tabulkovém rozhraní a o modul „Mapa kurýrů“, ve kterém mají operátoři k dispozici zobrazení všech kurýrů na mapě Prahy.

#### 1.6.2.1 Aplikace Operátor

Na stránce se ve formě velké tabulky zobrazují aktuální zásilky, které je potřeba realizovat. Operátor může velice rychle i pomocí klávesových zkratek přiřazovat a jinak upravovat zásilky (omluvy zákazníkům, přiřazení jinému kurýrovi, změna parametrů zásilky).

Stránka v podstatě nepředstavuje funkcionalitu, kterou by nebylo možné nikde jinde provést, ale sdružuje veškeré potřebné funkce pro daný případ užití.

Je zde také možné nastavit volné přiřazování zásilek (kurýři si je sami rozebírají, takovéto zásilky se označují jako „Free“) a další podobná nastavení, která přímo ovlivňují chování kurýrní aplikace v určitých situacích.



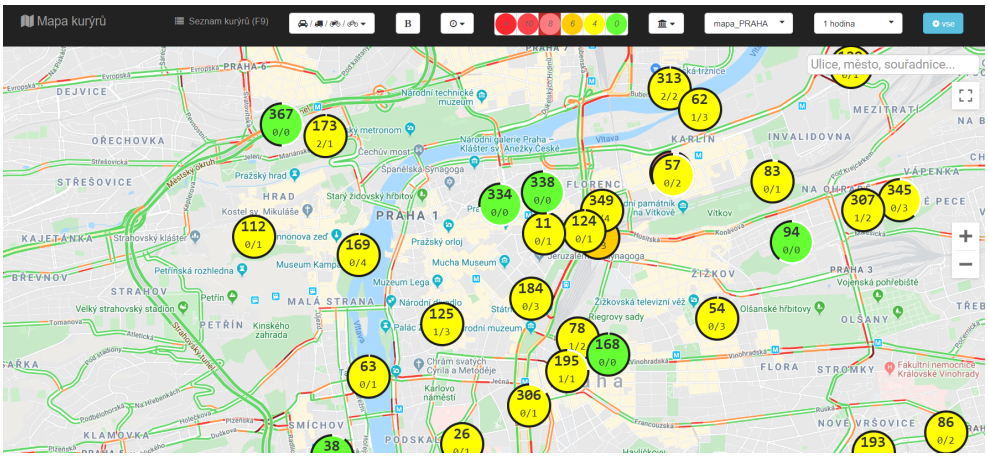
Obrázek 1.4: Rozhraní operátora

#### 1.6.2.2 Mapa s polohami kurýrů

Mapa obsahuje přehled všech kurýrů na mapě. Dříve zobrazovala pouze polohu poslední akce kurýra (vyzvednutí nebo doručení), nyní však aplikace odesílá polohu každou minutu a je tak možné lépe přiřazovat zásilky (sledování aktuální polohy byl hlavním důvodem zavedení nové mobilní aplikace).

Jak již bylo zmíněno v kapitole o specifických problémech (kapitola 1.5), často se stává, že je poloha neaktuální, případně nepřesná. Neaktuálnost polohy je zde prezentována ubývajícím obvodem kolečka znázorňujícího kurýra. S těmito vlivy ovšem operátor musí počítat.

Operátoři mají k dispozici také různé filtrovací možnosti a kurýři jsou také barevně odlišení podle jejich aktuálního vytížení. Mapa kurýrů je zobrazena na obrázku 1.5.




Obrázek 1.5: Mapa s polohami kurýrů

### 1.6.3 Historie a poslední úpravy

Aktuální modul operátora nahradil jeho původní verzi (program napsaný v prostředí Delphi[25]) a v podstatě zkopíroval jeho funkcionalitu. Během posledních let docházelo pouze k menším změnám. Za zmínku stojí například zakomponování rozhraní pro odesílání omluv za zpoždění zásilek.

Za jediné úpravy v pracovním procesu se dá považovat vytvoření mapy s polohami kurýrů. Jedná se o poměrně nový modul, který je v poslední době aktivně vyvíjený. Velkou změnou bylo především zavedení mobilních aplikací sledujících polohu kurýrů. Ta se odesílá každou minutu a mapa je díky nim daleko užitečnější. Nedávno se také přidávaly různé možnosti filtrování a vizualizace stáří polohy.

[25]Dostupné z: `https://delphi.cz`

## 1.7 Analýza možných metod optimalizace

V rámci optimalizace má smysl se omezit pouze na oblast expresní přepravy po Praze. To především z důvodu toho, že zásilky mimo Prahu jsou buď naplánovány den dopředu, vedou z depa (případně se jedná o svoz do depa) a nenastává velký překryv oblastí daných kurýrů, nebo se jedná o zásilky s vysokou prioritou a jejich doručení se řeší individuálně.

Nicméně po Praze přes pracovní dny bývá vysoká hustota kurýrů a je zde tedy vhodné odlišit různé případy, ve kterých je možné automaticky navrhnout přiřazení, protože není ve výkonnostních silách serverů aby neustále porovnávaly všechny cesty a snažily se hledat optimální rozdělení zásilek (variace na problém obchodního cestujícího[26]).

### 1.7.1 Přiřazování na základě polohy

Jedná se o případ, kdy nám do systému právě přibyla nová zásilka a potřebujeme ji vhodně přidělit mezi kurýry. V takovém případě bude nejlepší první vzít v úvahu aktuální polohy kurýrů a pokusit se najít kurýra, který se nachází nejblíže místu vyzvednutí.

Dalším postupem může být například analýza tras kurýrů a odhadnutí, který z kurýrů bude danou oblastí nejdříve projíždět (viz sekce specifické problémy).

### 1.7.2 Přiřazování na základě právě vykonané akce

V aplikaci již funguje nástroj, který kontroluje, jestli na místě, kde nyní kurýr provádí nějakou akci (vyzvednutí či doručení), není dostupná i další akce.

Reálně se jedná například o situaci, kdy kurýr doručuje objednávku, a na stejném místě je objednáno i vyzvednutí. V takovém případě je objednávka přiřazena a kurýr ji neprodleně převezme.

Kurýrovi se po vyzvednutí či doručení také zobrazí výběr volných „free“[27] zásilek v okolí, které si může přiřadit.

[26] Dostupné z: `https://cs.wikipedia.org/wiki/Problém_obchodního_cestujícího`

[27] interní označení zásilek, které si kurýři mohou přiradit sami

### 1.7.3 Analýza historických dat

V rámci historie zásilek je dostupné velké množství dat vhodných pro analýzu. Je možné je předzpracovat a následně při podobných situacích v provozu využít již použité schéma následujících akcí. Bude vhodné také tato data periodicky přepočítávát, protože se nalezené trendy v přiřazení mohou měnit.

### 1.7.4 Přiřazování na základě právě přiřazené zásilky

Idea tohoto způsobu přiřazení je situace, kdy byla právě kurýrovi přiřazena zásilka, a systém detekuje podobnou zásilku (ve smyslu míst vyzvednutí a doručení). V takovém případě je vhodné tuto zásilku doporučit k přiřazení.

Tato forma doporučení již v systému implementována je. Zodpovědnost je v tomto případě přímo na kurýrovi, kterému se po přiřazení zásilky v mobilní aplikaci zobrazí podobné zásilky, které jako podobné vyhodnotil systém. Obdobná situace nastává i během vyzvednutí nebo doručení, kdy se kurýrovi zobrazí nejbližší zásilky. I v tomto případě je v jeho kompetenci, zda je přijme, či nikoliv.

## 1.8 Nielsenova heuristická analýza

Pro účely rychlé heuristické analýzy provedené autorem je využita heuristika sepsaná autorem Jakobem Nielsenem. Jedná se o deset jednoduchých pravidel, podle nichž se vyhodnocuje přehlednost a uživatelská přívětivost. [20]

Běžně heuristickou analýzu provádí tři až pět expertů (podle Nielsena odhalí 5 expertů 75 % problémů a při větším počtu už přicházíme o cenovou optimalitu [21]). V rámci diplomové práce provede heuristickou analýzu autor a jeden další pracovník, který také studuje stejnou fakultu.

### 1.8.1 Body heuristické analýzy

Nielsonova heuristická analýza se skládá z deseti bodů:

1. **Viditelnost stavu systému** – Uživatel je informován o tom co se děje a dostává zpětnou vazbu na své akce.

2. **Shoda mezi systémem a reálným světem** – Aplikace využívá stejnou terminologii a koncepty, které se v daném odvětví využívají v reálném světě.

3. **Uživatelská kontrola a svoboda** – Uživatel by měl být informován o možných důsledcích své plánované akce a měl by mít možnost vrátit poslední akci.

4. **Konzistence a standardy** – Pro stejné situace a akce se vždy využívají stejné pojmy.

5. **Prevence chyb** – Systém by měl předcházet chybám uživatele (například potvrzovacími modálními okny).

6. **Rozpoznávání místo vzpomínání** – V každé části programu by se uživatel měl pohybovat intuitivně na základě předchozích zvyklostí.

7. **Flexibilní a efektivní užití** – Aplikace by měla být vhodná jak pro začátečníka, tak i pro zkušeného uživatele.

8. **Estetický a minimalistický design** – Aplikace by neměla zobrazovat nerelevantní a zbytečné informace.

9. **Pomoc uživatelům rozpoznat, pochopit a vzpamatovat se z chyb** – Chybové hlášky by měly být srozumitelné každému uživateli a nabídnou možnost vyřešení dané chyby.

10. **Nápověda a dokumentace** – Pro aplikaci by měla být dostupná nápověda a dokumentace.

## 1.9 Heuristická analýza modulu operátora

### 1.9.1 Analýza provedená autorem

1. **Viditelnost stavu systému** – Uživatel má přehled v podstatě o všem co potřebuje. Vidí všechny aktuální (relevantní) zásilky, vidí čas poslední aktualizace a ke každé zásilce také další souhrn informací potřebných k rozhodování.

2. **Shoda mezi systémem a reálným světem** – V této oblasti nejsou v aplikaci žádné nekonzistence.

3. **Uživatelská kontrola a svoboda** – Akce provedené s objednávkami většinou nemají možnost je jednoduše vrátit zpět. U většiny akcí uživatel není informován o možných důsledcích.

4. **Konzistence a standardy** – V této oblasti nejsou v aplikaci žádné nekonzistence. Graficky je v rámci celé aplikace využit CSS[28] framework Bootstrap[29].

5. **Prevence chyb** – Stejně jako v předchozích bodech, uživatel je zřídkakdy informován o možných následcích jeho akcí. Chyby technického rázu nastávají pouze v případě poškozených dat.

6. **Rozpoznávání místo vzpomínání** – Prvky UI jsou v rámci programu konzistentní a pro stejné úkony se používají i stejné ikony (např. otevření přehledu dané zásilky, zobrazení historie poloh daného kurýra).

7. **Flexibilní a efektivní užití** – Pro začátečníka aplikace příliš přívětivá není, neobsahuje žádné vysvětlivky a žádnou dokumentaci. Pro zkušené uživatele je aplikace přímo koncipována, obsahuje i několik klávesových zkratek pro zrychlenou navigaci v programu.

8. **Estetický a minimalistický design** – Pro základní prvky UI je v celém programu využit framework Bootstrap. Minimalistický hlavní přehled příliš není, protože obsahuje velké množství informací k jednotlivým zásilkám.

9. **Pomoc uživatelům rozpoznat, pochopit a vzpamatovat se z chyb** – Jak již bylo zmíněno, některé chyby jsou textově popsány a operátor se sám dokáže rozhodnout jak s ní naložit.

10. **Nápověda a dokumentace** – Nápověda se nachází pouze u některých prvků UI a je přítomné modální okno vysvětlující všechny klávesové zkratky, dokumentace přítomna není.

[28] Cascading Style Sheets
[29] Dostupné z: `https://getbootstrap.com`

### 1.9.2 Heuristická analýza provedená jiným pracovníkem

1. **Viditelnost stavu systému** – Celé rozhraní operátora se skládá z jedné obrazovky, na které jsou zobrazeny všechny aktuální zásilky odpovídající filtrům. Seznam zásilek se automaticky obnovuje a přiřazení zásilky kurýrovi je graficky znázorněno v reálném čase i pokud je přiřazení provedeno jiným pracovníkem.

2. **Shoda mezi systémem a reálným světem** – Shoda mezi systémem a reálným světem je problematická. V aplikaci se využívá velké množství výrazů, které ne úplně odpovídají pojmům z reálného světa, ale jsou ustálené napříč společností a není složité se specifické názvosloví naučit.

3. **Uživatelská kontrola a svoboda** – Návrat zpět většinou znamená provedení „opačné“ akce a bezpečný návrat není dostupný.

4. **Konzistence a standardy** – V rámci celého systému jsou prvky a zkratky konzistentní.

5. **Prevence chyb** – Prakticky celá stránka je vyplněna aktivními prvky, které vyvolávají určité akce kliknutím pravým tlačítkem myši, levým tlačítkem myši nebo klávesovou zkratkou. Uživatel může velmi rychle provést spoustu akcí, ale často není o výsledku informován a některé důležité akce může provést i omylem. Prevence chyb dle mého názoru neexistuje.

6. **Rozpoznávání místo vzpomínání** – Zde rozhodně platí, že uživatel je efektivní pouze při znalosti prostředí z dlouhodobého používání. Nový uživatel může být zmatený, různé prvky reagují různě, někdy má prvek nastavenou akci pro kliknutí pravým tlačítkem a jiný ji zase nemá.

7. **Flexibilní a efektivní užití** – Rozhraní může být velmi efektivní pro uživatele, kteří jsou zvyklí používat klávesové zkratky a používají rozhraní denně. Nabízí pokročilé filtrování, psaní expertních dotazů a klávesové zkratky i v kombinaci s pravým tlačítkem myši.

8. **Estetický a minimalistický design** – Design není minimalistický, protože obrazovka obsahuje velké množství informací. Bez rozdělení obrazovky na více stránek není příliš mnoho cest jak zobrazit takové množství informací efektivněji.

9. **Pomoc uživatelům rozpoznat, pochopit a vzpamatovat se z chyb** – Chyba nastává při selhání systému. Technické chyby jsou pro běžné uživatele nic neříkající a řeší je technická podpora.

10. **Nápověda a dokumentace** – Nápověda je přítomna u některých ovládacích prvků. Dále je k dispozici nápověda k vyhledávání a psaní pokro-

čilých filtrů. V dialogovém okně je dostupný seznam všech klávesových zkratek. Dokumentace ani nápověda typu „step-by-step“ neexistuje.

## 1.10 Analýza modulu operátora provedená s pracovníky

Analýza s pracovník proběhla formou rozhovorů. Byly kladeny otázky na kompaktnost a srozumitelnost prostředí aplikace pro operátory (moduly operátor a mapa kurýrů). Otázky byly ovlivněny také provedenými heuristickými analýzami.

Během rozhovorů byly také kladeny otázky na různé možnosti zakomponování doporučovacího systému do reálného pracovního procesu.

### 1.10.1 Vyhodnocení rozhovoru s ředitelem společnosti Radkem Schierreichem

S ředitelem byly probírány záležitosti týkající se diplomové práce nejčastěji, protože má zkušenosti se všemi pozicemi ve společnosti a má přímou zpětnou vazbu na cokoliv týkající se informačního systému.

Z rozhovorů vyplynulo několik dalších metod na určování vhodných kurýrů k zásilkám (například nutnost kontrolovat, aby místo doručení zásilky nebylo úplně mimo oblast, kam se kurýr chystá jet). Byly s ním probrány také možnosti začlenění výsledků doporučování do aktuálního rozhraní a na základě rozhovoru byly navrženy tři metody, které jsou prezentovány v dalších kapitolách.

Co se týče samotného aktuálního rozhraní operátorů, tak během rozhovoru nebyly identifikovány žádné větší problémy. Modul je v provozu už téměř deset let a většinou se aktualizovaly jen drobnosti (naposledy například možnost hromadných omluv) a žádné větší oblasti na zdokonalení nebyly identifikovány.

## 1.11 Problémy v aktuálním rozhraní operátorů

Z heuristických analýz vyplynuly dvě oblasti, které jsou problematické.

První je nedostatečné informování uživatele o možných následcích jeho akcí. Například při odeslání hromadných omluv uživatel pouze vybere důvod omluvy a odešle je. Není však informován o tom, že je tato omluva zaznamenána v historii zásilky a také že tím odeslal SMS zprávu kontaktní osobě.

Druhou oblastí je absence jakéhokoliv návodu či tutoriálu. Nový pracovník musí být zaškolen a trvá poměrně dlouho, než se s programem naučí pracovat na úrovni ostatních, již zběhlých operátorů.

## 1.12 Možná zdokonalení vyplývající z analýzy

Obě oblasti identifikované v předchozí kapitole jsou řešitelné. S ředitelem společnosti byly obě tyto oblasti probrány a v této kapitole jsou popsána možná řešení.

První problém je možné vyřešit úpravou zobrazovaných informací a přidáním modálních oken tam, kde probíhá nějaká neavizovaná akce a informovat tak uživatele o možných následcích jeho jednání. Potencionálním problémem je však obtěžování již zběhlých operátorů, a tedy i zpomalování pracovního procesu, což není vhodné.

Druhý problém je možné v budoucnu vyřešit vytvořením dokumentace či návodu, ve kterém by byl nový pracovník seznámen s problematikou a metodami doručování zásilek. Možností je také určitá forma ”demo” módu, kde by si nový pracovník mohl vyzkoušet práci v rozhraní na fiktivních zásilkách.

Heuristická analýza také pomohla s návrhem vhodného začlenění výsledků optimalizačního modulu. Vyplynulo především několik míst, kde není již vhodné přidávat další informace z důvodu možné nepřehlednosti.

Kapitola 2

# Návrhová část

V této kapitole jsou popsány návrhové aspekty obou oblastí, kterých se diplomová práce dotýká. V první části se jedná o návrh doporučovacího modulu z pohledu architektury a různých metod optimalizace a v druhé části se jedná o návrh začlenění výsledků optimalizace do existujícího rozhraní.

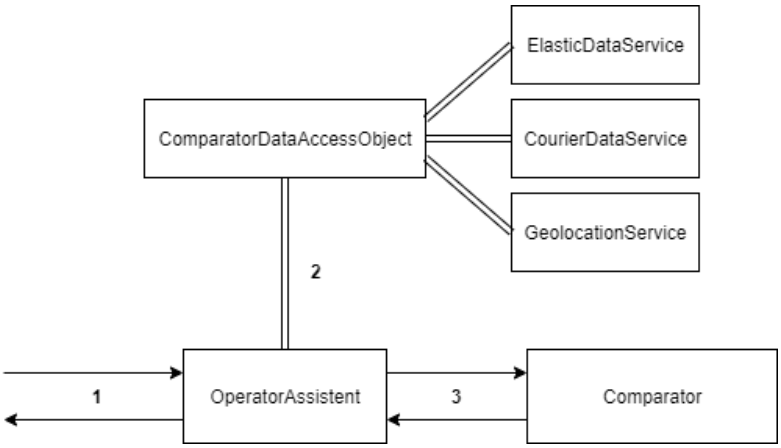
## 2.1 Návrh optimalizačního modulu

Modul bude implementován v rámci aplikace *Kolos* (na backendu). Veřejné rozhraní bude poskytovat pouze metody na doporučení kurýra k dané objednávce. Je definováno několik požadavků na tento modul:

- Nesmí zapisovat žádná data do objednávek. Celý modul pouze čte data z různých zdrojů a pracuje s nimi lokálně.
- Musí doporučit kurýra v rozumném časovém horizontu (maximálně několik sekund). Kdyby každou cestu modul řešil minutu, mohly by být objednávky v době doporučení již přiřazené.
- Všechny rozhraní a třídy jsou umístěny v jednom Java balíčku.
- Doporučení je možné volat jak při vytvoření nové objednávky, tak i například z cronu, který bude prohledávat aktuální zásilky.

## 2.2 Návrh architektury

Před začátkem implementace je potřeba lehce nastínit a promyslet možné rozdělení zodpovědností mezi třídy (respektive komponenty). Návrh je zobrazen na obrázku 2.1 a následně jsou vysvětleny nejdůležitější části návrhu.




Obrázek 2.1: Návrh architektury doporučovacího modulu

1. Modul se bude volat pouze přes rozhraní třídy `OperatorAssistent` (respektive třídy, která bude implementovat tohle rozhraní). Jedná se o třídu, která připraví data pro komparátory a postupně komparátor pro každého kurýra zavolá.

2. Třída `OperatorAssistent` bude také udržovat instanci třídy `ComparatorDataAccessObject`. Tato třída bude sloužit k předání závislostí do komparátorů, protože komparátory nebudou komponentami. V `ComparatorDataAccessObject` budou udržovány závislosti na datové zdroje, které budou využívat jak komparátory, tak i třída `OperatorAssistent`.

3. Pro každého kurýra bude volán komparátor, který určí kurýrovo skóre k dané objednávce. Třída `OperatorAssistent` následně kurýry podle skóre seřadí a vytvoří výsledný objekt, který bude vrácen.

## 2.3 Komparátory

Jádrem doporučování budou takzvané komparátory. Bude se jednat o třídy, které budou počítat skóre pro dvojici kurýr-objednávka. Komparátor bude tedy volán pro každého kurýra a na základě skóre, které určí, budou kurýři seřazeni a doporučeni k přiřazení k dané objednávce.

V této kapitole jsou popsány návrhy principů, které budou následně implementovány v jednotlivých komparátorech.

### 2.3.1 Analýza místa vyzvednutí a doručení

Bude vhodné se pokusit ve vyhledávači Elasticsearch najít, jestli analyzovaný kurýr již nemíří na některý z bodů analyzované objednávky. Pokud ano, může být výsledné skóre kurýra vylepšeno.

### 2.3.2 Analýza množství již přiřazených objednávek

Aktuální, ale i plánované zatížení kurýra hraje velkou roli během rozhodování, kterému z nich objednávku přiřadit. Roli hrají především počet přiřazených objednávek a počet aktuálně vyzvednutých objednávek.

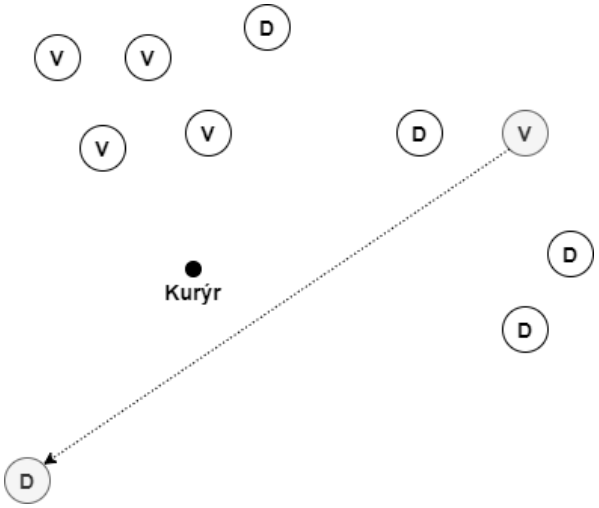
Druhé měřítko, počet aktuálně vyzvednutých objednávek hraje velkou roli především u kurýrů v autech, protože je zde předpoklad, že převážejí objemnější náklad, a kapacita auta je často limitujícím faktorem. Tohle měřítko však v rámci diplomové práce neřešíme, protože uvažujeme pouze standardní zásilky po Praze (více v kapitole 4.1.1).

### 2.3.3 Analýza již přiřazených zásilek

Ve chvíli, kdy přiřazujeme novou objednávku, mají již kurýři (většinou) několik přiřazených zásilek. Pro nově přiřazené zásilky tedy dává smysl, aby všechny body cesty (místa vyzvednutí a doručení) byly v blízkosti již přiřazených cest a jejich bodů.

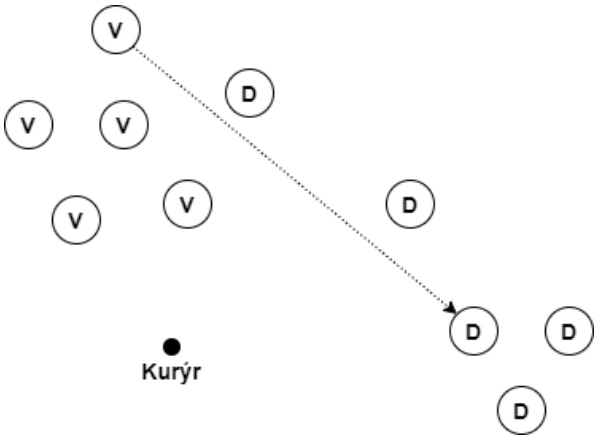
Na obrázku 2.2 je demonstrována pro kurýra nepříznivá situace při přiřazování. Znamená to, že analyzovaná zásilka se nachází mimo oblast, kam se kurýr chystá (na obrázku se nachází místo doručení hodně daleko od jakéhokoliv bodu již přiřazené cesty.

Na obrázku je zobrazen kurýr, již přiřazené cesty (bílé body s popisem „V“ – místo vyzvednutí a „D“ – místo doručení) a analyzovanou cestu (šedé body cesty propojené šipkou).




Obrázek 2.2: Analýza již přiřazených zásilek - nepříznivá situace

Příznivější situace je naznačena na obrázku 2.3. Oba body analyzované objednávky se nacházejí v oblasti, do které se kurýr chystá v rámci již přiřazených zásilek.




Obrázek 2.3: Analýza již přiřazených zásilek - příznivá situace

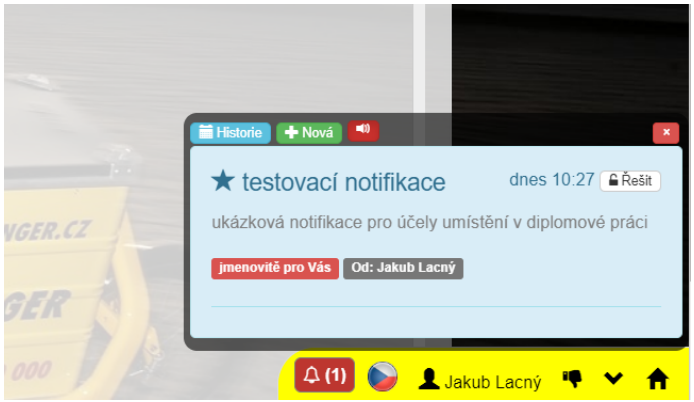
Nutno zmínit, že situace, která je demonstrována na obrázcích 2.2 a 2.3, kdy místa vyzvednutí a doručení tvoří jasně viditelné oblasti, je založena na reálných poznatcích. Je tedy předpoklad, že analýza tohoto typu může způsobit výrazné zlepšení výsledného algoritmu.

## 2.4 Návrh zakomponování doporučovacího modulu do rozhraní operátora

Výstupy z analýzy uživatelského rozhraní modulu operátora jsou využity primárně k návrhu zakomponování doporučovacího modulu. V této kapitole jsou popsány čtyři autorem navržené metody zakomponování.

### 2.4.1 Zobrazování doporučení pomocí notifikací

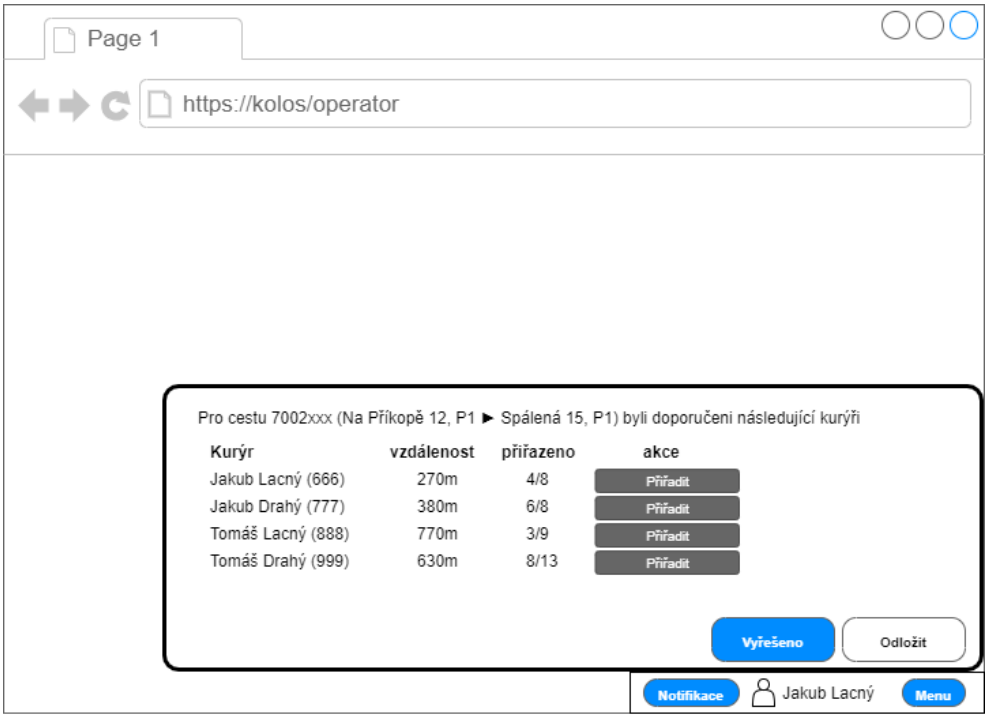
V informačním systému je implementován a využíván nástroj pro zobrazování notifikací. Prakticky se využívají na všechno možné (chybová hlášení, upozorňování na nečekané změny, požadavky na schválení určitých akcí) a pracovníci jsou na práci s nimi zvyklí. Ukázka reálné notifikace je demonstrována na obrázku 2.4.




Obrázek 2.4: Ukázka notifikace v informačním systému

Při vytvoření nové objednávky by v takovéto implementaci byla po zjištění vhodných kurýrů vytvořena notifikace směřovaná operátorům (buď všem, nebo třeba jenom některým) a operátor by měl možnost vybrat některého z doporučených kurýrů a přiřadit mu danou zásilku.

Podoba takové notifikace je naznačena na obrázku 2.5.




Obrázek 2.5: Zobrazování doporučení pomocí notifikací

Velkým omezením tohoto řešení je však omezená možnost implementovat výběr kurýrů dynamicky. Texty notifikací jsou uloženy v databázi (v mongoDB, viz 1.2.5) a není tak možné ani vhodné často aktuálizovat obsah notifikace. Jediná rozumná možnost by byla vždy při změně doporučovaných kurýrů notifikaci smazat a poslat nově vygenerovanou, hrozilo by ovšem přílišné zatížení notifikací a pro operátory by mohly být otravné.

#### 2.4.1.1 Vyhodnocení autorem

Výhody

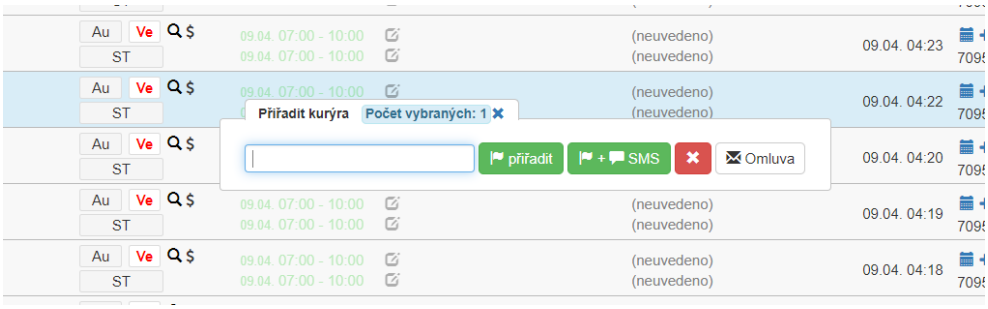
- Využití již hotového nástroje
- Uživatelé notifikace znají
- Jednoduchost implementace

Nevýhody

- Omezená možnost interaktivity
- Velmi omezená možnost budoucího rozvoje rozhraní
- Notifikace se perzistentně ukládají do databáze
- Notifikace jako takové nemají sloužit k tomuto účelu

### 2.4.2 Zobrazení zakomponované do modulu operátora

Další možností je integrace doporučení přímo do modulu operátora. V rozhraní se již nachází místo vhodné pro umístění těchto doporučení. Po kliknutí pravým tlačítkem na zásilku (případně i po označení více zásilek) se objeví malé modální okno obsahující formulářový vstup sloužící k zadání kurýra a několik možných akcí. Aktuální podoba tohoto modálního okna je ukázána na obrázku 2.6.

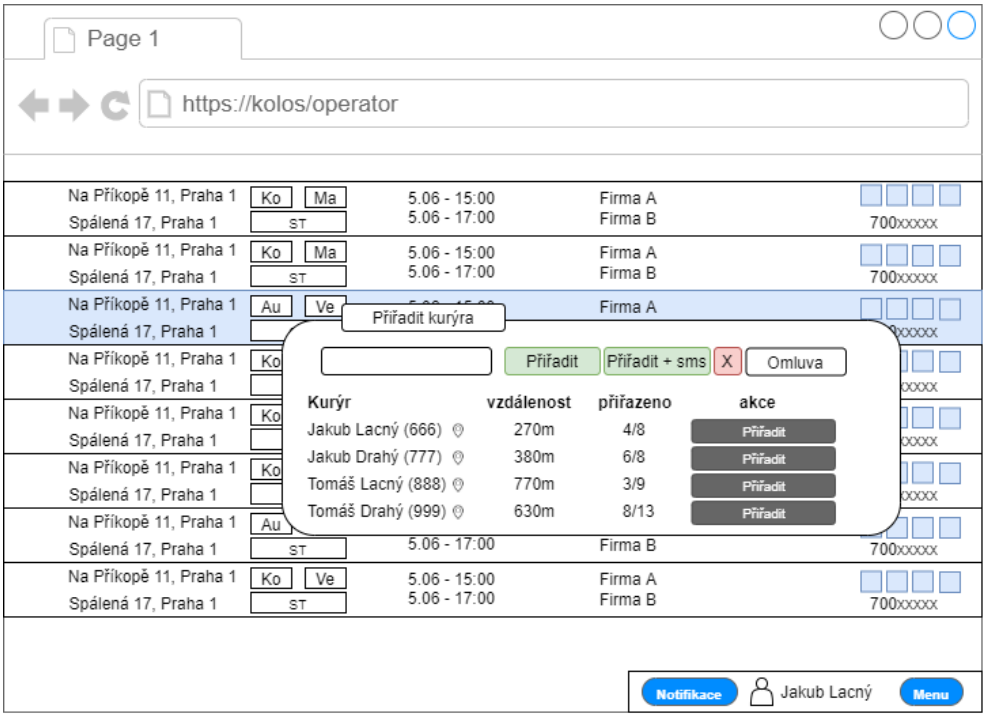



Obrázek 2.6: Ukázka modálu pro přiřazení kurýra

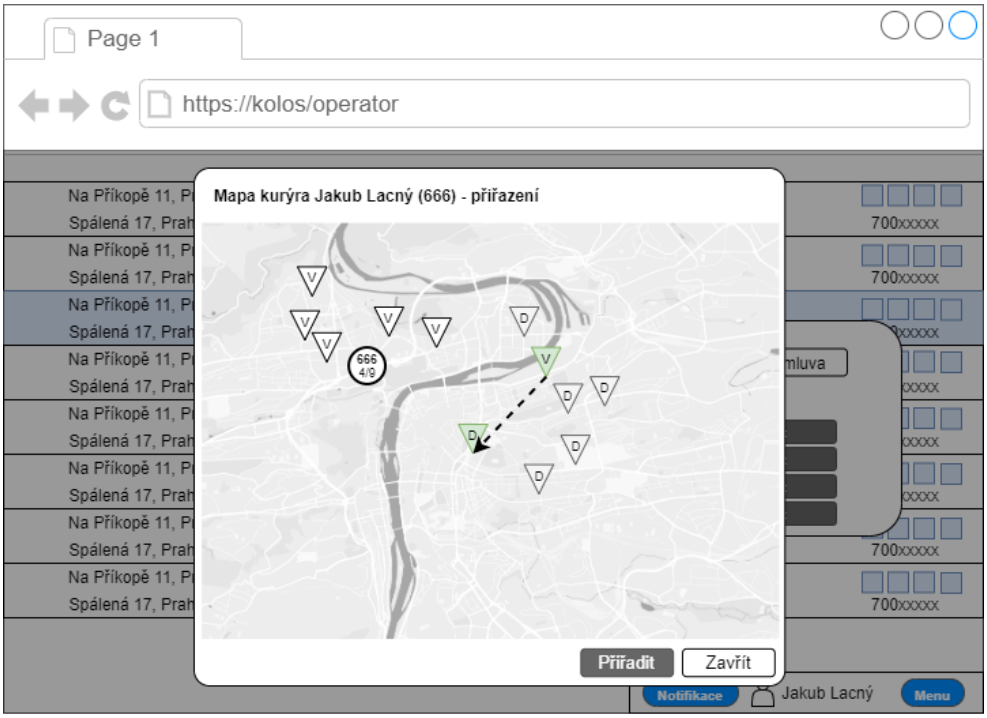
Do tohoto modálního okna by bylo možné umístit stejné rozhraní pro výběr kurýra jako do notifikace. Umístění je naznačeno na obrázku 2.7.

Výhodou oproti umístění v notifikaci je možnost dalšího rozšíření rozhraní po kliknutí na tlačítko umístěné pod rozhraním. Může se například zobrazit mapa, případně podrobnější a interaktivnější nabídka kurýrů.

Při kliknutí na piktogram polohy vedle jména navrhovaného kurýra je možnost zobrazit modální okno zobrazující polohu kurýra, jeho přiřazené objednávky a také analyzovanou objednávku. Tohle modální okno je navrženo na obrázku 2.8.




Obrázek 2.7: Zobrazování zakomponované do modulu operátora






Obrázek 2.8: Zobrazování zakomponované do modulu operátora – modální okno

Kruh s číslem označuje polohu kurýra (číslo je identifikátorem kurýra) a trojúhelníky s „V“ a „D“ označují místa vyzvednutí a doručení již přiřazených objednávek. Zelené trojúhelníky s „V“ a „D“ propojené šipkou označují analyzovanou objednávku.

#### 2.4.2.1 Vyhodnocení autorem

Výhody

- Zakomponování do opět již existujícího rozhraní
- Umístěno tam, kde to dává smysl
- Možnost větší interakce než v notifikaci

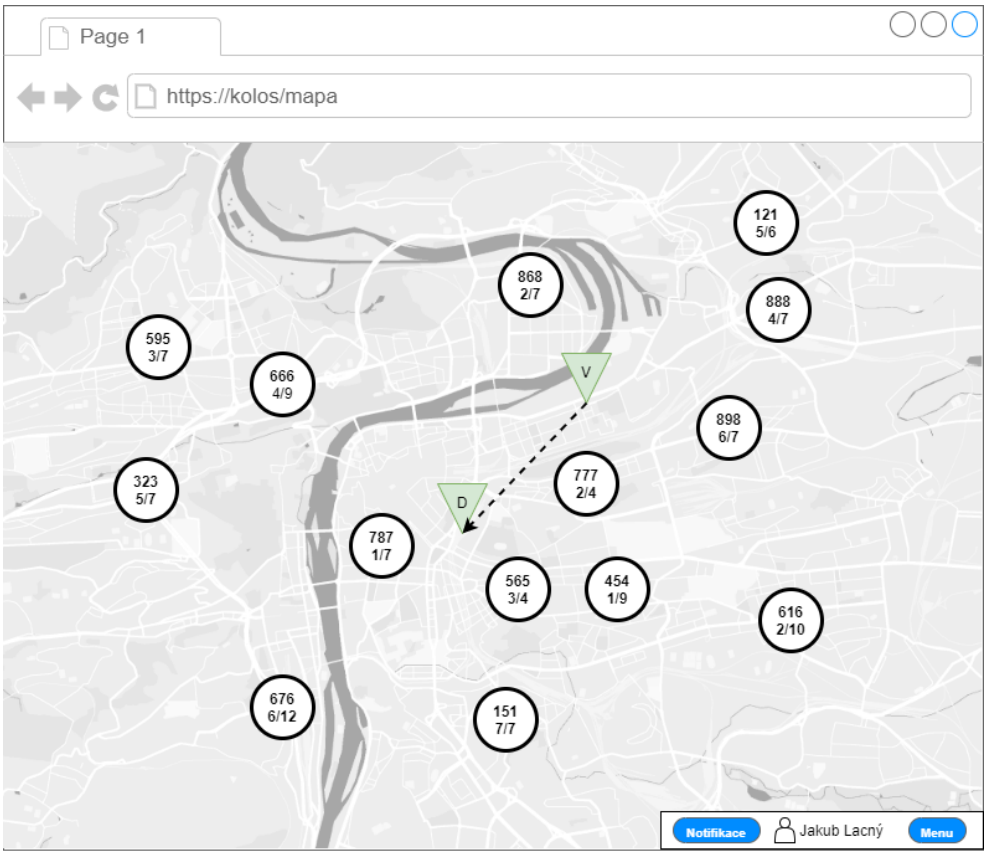
Nevýhody

- Opět omezené možnosti interakce přímo na základním rozhraní
- Doporučení by bylo nutné buď počítat dopředu, aby se zobrazily okamžitě, nebo by se na ně muselo čekat po kliknutí pravým tlačítkem
- Složitost implementace, protože modul operátora je postaven na již zastaralých technologiích

### 2.4.3 Zobrazení zakomponované do mapy kurýrů

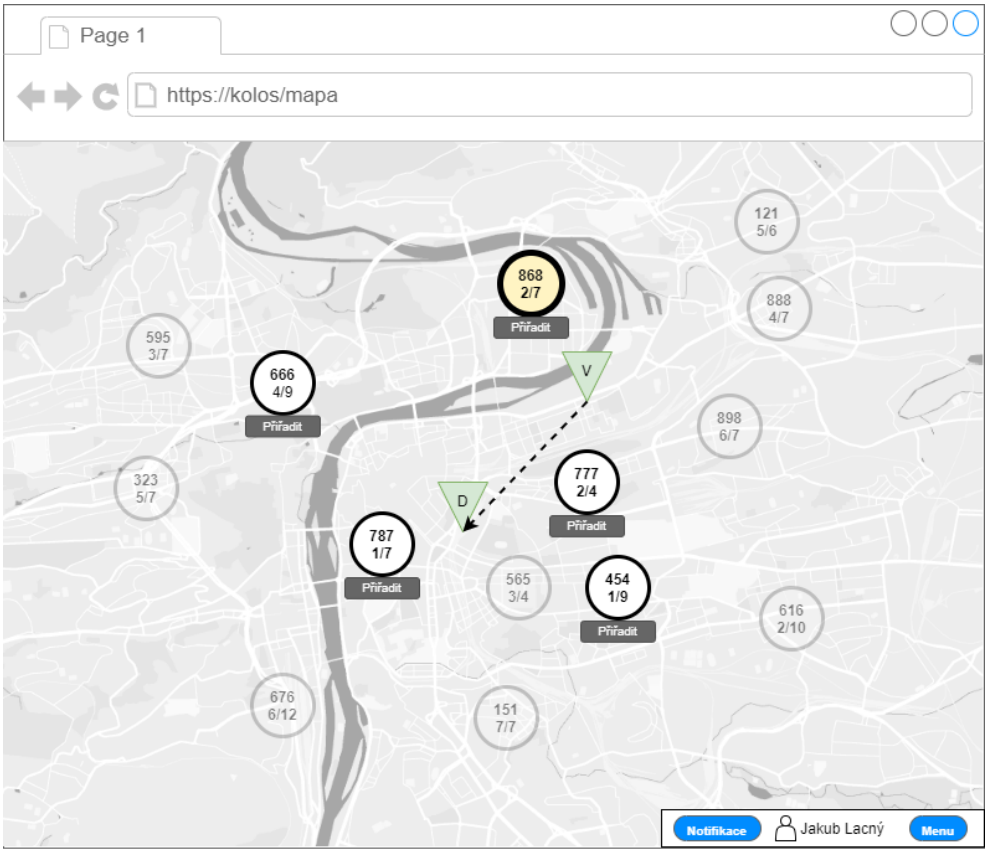
Stejně jako modul operátora se i modul mapy kurýrů jakožto využívaný nástroj hodí k integraci doporučovacího systému. Nyní se na mapě zobrazují pouze polohy kurýrů s několika podstatnými informacemi (stáří nejnovější polohy, počet přiřazených zásilek a podobně. Aktuální podoba je zobrazena na obrázku 1.5.

Na mapě by se mohly zobrazovat nové relevantní objednávky. Objednávka bude znázorněna trojúhelníky na místě vyzvednutí a doručení. Už takovéto zobrazení může dát operátorovi tip na vhodnost přiřazení pod určité kurýry. Objednávky by se mohly zobrazovat například při vytvoření, případně by se mohly objevit i starší objednávky, u kterých se blíží časové okno vyzvednutí, ale ještě nejsou přiřazené. Tento typ zobrazení je znázorněn na obrázku 2.9.




Obrázek 2.9: Integrace do rozhraní mapy kurýrů

Po kliknutí na místo vyzvednutí nebo doručení (znázorněné trojúhelníkem) se zvýrazní vhodní kurýři v okolí. Vhodné je také odlišit nejlepší návrhy (nejvhodnější kurýry) například jinou barvou. Takovýto detail dané objednávky je zobrazen na obrázku 2.10.




Obrázek 2.10: Integrace do rozhraní mapy kurýrů – zvýraznění

#### 2.4.3.1 Vyhodnocení autorem

Výhody

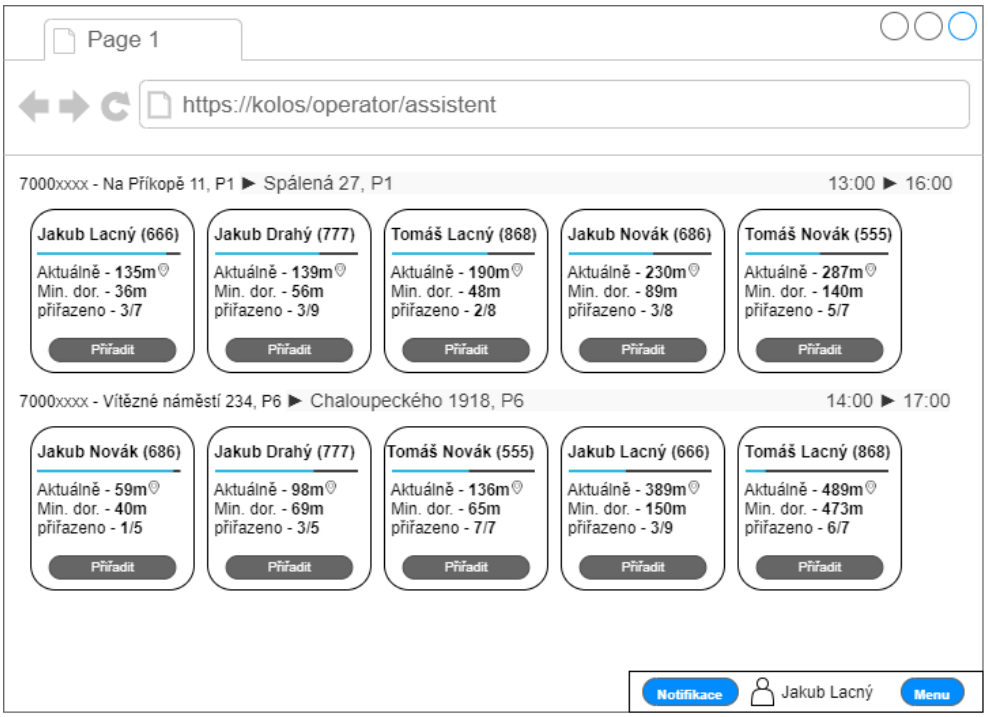
- Opět se jedná o zakomponování do existujícího a používaného rozhraní
- Operátor okamžitě vidí, jaká je situace a kteří kurýři jsou nejblíž
- Velké možnosti interakce (a případného rozšiřování)

Nevýhody

- Možné snížení přehlednosti mapy kurýrů
- Možná vysoká výkonová zátěž pro prohlížeč (už nyní je mapa docela náročným modulem)

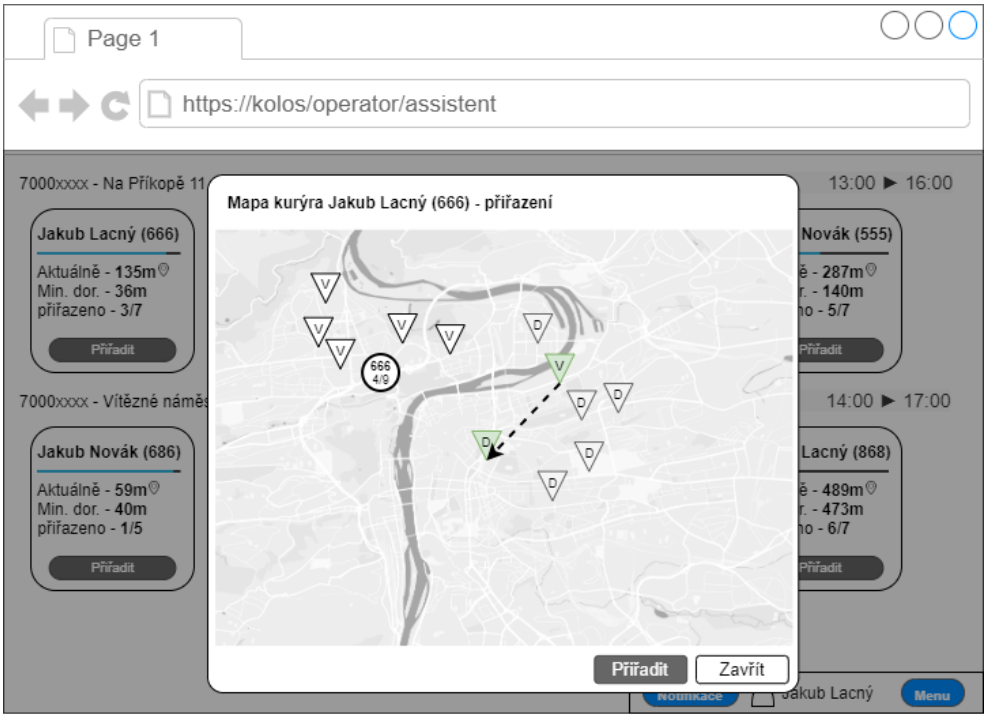
### 2.4.4 Separátní rozhraní na zobrazování doporučení

Jako poslední možnost se nabízí vytvořit úplně nové, samostatné zobrazení pro zobrazování doporučených kurýrů k objednávkám. Tohle zobrazení by nabízelo největší prostor na případný rozvoj a mohlo by také spolupracovat s ostatními moduly. Návrh tohoto rozhraní je zobrazen na obrázku 2.11.




Obrázek 2.11: Zobrazování na separátní stránce

Stejně jako u zobrazení v modulu operátora, i zde by mohla být například možnost zobrazit mapu s polohou kurýra, jeho přiřazenými objednávkami a analyzovanou cestou. Tohle modální okno je zobrazeno na obrázku 2.12.




Obrázek 2.12: Zobrazování na separátní stránce – modální okno

Takovéto separátní rozhraní by umožňovalo také spolupráci s ostatními moduly. Komunikace by probíhala na pozadí reagovala by na akce provedené v modulu operátora nebo na mapě kurýrů. Možné by byly například následující interakce:

- Pokud operátor označí v modulu operátora nějakou objednávku (či více objednávek), zobrazily by se doporučení pro tuto objednávku (nebo pro více objednávek v případě označení více než jedné).
- Při označení kurýra na mapě kurýrů by se mohly zobrazit doporučené objednávky pro tohoto kurýra (inverzní metoda doporučování).
- Při automatickém přidání nové objednávky do zobrazení pro doporučování by se se mohla v modulu operátora zvýraznit daná objednávka.

#### 2.4.4.1 Vyhodnocení autorem

Výhody

- Největší možný prostor pro vývoj v budoucnu
- Možnost spolupráce s ostatními moduly
- V základu nijak nezasahuje do funkcionality ostatních modulů
- Možná vysoká míra interaktivity

Nevýhody

- Nutnost dalšího monitoru (nebo například tabletu) pro operátory na zobrazení

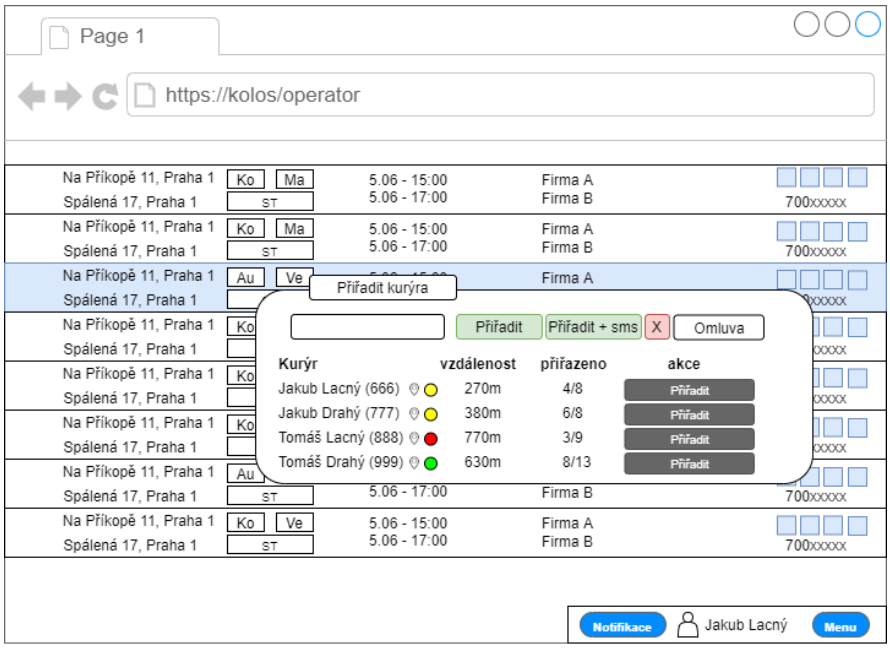
## 2.5 Vyhodnocení návrhů integrace s pracovníky

Návrhy integrace byly zkonzultovány s pracovníky společnosti a vzniklo ještě několik připomínek.

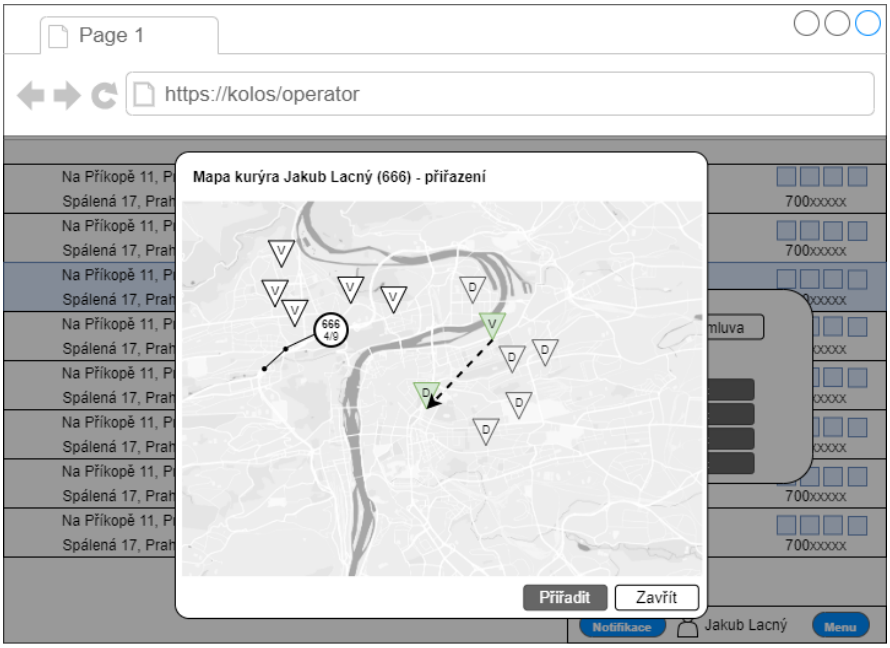
- Jako obecně nejlepší řešení je považováno integrování do rozhraní operátora (do nabídky vyskakující po stisku pravého tlačítka). Bylo tak učiněno především proto, že se v takovém případě jedná o informaci, kterou si zobrazí operátor tehdy, když ji potřebuje, a když tyto informace nepotřebuje, tak jej „neotravují".
- Je vhodné zobrazit u kurýrů i barevné označení indikující jejich vytíženost. Jednalo by se o stejné barevné schéma jako na aktuální mapě kurýrů (lze vidět na obrázku 1.5).
- Do modálního okna zobrazujícího zásilky jednoho kurýra společně s jeho aktuální polohou a analyzovanou objednávkou je vhodné přidat i indikaci směru pohybu kurýra, tedy například propojené jeho poslední tři polohy.
- Integrace do mapy kurýrů by podle ředitele společnosti způsobila nárazově silné znepřehlednění mapy.
- Vytvoření nového rozhraní by nebylo vhodné z důvodu nutnosti kontrolovat další obrazovku.
- Do notifikací sice původně bylo v plánu takováto doporučení zakomponovat, situace se však změnila a touto cestou není vhodné se vydávat. Hlavním důvodem je možné zahlcení notifikací a „otravování" operátorů

### 2.5.1 Úpravy návrhů na základě připomínek

Na obrázcích 2.13 a 2.14 se nacházejí upravené návrhy na základě připomínek pracovníků. Do rozhraní výběru doporučených kurýrů v modulu operátora byla přidána barevná indikace jejich zatížení (barvy jsou stejné jako v modulu mapy kurýrů kvůli konzistenci). Do modálního okna s mapou byla přidána indikace směru kurýra.




Obrázek 2.13: Úprava rozhraní na základě konzultace




Obrázek 2.14: Přidání směru kurýra na základě konzultace

Kapitola 3

# Implementace prototypu optimalizátoru

## 3.1 Struktura modulu

Modul bude umístěn mezi ostatními logicky oddělenými částmi aplikace. Z ostatních částí programu se bude bude volat pouze přes třídu definovanou rozhraním `OperatorAssistent`, která obsahuje metodu na doporučení kurýra pro předanou cestu a metodu na testování, která počítá s předáním již ukončené cesty.

## 3.2 Veřejné rozhraní modulu

Veřejné rozhraní modulu definuje pouze dvě potřebné metody pro komunikaci. V reálném využití v rámci modulu operátora bude využita metoda `getBestCourierForRoute`. Jediným parametrem je identifikátor cesty.

Druhou metodou je metoda `getBestCourierForRouteInAssignmentTime`. Jako parametr zde předáváme i vyžadovaný komparátor pro účely jednodušího testování (běžná metoda bude mít nastavený komparátor v závislosti na výsledku testování).

Obě metody vrací objekt `OperatorAssistentResultDTO`, který obsahuje informace o nejvhodnějších kurýrech včetně jejich dosaženého skóre.

Ukázka kódu 3.1: Veřejné rozhraní modulu

```
/**
 * Analyzes given route and chooses best courier to assign to
 * it (in present time).
 *
 * @param routeImmutableId Id of route to process
 * @return Best courier to assign
 */
public OperatorAssistentResultDTO getBestCourierForRoute (
        Integer routeImmutableId
);

/**
 * Analyzes given route and chooses best courier to assign to
 * it (in given time).
 * Used for testing
 *
 * @param routeImmutableId Id of route to process
 * @return Best courier to assign
 */
public OperatorAssistentResultDTO
    getBestCourierForRouteInAssignmentTime (
        Integer routeImmutableId,
        OperatorAssistentComparator comparator
);
```

## 3.3 Přístup k datům

Třídy pro práci s databázemi jsou již implementovány v aplikaci. Pro přehlednost však autor zvolil vytvoření nových komponent určených pouze pro nový modul, které volají již existující služby a zpracovávají data do modulem požadovaného formátu. Následuje popis jednotlivých datových zdrojů.

- **CourierDataService** – Poskytuje metodu pro prosté získání dat o kurýrovi na základě předaného identifikátoru a metodu `getSuitableCouriersForRoute`, která získá v databázi vhodné kurýry pro zpracování k analyzované cestě (v rámci diplomové práce je tato metoda zjednodušena na prosté získání aktivních pražských kurýrů, reálně bude vybírat aktivní kurýry v lokalitě realizace objednávky a budou se aplikovat i možná omezení, které do diplomové práce neuvažujeme, třeba potřebný dopravní prostředek).
- **ElasticDataService** – V databázi Elasticsearch se nachází předzpracovaná data ke všem objednávkám. Datový zdroj tedy vrací objekty `VyhledavaniCesta` na základě identifikátoru.

- **GeoLocationService** – Obsahuje metody na získání lokačních údajů o kurýrech (jak aktuální data, tak i informace o poloze k předanému datu). Poskytuje také metody pro práci s geokódovacími službami. Samotná volání těchto služeb jsou již v programu implementována, tedy stejně jako u ostatních datových zdrojů se převolávají tyto služby. Využívají se geokódovací služby Google[30], Seznam[31] a OpenStreetMap (služba Nominatim[32]).

### 3.3.1 Příklad práce s vyhledávačem Elasticsearch

Pro práci s Elasticsearch je pro jazyk Java dostupná knihovna, která umožňuje vytvářet dotazy do tohoto vyhledávače pomocí tříd. [22] V reálu má tedy každý druh filtru nebo třeba agregace svou třídu, pomocí které definujeme požadované vlastnosti dotazu a o zpracování se již postará knihovna.[23]

Příklad dotazu do tohoto vyhledávače je prezentován na příkladu, který zjistí počet objednávek, které daný kurýr v daný den již vyzvednul. Třída, ve které se tento dotaz realizuje, musí dědit od objektu `AbstractElasticSearch<>`.

Ukázka kódu 3.2: Ukázka dotazu do Elasticsearch

```
2     List<FilterBuilder> filterBuilders =
          new LinkedList<FilterBuilder>();
4             filterBuilders.add(FilterBuilders.andFilter(
                  FilterBuilders.termFilter( "kuryrId", courier.id ),
6                 FilterBuilders.rangeFilter( "vyzvednuto" )
                  .from(beginningOfDay)
8                 .to(givenDate)
              ));
10
          FulltextSearchResult<VyhledavaniCesta> results =
12            super.search( null,
                  FilterBuilderHelper.joinFilters( filterBuilders ),
14                null, null, null, 0, 10
          );
```

## 3.4 Práce s geolokačními údaji

Třída implementující rozhraní `GeoLocationService` také poskytuje metody sloužící k práci s již získanými geolokačními údaji. Obsahuje metodu sloužící k výpočtu vzdálenosti mezi dvěma body v metrech (metoda výpočtu je zjednodušená tím, že nebere v potaz nadmořskou výšku, tyto údaje totiž u poloh nejsou uložena).

[30] Dostupné z: `https://developers.google.com/maps/documentation/geocoding`
[31] Dostupné z: `https://api.mapy.cz/view?page=geocoding`
[32] Dostupné z: `https://wiki.openstreetmap.org/wiki/Nominatim`

## 3.5 Implementace komparátorů

Třídy implementující rozhraní `OperatorAssistentComparator` představují hlavní komponentu celého modulu a během testování budou vyvíjeny stále lepší a komplexnější komparátory.

Úkolem komparátoru je pro dvojici `<objednávka, kurýr>` vypočítat skóre, podle kterého je následně vybrán vhodný kurýr pro danou cestu. U skóre platí čím vyšší, tím méně vhodný daný kurýr je.

Různé komparátory mohou pro zpracování vyžadovat různá data. Mohou zhodnotit například již přiřazené zásilky pod daným kurýrem, více historických poloh kurýra a podobně. Jednotlivé komparátory budou popsány v kapitole o vývoji a testování komparátorů, protože budou vyvíjeny na základě výsledků testování předchozích verzí.

## 3.6 Zpracování požadavku

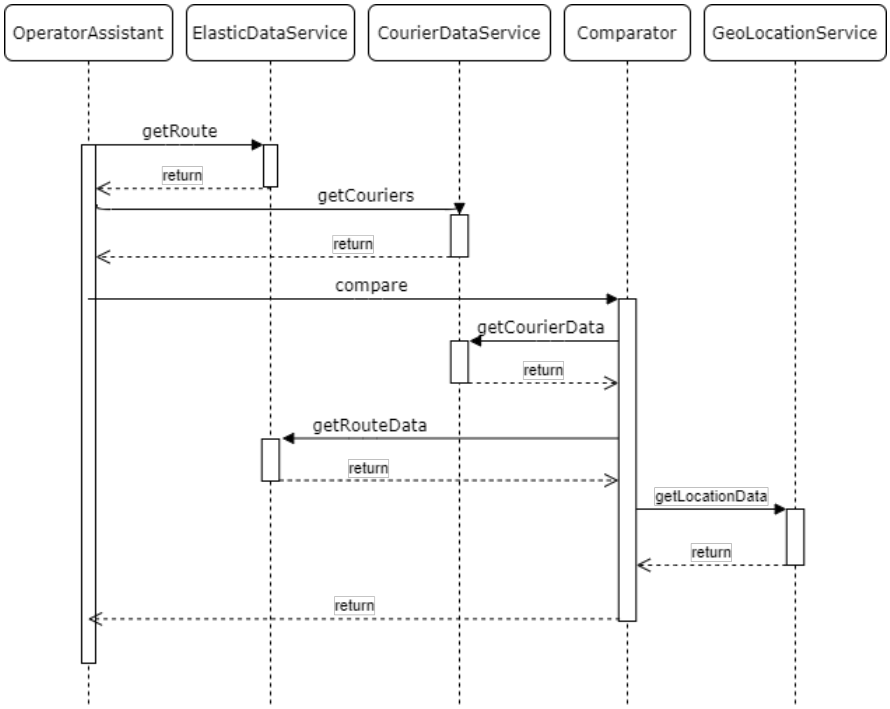
Jak již bylo zmíněno, volání proběhne přes veřejné rozhraní modulu. Následně proběhne získání dat k dané cestě z databáze Elasticsearch a získání seznamu vhodných kurýrů pro následné porovnání pomocí komparátoru (v testovací metodě se komparátor předává jako parametr, na rozdíl od produkční metody, kde se počítá s pevně nastaveným vhodným komparátorem).

Následně se porovná každý kurýr s danou cestou a ke každému je tak vypočteno skóre, podle kterého se určí nejvhodnější kandidát na přiřazení. Celý tento proces je prezentován na sekvenčním diagramu 3.1.

## 3.7 Předávání závislostí v rámci modulu

K přístupu k datovým zdrojům jsou ve frameworku Spring využívány komponenty. Jedná se o třídy s anotacemi `@Component`, které následně můžeme získat v ostatních komponentách pomocí anotace `@Autowired`. Anglicky se této metodě říká Autowiring, vložení závislosti do komponenty se pak říká Injecting (Obecně je tento proces také známý pod pojmem Dependency Injection, zkráceně DI). [24]

Dependenci Injection je využito v rámci celého modulu na propojení tříd, které zpracovávají požadavek s třídami, které zprostředkovávají komunikaci s datovými zdroji. Na ukázce 3.3 je předvedeno standardní využití DI ve frameworku Spring.




Obrázek 3.1: Sekvenční diagram zpracování požadavku na doporučení kurýra

Ukázka kódu 3.3: Ukázka použití Dependency Injection

```
1
// Kod umisteny ve tride, kterou "injectujeme"
3
@Component
5 public class ElasticDataServiceImpl
      extends AbstractElasticSearch<VyhledavaniCesta>
7     implements ElasticDataService {

9     ...

11 // Kód na zacatku tridy, ve které vyzadujeme závislost tridy
   // implementujici rozhrani ElasticDataService
13
   @Component
15 public class OperatorAssistentImpl
      implements OperatorAssistent {
17
      @Autowired
19    private ElasticDataService elasticDataService;

21    ...
```

Odlišná metoda DI je využita v komparátorech. U komparátorů není vhodné, aby se jednalo o komponenty, ale o klasické třídy které si můžeme vytvořit

a máme tak možnost mít více instancí komparátorů jednoho typu najednou (to je u standardního DI také možné, ale náročné na paměť, protože bychom si museli držet vysoký počet instancí komparátoru).

Pro přístup k datům z komparátorů je využitý princip lehce založen na návrhovém vzoru Mediator.

Mediator patří mezi behaviorální návrhové vzory. Zprostředkovává tedy komunikaci mezi více objekty v programu. Nejbližší české označení je „prostředník“. Pro vysvětlení jeho funkcionality se nejčastěji využívá analogie v komunikaci mezi letadly. Ty spolu nekomunikují přímo, ale vždy komunikují přes řídící věž na letišti. Řídící věž zde představuje prostředníka mezi různými letadly. [25]

V modulu je tento princip využit tak, že máme vytvořenou komponentu `ComparatorDataAccessObject`, která udržuje závislosti na komponenty zprostředkovávající přístup k datům (Ukázka této třídy se nachází v ukázce kódu 3.4). Tato komponenta se předává komparátorům při volání porovnávací funkce a ty přes ni tak mohou komunikovat s datovými zdroji, aniž by sami byli komponentami.

Velice podobná mechanika je již v programu využita u vytváření akcí nad objednávkami. Jednotlivé akce také nejsou komponentami a komunikují s datovými zdroji a ostatními komponentami přes podobného prostředníka.

Ukázka kódu 3.4: Ukázka třídy sloužící k předání závislostí

```
@Component
public class ComparatorDataAccessObject {

    @Autowired
    public CourierDataService courierDataService;

    @Autowired
    public GeoLocationService geoLocationService;

    @Autowired
    public ElasticDataService elasticSearch;

}
```

Kapitola 4

# Vývoj a testování komparátorů

## 4.1 Úvod

Testování metod doporučování proběhlo zároveň s implementací. Podle jednotlivých testů byly vždy navrženy úpravy (především) komparátoru za účelem zpřesnění doporučování. Jako reference jsou použita reálná přiřazení operátory.

### 4.1.1 Úpravy nutné k testování na historických datech

Testování modulu proběhlo na historických datech. Z toho důvodu bylo nutné učinit několik úprav provádění algoritmu.

- Jelikož není možné využít "aktuální" čas, je hledán ideální kurýr k času reálného přiřazení dané zásilky (tedy čas, kdy operátor kurýrovi přiřadil danou zásilku).
- Stejně tak při dotazu do vyhledávače Elasticsearch musí být zohledněn čas doručování zásilky.
- Historická data poloh kurýrů nejsou umístěna v Elasticsearch, ale v databázi, je tedy nutné hledat polohy databázovým dotazem. To bude mít pravděpodobně za příčinu zpomalení testovacího algoritmu oproti produkčnímu.

Během testování bylo pravděpodobné, že dojde i k situacím, kdy se kurýr, který reálně zásilku doručoval vůbec nebude vyskytovat mezi výsledky. Může to mít několik příčin:

- Kurýr již pro společnost MESSENGER nepracuje a nejsou tedy dostupná jeho data o polohách.

- V testovaném období se stále ještě zaváděla nová kurýrní aplikace umožňující sledování polohy. Kurýr tedy nemusí mít k danému datu informace o polohách a modul jej bude považovat za v ten moment nepracujícího.
- Operátor také mohl přiřadit zásilku v té chvíli nepřihlášenému kurýrovi. Takový kurýr nebude mít záznamy o poloze a modul jej tedy nebude moct doporučit.
- Poloha kurýra může být v danou chvíli velmi nepřesná a komparátor jej kvůli tomu vyřadí.

### 4.1.2 Použitá měřítka kvality komparátorů

Pro posouzení kvality komparátorů bude využito několik měřítek. Jako úspěšně přiřazenou zásilku bude považována taková, kterou modul přiřadí tomu kurýrovi, který ji reálně doručoval.

- Úspěšnost přiřazení – Poměr úspěšně přiřazených vůči celkovému počtu testovaných zásilek. Uváděno v procentech.
- Průměrné pořadí – Udává průměrné umístění kurýra v seznamu seřazeném podle dosaženého skóre.
- Umístění do pátého místa – Stejně jako úspěšnost přiřazení, ale za úspěch se považuje umístění kurýra do pátého místa ve finálním pořadí. Počet je určen na základě toho, že je pět doporučených kurýrů vhodné množství pro zobrazení operátorům.
- Mean reciprocal rank – Jedná se o statistickou metriku pro ohodnocování procesů, které vytváří seznam možných "odpovědí" na určitou otázku (v našem případě kterého kurýra přiřadit). [26]

  Výpočet této metriky je definován následujícím vzorcem (|Q| je počet dotazů a rank je pořadí správného výsledku dotazu, v našem případě pořadí kurýra):

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i}. \tag{4.1}$$

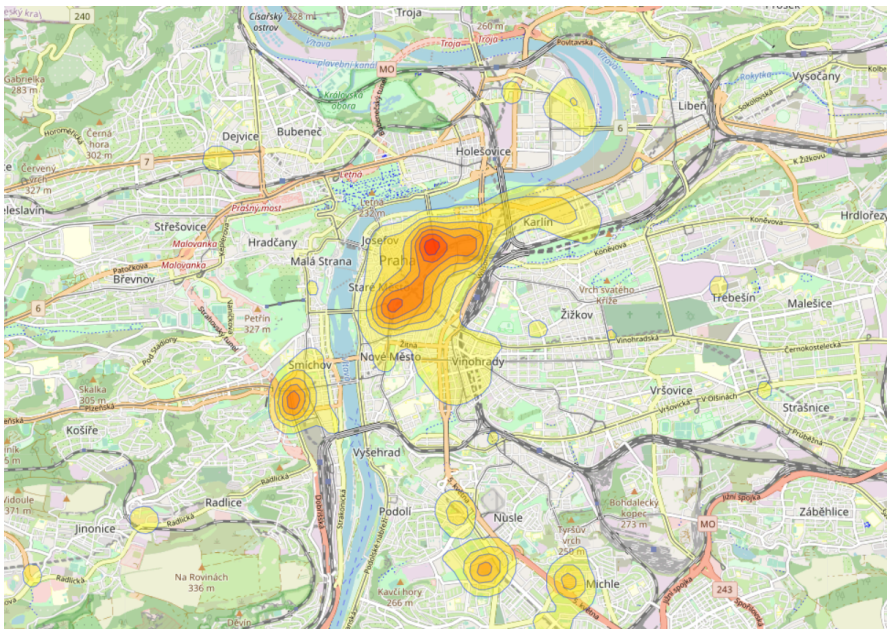
## 4.2 Popis zvolených testovacích dat

Pro testování byla zvolena sada dat z ledna roku 2020. Jedná se o zhruba šest tisíc objednávek, které svým typem odpovídají případu užití vyvíjeného modulu. Splňují tedy následující požadavky.

- Jedná se o zásilky vyzvedávané i doručované po Praze.

- Jedná se o zásilky s produktem STANDARD (tedy přímé cesty z místa vyzvednutí do místa doručení, nejsou využity depa).
- Zásilky byly přiřazeny operátorem.

Na obrázku 4.1 se nachází vizualizace bodů všech objednávek v testovací sadě (místa vyzvednutí i doručení).




Obrázek 4.1: Vizualizace bodů všech objednávek v testovací sadě

Z vizualizace je zřejmé, že většina doručování se odehrává na území Prahy 1 a 2. Na tom po diskuzi s pracovníky není nic překvapivého. Další navštěvovanější místa představují Smíchov a Pankrác. Dá se tedy říct, že primárně se navštěvují různá stravovací a ubytovací zařízení v centru a kancelářské komplexy.

## 4.3 Otestování funkčnosti kostry modulu na komparátoru s náhodným výsledkem

Tento komparátor slouží pouze k otestování funkčnosti modulu a také jako reference, že i jednoduchá heuristika založená na vzdálenosti od místa vyzvednutí funguje lépe než vyloženě náhodné řešení.

### 4.3.1 Výsledky testování

- Úspěšnost přiřazení – 1 %
- Průměrné pořadí – 62,54
- Umístění do pátého místa – 3 %
- MRR – 0,04

## 4.4 Základní komparátor založený na vzdálenosti

Tento komparátor pouze vypočítá vzdálenost místa vyzvednutí zásilky a porovná jej s aktuálními polohami kurýrů. Skóre je tedy rovno vzdálenosti v metrech mezi danými místy.

Je jisté, že tento komparátor bude mít velice nízkou úspěšnost. Předpokladem je také vysoké umístění požadovaných kurýrů v rámci metriky průměrné pořadí (viz 4.1.2).

### 4.4.1 Výsledky testování

- Úspěšnost přiřazení – 14 %
- Průměrné pořadí – 13,23
- Umístění do pátého místa – 32 %
- MRR – 0,268

### 4.4.2 Komentář výsledků

Testy základního komparátoru dopadly poměrně nad očekávání dobře. Už u tohoto velmi jednoduchého posuzování podle vzdálenosti od místa vyzvednutí se podařilo dosáhnout úspěšnosti 14% a ve 32% případů se umístil kurýr do pátého místa. Jedná se tedy o velmi dobrý odrazový můstek a je rozhodně na čem stavět.

## 4.5 Komparátor rozšířený o analýzu místa vyzvednutí a doručení

Prvním zdokonalením bude zohlednění možné shody místa vyzvednutí a doručení se zásilkou, kterou má již kurýr přiřazenou a na dané místo (či místa) v daný den pojede. Bude se kontrolovat shoda na úrovni ulice.

Zvýhodnění daného kurýra bude provedeno ponížením jeho skóre o 30 %.

První test byl proveden na verzi zohledňující pouze místo doručení (protože místo vyzvednutí je zohledněno již výpočtem vzdálenosti).

### 4.5.1 Výsledky prvního testování

- Úspěšnost přiřazení – 16 %
- Průměrné pořadí – 16,2
- Umístění do pátého místa – 36 %
- MRR – 0,274

### 4.5.2 Komentář prvního testování

Zohlednění místa doručení a již přiřazených zásilek kurýra ukázalo drobné zlepšení výkonnosti ve všech metrikách. Nyní se otestuje přidání podobné metriky i pro místo vyzvednutí.

### 4.5.3 Popis vylepšené verze

- Skóre každého kurýra začíná na hodnotě 500 nebo 1000 (otestovány budou obě verze a bude zvolena ta úspěšnější).
- Pokud má kurýr již přiřazenou objednávku vedoucí do místa vyzvednutí analyzované objednávky, je skóre o 30 % sníženo. Jestliže ne, je počítána opět vzdálenost v metrech od aktuální polohy kurýra k místu vyzvednutí.
- Pokud má kurýr již přiřazenou objednávku vedoucí do místa doručení analyzované objednávky, je skóre také poníženo o 30

Problém této verze je zřejmý. Pokud bude mít více kurýrů shodu s místy vyzvednutí a doručení, pak skončí se stejným skóre. Tento problém bude řešen v následující verzi komparátoru. Postup určení skóre je naznačen v v pseudokódu 1.

**Pseudokód 1:** Kombinace vzdálenosti a shody místa vyzvednutí a doručení

```
Result: Kombinace vzdálenosti a shody místa vyzvednutí a doručení
score = 500;
if existuje cesta se stejným místem vyzvednutím přiřazená tomuto
 kurýrovi then
    score = score - ( score / 3 );
else
    if není možné zjistit polohu kurýra then
        score = maximální možné skóre;
        return
    end
    score += vzdálenost polohy kurýra a místa vyzvednutí;
end
if existuje cesta se stejným místem doručením přiřazená tomuto
 kurýrovi then
    score = score - ( score / 3 );
end
```

### 4.5.4 Výsledky třetího testování (počáteční hodnota 500)

- Úspěšnost přiřazení – 13,8 %
- Průměrné pořadí – 14,5
- Umístění do pátého místa – 39 %
- MRR – 0,266

### 4.5.5 Výsledky třetího testování (počáteční hodnota 1000)

- Úspěšnost přiřazení – 14 %
- Průměrné pořadí – 14,65
- Umístění do pátého místa – 39 %
- MRR – 0,265

### 4.5.6 Komentář třetího testování

Počáteční hodnota nevykazuje příliš velký vliv na kvalitu doporučování. Pokračovat se bude s počáteční hodnotou ve výši 500. Nyní provede autor ještě jeden test, který upraví koeficient snížení skóre při shodě místa vyzvednutí ze třiceti na deset procent.

### 4.5.7 Výsledky čtvrtého testování (počáteční hodnota 500, snížení při shodě místa vyzvednutí 10%)

- Úspěšnost přiřazení – 13,6 %
- Průměrné pořadí – 14,63
- Umístění do pátého místa – 41,6 %
- MRR – 0,267

### 4.5.8 Komentář finální testování

Ukázalo se, že nastavení této penalizace na 10% nemá příliš velký vliv, ale mírné zlepšení způsobilo. Koeficient snížení skóre při shodě místa vyzvednutí bude tedy nastaven na 10% i v následujících komparátorech.

## 4.6 Komparátor s penalizací

Další verze komparátoru přidává kontrolu počtu již přiřazených zásilek na kurýrech. Pokud jich má přiřazených hodně, je znevýhodněn oproti ostatním, kteří nejsou tak vytížení. Výpočet penalizace je znázorněn v pseudokódu 2.

**Pseudokód 2:** Výpočet penalizace kurýrů za počet přiřazených zásilek

```
Result: Hodnota penalizace
if počet přiřazených > 10 then
    penalizace = 20 * ( počet přiřazených - 10 );
    if počet přiřazených > 25 then
        penalizace = 50 * ( počet přiřazených - 25 );
    end
else
    aktuální skóre = aktuální skóre - ( aktuální skóre / 10 );
end
```

### 4.6.1 Výsledky testování

- Úspěšnost přiřazení – 19,22 %
- Průměrné pořadí – 9,46
- Umístění do pátého místa – 43 %
- MRR – 0,316

### 4.6.2 Komentář výsledků testování

Výsledky poprvé vykazují větší míru zlepšení. Úspěšnost se podařilo zvednout o zhruba čtyři procenta, ale výrazné zlepšení o více než polovinu lze sledovat na průměrném pořadí.

## 4.7 Pokročilý komparátor

Další verze komparátoru bude implementovat analýzu již přiřazených zásilek naznačenou v kapitole 2.3.3. Jde tedy o to, aby analyzovaná cesta kurýra nevedla mimo již jeho přiřazené zásilky, respektive aby výrazně nevedla mimo oblast, do které se chystá.

V rámci takového dotazu je potřeba projít veškeré cesty přiřazené danému kurýrovi v daný čas. Reálně se opět bude jednat o dotaz do vyhledávače Elasticsearch. Struktura filtrů je zobrazena na ukázce kódu 4.1.

Ukázka kódu 4.1: Dotaz do Elasticsearch na přiřazené zásilky

```
filterBuilders.add( FilterBuilders.andFilter(
    FilterBuilders.termFilter( "kuryrId", courierId ),
    FilterBuilders.rangeFilter( "doruceno" )
        .from( date )
        .to( endOfDay ),
    FilterBuilders.notFilter(
        FilterBuilders.termFilter( "immutableId", routeId ) )
    )
);
```

Vyhledáváme tedy objednávky, které jsou přiřazeny pod analyzovaným kurýrem (atribut `kuryrId`), které ještě nebyly doručeny (atribut `doruceno`, jelikož vyhledáváme v historických datech, vybíráme objednávky, které byly doručeny v daný den, ale až po námi zvoleném čase, ve kterém přiřazujeme analyzovanou objednávku) a jelikož vyhledáváme mezi historickými daty, musíme explicitně vyloučit analyzovanou objednávku (protože kurýr, který ji opravdu doručoval by měl v rámci historické analýzy výhodu).

Pro každou z těchto objednávek je potřeba vypočítat vzdálenost míst vyzvednutí (pokud ještě nebyla vyzvednuta) a doručení od místa doručení analyzované objednávky. Nejnižší vzdálenost v metrech se následně vynásobí koeficientem (budou otestovány tři hodnoty) a přičte ke skóre.

Kurýři, kteří se do oblasti doručení již chystají jsou tak silně zvýhodněni oproti ostatním. Otestovány budou koeficienty 5, 10 a 15.

### 4.7.1 Výsledky testování s koeficientem 5

- Úspěšnost přiřazení – 65,66 %
- Průměrné pořadí – 2,48
- Umístění do pátého místa – 92,75 %
- MRR – 0,71

### 4.7.2 Výsledky testování s koeficientem 10

- Úspěšnost přiřazení – 69,22 %
- Průměrné pořadí – 1,95
- Umístění do pátého místa – 93,98 %
- MRR – 0,775

### 4.7.3 Výsledky testování s koeficientem 15

- Úspěšnost přiřazení – 69,22 %
- Průměrné pořadí – 1,98
- Umístění do pátého místa – 92,86 %
- MRR – 0,768

### 4.7.4 Zhodnocení výsledků

Jak se předpokládalo, zakomponování analýzy místa doručení způsobilo dramatické zvýšení přesnosti algoritmu. téměř 70 % se podařilo přiřadit stejně, jak byly v reálu přiřazeny a téměř vždy se zvolený kurýr objevil v první pětce.

Hodnota koeficientu bude pro finální verzi nastavena na hodnotu 10, protože menší i větší hodnoty vykázaly mírná zhoršení výsledků.

Prostor pro zlepšení nyní spočívá už v pouze v různých kombinacích nastavení koeficientů, které vytváří finální skóre. Jelikož v posledním testování nastalo tak velké zlepšení, je možné zkusit vypustit některé předchozí analýzy a zjistit, jestli mají podstatný vliv na dobrý výsledek. Pokud ne, je možné je vynechat z důvodu zvýšení výkonnosti.

## 4.8 Zjednodušení komparátoru

Nabízí se odstranit analýzu počtu přiřazených zásilek pod kurýrem a zrušení analýzy shody místa doručení s již přiřazenými objednávkami (vhodnost místa doručení stejně analyzujeme v předchozím rozšíření komparátoru).

### 4.8.1 Výsledky testování

- Úspěšnost přiřazení – 65,65 %
- Průměrné pořadí – 2,08
- Umístění do pátého místa – 92,75 %
- MRR – 0,722

Odstranění zohledňování shody s místem doručení a množství již přiřazených zásilek způsobilo mírné zhoršení výsledků. Je tedy možné uvažovat o odstranění těchto částí algoritmu.

## 4.9 Zhodnocení vývoje komparátoru

Ve dvou finálních verzí komparátoru se podařilo dosáhnout velmi kvalitních výsledků. V tabulce 4.1 jsou uvedeny výsledky všech důležitých testování (jsou vynechány některé testy sloužící k nastavení parametrů).

| **Typ komparátoru** | **úspěšnost** | **Průměrné pořadí** | **umístění do 5. místa** | **MRR** |
|---|---|---|---|---|
| Základní | 14 % | 13,23 | 32 % | 0,268 |
| S analýzou místa vyzvednutí | 16 % | 16,2 | 36 % | 0,274 |
| S analýzou obou míst | 13,6 % | 14,63 | 41,6 % | 0,267 |
| S penalizací kurýrů | 19,22 % | 9,46 | 43 % | 0,316 |
| Pokročilý komparátor | 69,22 % | 1,95 | 93,98 % | 0,775 |
| Zjednodušený komparátor | 65,65 % | 2,08 | 92,75 | 0,722 |

Tabulka 4.1: Souhrnné výsledky vývoje komparátoru

Jak již bylo řečeno v předchozí kapitole, zjednodušení komparátoru zapříčinilo pouze menší zhoršení výsledků. Je tedy možné uvažovat o použití obou verzí finálních komparátorů. Směrodatné budou v tomto směru měření rychlosti, které je provedeno v kapitole o výkonnostním testování (viz kapitola 5.2).

Kapitola 5

# Testovací část

## 5.1 Jednotkové testy

Modul obsahuje testy všech metod datových zdrojů, kde to dává smysl (je zbytečné testovat metody sloužící čistě k databázovému dotazu).

V `GeolocationService` je tak otestováno například počítání vzdálenosti mezi dvěma dvojicemi zeměpisných souřadnic nebo je také otestována kontrola přesnosti kurýrních poloh (pokud je poloha méně přesná než 150 m, je ignorována). Otestována je také funkčnost finálního komparátoru.

K testování jsou použity standardní jednotkové testy z Java knihovny JUnit[33]. Skládá se ze třech částí, a to JUnit Platform, JUnit Jupiter a JUnit Vintage. Představuje tak robustní základ pro testování programů v jazyce Java. [27]

Použit je také framework Mockito[34] sloužící k přetížení metod některých komponent. Typické využití je takové, že se u datového zdroje přetíží metoda tak, aby se nedotazovala do databáze a vrátila předem nadefinovaná data. Díky tomuto je možné testovat i metody využívající přístup do databáze bez inicializace celého prostředí aplikace.

Na ukázce kódu je zobrazen test metody `getCourierLastLocation()`. Jak již bylo zmíněno, testuje se, zda metoda správně vrátí výsledek `null`, pokud je přesnost získané polohy horší než 150 m. Využitá je zde právě i knihovna Mockito, která umožňuje nastavit metodě `getCourierLatestLocation()` vrátit předem nadefinovaný výsledek i bez dotazu do Elasticsearch.

[33] Dostupné z: `https://junit.org/`
[34] Dostupné z: `https://site.mockito.org`

Ukázka kódu 5.1: Ukázka jednotkového testu

```
@Test
public void getCourierLastLocation () {
    CourierLocation location = new CourierLocation();
    location.accuracy = 1000.0;
    Mockito
        .when( courierLocationHelper
            .getCourierLatestLocation( 166 )
        )
        .thenReturn( location );

    CourierLocationOperatorAssistentDTO test =
            geoLocationService.getCourierLastLocation( 1666 );

    assertNull( test );
}
```

## 5.2 Výkonnostní testování

V této kapitole je popsán princip a výsledky testování výpočetní náročnosti zvolené verze optimalizátoru. Zajímá nás doba běhu algoritmu na jedné objednávce v reálném čase. Na rozdíl od testování během vývoje budou využity produkční metody na získání relevantních dat.

Největší předpoklad na zrychlení je metoda získání aktuální polohy kurýra. Během vývoje bylo nutné hledat mezi historickými polohami ve standardní SQL databázi, což bylo velmi časově náročné. V produkčním prostředí bude využit index ve vyhledávači Elasticsearch, což bude mít s velkou pravděpodobností vliv na rychlost získání výsledku.

Další oblastí, která může rychlost podstatně ovlivnit, je geokódování, tedy získávání GPS souřadnic podle adres. Tuto službu poskytují většinou poskytovatelé map. V *Kolosu* se využívají tři poskytovatelé, a to Google[35], Seznam[36] a OpenStreetMap (služba Nominatim[37]).

Volání těchto API však zabere nějakou dobu, a tak je vhodné výsledky ukládat i lokálně (takzvaně „cachovat“). Tohle bohužel umožňuje pouze služba Nominatim (podle [28]), ovšem také se jedná o službu, která vrací nejméně relevantní výsledky. Google i Seznam jakékoliv ukládání výsledků zakazují a jejich výsledky tedy není možné cachovat pro rychlejší přístup k nim. [29][30]

Pro zjištění vlivu volání geokódovacích API bude změřen výkon s použití cache a bez ní.

[35]Dostupné z: `https://developers.google.com/maps/documentation/geocoding/start`
[36]Dostupné z: `https://api.mapy.cz/view?page=geocoding`
[37]Dostupné z: `https://wiki.openstreetmap.org/wiki/Nominatim`

| **Typ komparátoru** | **čas běhu s cachováním (ms)** | **čas běhu bez cachování (ms)** |
|---|---|---|
| Základní | 580 | 998 |
| pokročilý | 690 | 1083 |
| zjednodušený | 667 | 1072 |

Tabulka 5.1: Výsledky výkonnostního testování

Výsledky potvrdily očekávané zhoršení výkonu při absenci cache na výsledky geokódovacích dotazů. Jedná se téměř o dvojnásobnou dobu běhu, se kterou však bude nutné počítat kvůli podmínkám užití většiny geokódovacích služeb.

Co se týče rozdílu mezi finálními komparátory (tedy pokročilý komparátor a zjednodušený komparátor), jedná se o rozdíly v řádech milisekund (přesně se jedná o rozdíl 23ms), není tedy nutné použít komparátor zjednodušený.

Je to zapříčiněno především tím, že v obou verzích se dotazujeme zhruba ve stejné míře na již přiřazené cesty kurýrů a taková operace je díky vyhledávači Elasticsearch velice rychlá.

# Závěr

Závěrem by autor rád zdůraznil, že byly naplněny všechny body zadání diplomové práce a vznikl robustní základ, na který se naváže v rámci dalšího vývoje.

Výsledkem je backendová část optimalizačního modulu, která je připravena pro další testování, především v reálném provozu. K implementaci jsou také připraveny návrhy uživatelského rozhraní integrující optimalizační modul do existujícího rozhraní.

## Zhodnocení vývoje

Během vývoje bylo využito několik datových zdrojů a metodik. Jejich váhy byly postupně upraveny tak, aby dávaly co nejlepší výsledky, a i vedení společnosti bylo s výsledky spokojeno.

Během konzultací ohledně návrhů integrace návrhů do uživatelského rozhraní bylo identifikováno několik problémů a ty byly následně opraveny. Návrhy jsou tedy připravené pro realizaci.

Hlavním cílem vývoje bylo vyvinutí použitelného doporučovacího modulu, který bude připraven na další testování (především v reálném provozu) a toho bylo také dosaženo. Doporučovací modul je připraven na integraci a propojení s různými částmi systému.

## Přínos pro autora

Autor se během práce ještě více dozvěděl o procesech společnosti a o trendech při doručování zásilek po Praze. Rozšířil své návrhové a implementační schopnosti v rámci existujícího systému a je připraven k dalšímu vývoji jak tohoto modulu, tak i jiných částí informačního systému *Kolos*.

Autor měl během práce také možnost provést různé analytické a návrhové výzkumy, na které během reálného provozu často není příliš prostoru a času.

## Možnosti budoucího vývoje

Jak již bylo zmíněno, modul je plně připraven na další fáze vývoje. V blízké době je v plánu zahájit testování v reálném provozu, během kterého bude ještě možné zdokonalit vyvinutý komparátor (v první fázi testování poběží optimalizační modul na pozadí a bude se pouze sledovat jeho úspěšnost). Je možné i přidávat další měřítka či metodiky pro dokonalejší doporučení kurýrů. V úvahu připadá například využití různých geolokačních dotazů do vyhledávače Elasticsearch (aktuálně je implementace těchto dotazů problematická z důvodu zastaralé verze Elasticsearch používané ve společnosti) nebo i třeba umělé neuronové sítě.

# Literatura


1. TOTH P.; Vigo, D. *The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications*. Society for Industrial a Applied Mathematics, 2002. ISBN 0-89871-579-2.
2. LARSEN, Allan. The dynamic vehicle routing problem. 2000, s. 3–7.
3. PILLAC, Victor; GENDREAU, Michel; GUÉRET, Christelle; MEDAGLIA, Andrés L. A review of dynamic vehicle routing problems. *European Journal of Operational Research*. 2013, roč. 225, č. 1, s. 1–11.
4. PSARAFTIS, Harilaos N. *Dynamic Vehicle Routing Problems*. Elsevier Sciense Publishers B.V., 1988.
5. VICKREY, William. *Counterspeculation, Auctions, and Competitive Sealed Tenders*. Wiley for the American Finance Association, 1961.
6. AUSUBEL, Lawrence M. *The Lovely but Lonely Vickrey Auction*. Stanford Institute for Economic Policy Research, 2004.
7. Routific. *Route Optimization, Delivery Route Planner - Routific* [online]. ©2020. Dostupné také z: `https://routific.com`.
8. Route Planning for Your Business. *Route4Me* [online]. ©2020. Dostupné také z: `https://www.route4me.com`.
9. TIOBE Index for June 2020 [online]. @2020 [cit. 2020-06-23]. Dostupné z: `https://www.tiobe.com/tiobe-index/`.
10. Most Popular Java Web Frameworks in 2019. *Rollbar* [online]. @2020 [cit. 2020-07-26]. Dostupné z: `https://rollbar.com/blog/most-popular-java-web-frameworks/`.
11. State of API Integration Report 2017 [online]. @2017 [cit. 2020-06-23]. Dostupné z: `https://offers.cloud-elements.com/the-state-of-api-integrations-report-2017-download`.
12. JavaScript Versions. *w3shools* [online]. @2020 [cit. 2020-06-23]. Dostupné z: `https://www.w3schools.com/js/js_versions.asp`.

13. JavaScript [online]. @2020 [cit. 2020-06-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

14. jQuery history. *jQuery* [online]. @2020 [cit. 2020-06-24]. Dostupné z: https://jquery.org/history/.

15. AUSTIN, Andrew. An Overview of AngularJS for Managers. *Andrew Austin* [online]. @2014 [cit. 2020-06-24]. Dostupné z: https://andrewaustin.com/an-overview-of-angularjs-for-managers/.

16. DB-Engines Ranking. *DB-Engines* [online]. @2020 [cit. 2020-06-25]. Dostupné z: https://db-engines.com/en/ranking.

17. Apache Tomcat. *Apache* [online]. @2020 [cit. 2020-06-28]. Dostupné z: http://tomcat.apache.org.

18. Sites, Applications, and Systems that are Powered By Tomcat. *Confluence* [online]. @2020 [cit. 2020-06-28]. Dostupné z: https://cwiki.apache.org/confluence/display/TOMCAT/PoweredBy.

19. HAProxy – The Reliable, High Performance TCP/HTTP Load Balancer. *HAProxy* [online]. @2020 [cit. 2020-06-28]. Dostupné z: http://www.haproxy.org.

20. NIELSEN, Jakob. 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group*. 1994. Dostupné také z: https://www.nngroup.com/articles/ten-usability-heuristics/.

21. NIELSEN, Jakob. Why You Only Need to Test with 5 Users. *Nielsen Norman Group*. 2000. Dostupné také z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

22. Query DSL. *elastic* [online]. @2020 [cit. 2020-06-29]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/java-query-dsl.html.

23. Search API. *elastic* [online]. @2020 [cit. 2020-06-29]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/java-search.html.

24. CRUSOVEANU, Loredana. Intro to Inversion of Control and Dependency Injection with Spring. *Baeldung* [online]. @2020 [cit. 2020-07-10]. Dostupné z: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring.

25. SHVETS, Alexander. Dive Into Design Patterns. In: Refactoring.Guru, 2019, s. 304–307.

26. VOORHEES, Ellen M. The TREC-8 Question Answering Track Report. In: *In Proceedings of TREC-8*. 1999, s. 77–82.

27. JUnit 5 User Guide. *JUnit 5* [online]. @2020 [cit. 2020-07-24]. Dostupné z: https://junit.org/junit5/docs/current/user-guide/.

28. Nominatim Usage Policy. *Nominatim* [online]. @2020 [cit. 2020-07-15]. Dostupné z: `https://operations.osmfoundation.org/policies/nominatim/`.

29. Geocoding API Policies. *Google Maps Platform* [online]. @2020 [cit. 2020-07-15]. Dostupné z: `https://developers.google.com/maps/documentation/geocoding/policies#pre-fetching,-caching,-or-storage-of-content`.

30. Smluvní ujednání pro službu Mapy API. *API Mapy.cz* [online]. @2020 [cit. 2020-07-20]. Dostupné z: `https://api.mapy.cz`.

Dodatek A

# Seznam použitých zkratek

**GUI** Graphical user interface

**XML** Extensible markup language

**JS** Javascript

**AJAX** Asynchronous Javascript And XML

**ES** Elasticsearch

**API** Application Programming Interface

**SQL** Structured Query Language

**MSSQL** Microsoft SQL Server

**HTML** Hypertext Markup Language

**MRR** Mean reciprocal rank

**DI** Dependency Injection

**GPS** Global Positioning System

**PDF** Portable Document Format

**DOM** Document Object Model

**SMS** Short Message Service

Dodatek B

# Zprovoznění zdrojových kódů

Jelikož byl modul implementován v rámci monolitické aplikace *Kolos*, je zprovoznění samotného modulu problematické. Bylo by nutné nahradit všechny třídy převzaté z Kolosu (jedná se o tři třídy) a také nakonfigurovat přístup k datovým zdrojům. Následuje popis tříd, které by bylo nutné nahradit vlastními:

1. **VyhledavaniCesta** – Jedná se o třídu představující jeden řádek ve vyhledávači Elasticsearch. Ve skutečnosti třída obsahuje několik stovek atributů, v modulu se ovšem využívají pouze adresy vyzvednutí a doručení, časy přiřazení, vyzvednutí a doručení a identifikátor kurýra, který objednávku realizoval.

2. **Kuryr** – Tato třída představuje entitu kurýra v databázi. V modulu se využívá pouze identifikátor.

3. **GeolocationLatLon** – Jedná se o návratovou hodnotu geokódovacích služeb v Kolosu. Obsahuje pouze zeměpisnou délku a šířku.

Dále je nutné v třídách `CourierDataService` a `GeolocationService` nahradit volání již implementovaných služeb v Kolosu za vlastní služby dotazující se do vlastních datových zdrojů. Třída `ElasticDataService` je sama třídou realizující dané dotazy, je ovšem potřeba mít v projektu nakonfigurované napojení na databázi Elasticsearch. Může být také problematické to, že v Kolosu je využita starší verze Elasticsearch (verze 1.6) a využité filtry a dotazy nemusí být s aktuálními verzemi kompatibilní.

Pro správnou funkčnost je také nutné mít k dispozici uložené aktuální polohy kurýrů a co nejčastěji je aktualizovat. Pokud by nebyly aktuální, modul nemusí vracet příliš dobré výsledky.

Dodatek C

# Obsah přiloženého CD

- `README.txt` ..... stručný popis obsahu CD
- `zdrojové kódy`
  - `src.zip` ..... zdrojové kódy implementace
  - `DP_Lacny_Jakub_2020.zip` ... zdrojová forma práce ve formátu LaTeX
  - `README.txt` ..... Informace ke zdrojovým kódům implementace
- `wireframy` ..... wireframy vytvořené v rámci návrhové části
- `text` ..... text práce
  - `DP_Lacny_Jakub_2020.pdf` ..... text práce ve formátu PDF