TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Dolování dat z databáze podnikového informačního systému Helios Orange

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Tomáš Košek**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Data mining from the database of the Helios Orange enterprise information system

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Tomáš Košek**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Košek**
Osobní číslo:	**M14000044**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Dolování dat z databáze podnikového informačního systému Helios Orange**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s obsahem databáze a hlavními funkcemi IS Helios Orange pro implementaci firemních procesů v malém nebo středně velkém podniku.
2. Vyberte vhodné nástroje pro implementaci vlastní nadstavby tohoto IS, která umožní uložená data agregovat, filtrovat a prezentovat v uživatelsky přívětivé přehledné formě.
3. Nástavba by měla uživatelům umožnit snadný výběr vstupních dat pro analýzu, definovat vlastní metody jejich zpracování a vyhodnocení, nástroj by měl být snadno rozšiřitelný o nové agregační či jiné funkce.
4. Na vybraných příkladech pokročilého zpracování dat demonstrujte správnou funkci vytvořené nadstavby systému a v závěru shrňte výhody a nevýhody vámi implementovaného řešení.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

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

Seznam odborné literatury:

[1] **LACKO, L'uboslav, 2013. Mistrovství v SQL Server 2012: [kompletní průvodce databázového experta]. Brno: Computer Press. ISBN 9788025137734.**

[2] **Online příručka HELIOS Orange [online]. 2015. [cit. 2015-10-20]. Dostupné z: https://forum.helios.eu/orange/doc/cs/Helios_Orange_Hlavn%C3%AD_strana**

Vedoucí bakalářské práce: **Ing. Jan Kraus, Ph.D.**

Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2017**

Termín odevzdání bakalářské práce: **14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.5.2018

Podpis: 

Abstrakt

Tato bakalářská práce popisuje tvorbu aplikace pro práci nad daty z informačního systému Helios Orange. Zprvu je prezentována klientská desktopová aplikace napsaná v jazyce C# pro načítání dat z databázového souboru systému Helios. Tato data umožňuje filtrovat dle předem definovaných filtrů nebo umožňuje uživateli napsat svůj vlastní filtr dle uvážení. To je možné díky implementované možnosti kompilace kódu za běhu aplikace. Dále popisuje vytvoření HTTP serveru za pomocí frameworku Node.js. Ten je důležitý k zpřístupnění skriptů, napsaných v jazyce JavaScript, které slouží pro práci s API od společnosti Seznam.cz. Toto API umožňuje získat z adres zákazníků přesné souřadnice, které zobrazuji na mapě jako přehled, kde se nachází nejvíce klientů, nebo s nimi plánuji služební cesty. K vytvoření ideální trasy používám přesné algoritmy branch and bound a brute force, heuristický algoritmus nearest neighbour a jejich kombinace. Následně popisuji jejich efektivitu a náročnost na systémové prostředky.

Klíčová slova:

filtrování dat, problém obchodního cestujícího, api.mapy.cz, Helios Orange, souřadnicový systém WGS84

Abstract

This bachelor thesis describes creation of an application to work with data from Helios Orange information system. Firstly, presents the client desktop application written in C# language for loading data from the Helios system database file. This data allows to filter with the predefined filters or allows the user to write their own filter, according what they need. This is possible thanks to implementation of runtime code compilation. After that describes how to create HTTP server using Node.js framework. It is important for making scripts, written in JavaScript language, available so they can be used to work with API from Seznam.cz company. This API allows to get the exact coordinates from customer's addresses, which I show on map like an overview, where most of clients are located or planning business trip with them. I use exact algorithms like branch and bound

or brute force and heuristic algorithm nearest neighbour or their combinations to create an ideal route. Lastly, I describe their efficiency and complexity of system resources.

Key words:

data filtering, travelling salesman problem, api.mapy.cz, Helios Orange, coordinate system WGS84

Obsah

1 Úvod 13

2 Informační systém Helios 15

2.1 SQL Server 15

2.2 Požadovaná data 15

3 Desktopová aplikace pro načtení a zobrazení dat 16

3.1 Návrhový vzor MVVM, „binding“ a „commands“ 16

3.1.1 Model 16

3.1.2 View 16

3.1.3 ViewModel 17

3.1.4 Výhody a nevýhody 17

3.2 Grafické rozhraní WPF 18

4 Helios Extender C# 19

4.1 Úvodní okno aplikace 19

4.2 Filtrování dat 20

4.3 Kompilace uživatelského kódu 21

4.3.1 Použité technologie 21

4.3.2 Textový dokument jako skript 22

5 Souřadnice WGS84 a práce s api.mapy.cz 25

5.1 Souřadnicový systém WGS84 25

5.2 Převod WGS84 na double 25

5.3 Výpočet vzdálenosti mezi dvěma body 26

5.4 Api.mapy.cz 26

5.4.1 Získání souřadnic – geokódování 27

5.4.2 Získání adresy – reverzní geokódování 27

6 Helios Extender – JavaScript 28

6.1 WebSocket 28

6.1.1 Server C# 28

6.1.2 Instalace 28

6.1.3 Inicializace serveru C# 28

6.1.4 Klient JavaScript 30

6.2 Singleton C# 30
6.2.1 Vytvoření návrhového vzoru Singleton 30
6.3 Spuštění http-serveru pro práci s JavaScriptovými soubory 31
6.3.1 Node.js 31
6.3.2 Instalace 32
6.3.3 Použití 32
6.3.4 Příkazová řádka systému Windows 33
6.4 getWGS84.html 34
6.5 selectWGS84.html 34
6.6 showAllWGS84.html 35
6.6.1 Shlukování bodů 35
6.7 makeRoute.html 36
6.7.1 Práce s SMap.URL.Route 36
6.8 Knihovna Internet Explorer 37
7 Naplánování služební cesty 39
7.1 Algoritmus nearest neighbour 39
7.1.1 Princip 39
7.2 Algoritmus brute force 40
7.2.1 Princip 40
7.2.2 Konkrétní implementace 40
7.2.3 Výsledky 41
7.3 Algoritmus branch and bound 42
7.3.1 Prioritní binární halda 42
7.3.2 Princip algoritmu branch and bound 43
7.3.3 Konkrétní implementace 44
7.3.4 Výsledky 45
7.4 Shrnutí 46
8 Závěr 47
Literatura 49
A Uživatelský manuál – jak vytvořit filtr 53
B Schéma aplikace 55
C Obsah přiloženého CD 56

Seznam zdrojových kódů

Zdrojový kód 1: Ukázka View, jazyka XAML, binding a commands 17

Zdrojový kód 2: Jak přidat do kompilátoru vlastní objekt "Customer" [21] 21

Zdrojový kód 3: Jak umožnit v kompilovaném kódu používání kolekce List<T>..... 22

Zdrojový kód 4: Hlavička metody, do které bude vložen uživatelský kód 22

Zdrojový kód 5: Nezbytná vlastnost každého uživatelského kódu – návratová hodnota za klíčovým slovem "return" 23

Zdrojový kód 6: Ukázka komplexnějšího kódu, navrácení kolekce List<Customer> ... 23

Zdrojový kód 7: Přepočet souřadnic ze stupňů, minut a vteřin na jedno desetinné číslo26

Zdrojový kód 8: Přidání online API od společnosti Seznam.cz 27

Zdrojový kód 9: Práce s třídou SMap.Geocoder 27

Zdrojový kód 10: Práce s reverzním geokódováním 27

Zdrojový kód 11: Instalace WebSocket serveru 28

Zdrojový kód 12: Přidání reference na WebSocketSharp 28

Zdrojový kód 13: Inicializace serveru WebSocketSharp 29

Zdrojový kód 14: Ukázka implementace třídy dědící od WebSocketBehavior 29

Zdrojový kód 15: Ukázka implementace klienta protokolu WebSocket v jazyce JavaScript..... 30

Zdrojový kód 16: Použití návrhového vzoru Singleton pro uložení informace po celou dobu běhu aplikace 31

Zdrojový kód 17: Instalace NPM balíčku http-server 32

Zdrojový kód 18: Spuštění HTTP serveru na portu 45222..... 32

Zdrojový kód 19: Spuštění příkazové řádky, ze které nastartuji HTTP server..... 33

Zdrojový kód 20: Ukončení příkazové řádky a všech potomků..... 33

Zdrojový kód 21: Konstruktor třídy SMap.Marker.Clusterer..... 36

Zdrojový kód 22: Konstruktor třídy SMap.URL.Route 37

Zdrojový kód 23: Ukázka vytvoření URL pro získání naplánované trasy na web mapy.cz 37

Zdrojový kód 24: Knihovna pracující s internetovým prohlížečem Internet Explorer... 38

Zdrojový kód 25: Vytvoření objektu InternetExplorer a vyvolání prohlížeče na webové adrese uložené v proměnné URL 38

Seznam obrázků

Obrázek 1: Úvodní okno mé aplikace napsané v jazyce C# 19

Obrázek 2: Nastavení filtrů pro výběr konkrétních zákazníků 20

Obrázek 3: Výsledek aplikování filtru z Obrázek 2 20

Obrázek 4: Ukázka výběru skriptu 21

Obrázek 5: Výsledek po zkompilování uživatelského kódu z ukázky Zdrojový kód 5 . 23

Obrázek 6: Výsledek po zkompilování ukázky kódu z Zdrojový kód 6 24

Obrázek 7: Uživatel vybírá, kde se přesně nachází firma TRADECOM INC 35

Obrázek 8: Ukázka shlukování bodů v Praze [28][34] 36

Seznam grafů

Graf 1: Počet iterací algoritmem v závislosti na počtu adres, BF = brute force, NN = nearest neighbor 42

Graf 2: Využití paměti RAM v závislosti na počtu adres, BaB = branch and bound, NN = nearest neighbor 45

Seznam rovnic

Rovnice 1: Počet iterací algoritmem brute force bez dalších vylepšeních 40

Rovnice 2: Výpočet odkazu na dva potomky v binární haldě 43

Rovnice 3: Výpočet odkazu na rodiče v binární haldě 43

Seznam zkratek

C#	C Sharp, objektový programovací jazyk
CMD	Příkazová řádka systému Windows
API	Application Programming Interface, rozhraní pro programování aplikací
HTML	HyperText Markup Language, značkovací jazyk pro tvorbu webových stránek
WGS84	World Geodetic System 1984, geodetický standard, který definuje souřadnicový systém
URL	Uniform Resource Locator, jasně strukturovaný řetězec, který slouží k přesné definici zdroje (cíle)
WPF	Windows Presentation Foundation, knihovna tříd pro tvorbu GUI
MVVM	Model–View–ViewModel, návrhový vzor pro WPF aplikace psané v C#
XAML	Extensible Application Markup Language, jazyk určený k popisu GUI
GUI	Graphical User Interface, uživatelské grafické rozhraní aplikace
DPI	Dots Per Inch, kolik pixelů se vejde do jednoho palce
CSS	Cascading Style Sheets, jazyk pro grafickou úpravu webových stránek
CNC	Computer Numerical Control, počítačem řízený obráběcí stroj
NPM	Natural Polyglot Machine, balíčkovací systém pro JavaScript
SQL	Structured Query Language, standardizovaný jazyk pro práci s databázemi
HTTP	Hypertext Transfer Protocol, protokol pro výměnu HTML dokumentů

1 Úvod

V této bakalářské práci popisuji tvorbu aplikace pro získání informací z dat informačního systému Helios, která daný systém nenabízí. Jedná se o práci s adresami jednotlivých zákazníků, ze kterých získávám souřadnice a vytvářím zajímavé pohledy na jejich lokalizaci a plánuji služební trasy.

Vytvořený software se skládá z následujících částí:

- Z uživatelsky přívětivé aplikace napsané v jazyce C#, která slouží k zobrazení a správě dat uložených v databázi běžící na Microsoft SQL serveru 2016. Dále dává uživateli možnost provádět filtrování dat, ať za pomocí předem definovaných možností, nebo si může napsat jednoduchým způsobem vlastní skript. Ten mu umožní si zobrazit přesně ta data, která právě požaduje. A v poslední řadě aplikace řadí adresy zákazníků, aby vytvořily optimální služební cestu.
- Ze skriptů napsaných v jazyce JavaScript pro práci se souřadnicemi. Konkrétně se jedná o práci s API od společnosti Seznam.cz. Jednak pro získání souřadnice na základě adresy, dále pro zobrazení zákazníků přehledně na mapě, nebo pro vytvoření URL z adres, které jsou součástí služební cesty.
- Z HTTP serveru, který slouží pro zpřístupnění výše zmíněných JavaScriptových skriptů.

Zprvu se věnuji informačnímu systému Helios Orange. Mým cílem je zjistit, jak s ním a SQL serverem pracovat, jak má strukturovanou databázi, jaká data od něj budu vyžadovat. Dále vybrat vhodné vývojové prostředí, návrhový vzor a další technologie pro vytvoření aplikace, která zvládne komunikovat s API od spolčenosti Seznam.cz, které je napsané v JavaScriptu. Následně vymyslet způsob filtrování dat a možnost, jak udělat tuto aplikaci snadno rozšiřitelnou o nové agregační či jiné funkce.

Dalším bodem mé práce je seznámení se s programováním v JavaScriptu s API od společnosti Seznam.cz. Zjistit, jakým způsobem si lze získat souřadnice na základě uložené adresy, a jak s těmito souřadnicemi pracovat. Dále jak zobrazit libovolné adresy jako značky na mapě, anebo jak vytvořit URL s naplánovanou trasou na web mapy.cz.

Následně vymyslet, jak vytvořit HTTP server a zajistit komunikaci mezi aplikací napsanou v jazyce C# a zmíněnými JavaScriptovými soubory. V poslední řadě je mým cílem seznámit se s algoritmy na řešení známého problému obchodního cestujícího, abych mohl zajistit optimalizaci návrhu služební cesty tak, aby cesta byla co nejkratší.

2 Informační systém Helios

Systém Helios vyvíjený společností Asseco Solutions nabízí několik různých řešení pro řízení společnosti: Helios Green, Helios Orange, Helios Easy, Helios Red, Helios Fenix a další. Produkty Helios používá již více jak 14 000 zákazníků, včetně firem jako Seznam.cz, ČEZ Energo nebo Lomax & Co.

Helios Orange ve firmě Lomax & Co umožňuje například spravovat příjmy zakázek, logistiku, sklady, výrobní plán, který souvisí s výrobními příkazy, až po propojení systému s CNC robotizovaným pracovištěm a mnohé další [2].

Helios Green ve společnosti Seznam.cz například eviduje volná pracovní místa, sleduje nábor, připravuje pracovní smlouvy, zaznamenává interní a externí tréninky zaměstnanců, informuje o ukončení zkušební doby nebo o nutnosti provést povinná periodická školení. Dále se stará o částečnou automatizaci manuálního zadávání a zaúčtování účetních dokladů a spoustu dalších věcí 49[3].

2.1 SQL Server

Získání dat z demoverze nebylo jednoduché. Instalace informačního systému Helios se zároveň snaží sama připojit databázový soubor (HeliosDemo.mdf) k SQL serveru, což nefunguje příliš dobře. Bohužel požadovaný .mdf soubor nebylo možné získat jiným způsobem než úspěšnou dokončenou instalací. Na několikátý pokus se instalace Helios Orange a SQL serveru úspěšně dokončila a já si mohl soubor HeliosDemo.mdf připojit k SQL serveru dle mého uvážení a přistupovat k jeho datům.

2.2 Požadovaná data

Vstupní data do mé aplikace získávám z databáze demoverze systému Helios Orange. Tato databáze obsahuje téměř 2000 různě propojených tabulek. Do mé aplikace si načítám data například o uložených klientech, vydaných a přijatých fakturách nebo skladové karty zboží včetně záznamu, jaký zákazník si které zboží koupil nebo firmě prodal.

3 Desktopová aplikace pro načtení a zobrazení dat

Pro napsání klientské aplikace jsem si vybral jazyk C# a vývojové prostředí Visual Studio 2017, populární návrhový vzor MVVM s moderním využitím tzv. bindování a knihovnu tříd pro napsaní grafického rozhraní WPF.

3.1 Návrhový vzor MVVM, „binding“ a „commands“

Návrhový vzor Model, View, ViewModel vyčnívá svými vlastnostmi právě ve třídě zvané ViewModel. Ta se stará o chod celé aplikace. Nejenže má pod svými křídly uložené instance Modelů s nejrůznějšími daty, ale má v sobě uložené i jak aktuálně vypadá View. View totiž takzvaně binduje ViewModel, což ve výsledku znamená, že každá komponenta ve View má nastavenou vlastnost Binding, ve které se odkazuje na nějakou proměnnou ve ViewModelu. Celé je to řízené událostmi INotifyPropertyChanged, které promítnou změnu způsobenou ve ViewModelu na dané komponentě. Další podmínkou je mít data vlastních modelů v kolekcích (například ObservableCollection<T>), které vyvolávají události, pokud je přidán nebo odebrán prvek [11].

3.1.1 Model

Model se v tomto návrhovém vzoru příliš neliší od ostatních. I zde slouží k definici objektů a k určení, jaké datové typy budou mít jeho atributy. Patří sem například uživatelsky definovaná rozhraní, abstraktní třídy a jejich potomci, výčty hodnot nebo jakékoliv jiné třídy, které slouží jako předpis toho, jaká data bude třída obsahovat.

Jediný rozdíl je tedy v tom, že tyto třídy musí dědit od INotifyPropertyChanged. Takže musí mít implementovanou metodu OnPropertyChanged, která se stará o změny ve View i když změna hodnoty proběhne například v objektu uvnitř objektu.

3.1.2 View

View představuje uživatelské rozhraní, v tomto případě napsané v jazyce XAML. Je to jediná část aplikace, se kterou uživatel opravdu pracuje, a na oplátku mu jsou zobrazována data, které potřebuje vidět.

S pomocí tak zvaného „binding“ se View synchronizuje s ViewModelem a zajišťuje si tak soběstačnost. Nemusím při každé změně dat ihned informovat View, aby se o těchto změnách dozvěděl i uživatel. Tato operace se totiž děje automaticky, pokud nastane změna hodnoty proměnné, kterou View binduje.

Naproti tomu „commands“ slouží k provádění operací po stisknutí tlačítka a zjišťuje, zdali vůbec je možné tuto operaci provést. Dosáhne se toho implementací rozhraní ICommand, které vyžaduje implementaci dvou metod. První je metoda s návratovým datovým typem bool, která rozhoduje, jestli je možné spustit druhou metodu, jež vykoná naprogramované operace po stisknutí tlačítka.

```
<TextBlock Text="{Binding WareInfo}" />
<ComboBox ItemsSource="{Binding Path=Ware}"
SelectedValue="{Binding path=SelectedWare}" />
<TextBlock Grid.Row="2" Text="{Binding RegionInfo}" />
<ComboBox Grid.Row="3" ItemsSource="{Binding
Path=Regions}" SelectedValue="{Binding
Path=SelectedRegion}" />
<Button Grid.Row="4" Command="{Binding SaveMyFilter}"/>
```

Zdrojový kód 1: Ukázka View, jazyka XAML, binding a commands

3.1.3 ViewModel

A právě ViewModel je ten, kdo má v sobě uložené hodnoty bindované z View a commandy, které se vykonají po uživatelské interakci. Dále se stará o to, aby View dostalo prezentovatelná data z Modelu a drží si stav aplikace.

3.1.4 Výhody a nevýhody

Tento návrhový vzor není vhodný pro začínající programátory nebo pro vytváření své první aplikace „Hello world“. Dále bych jej nepoužil při programování jednoduchých aplikací o dvou tlačítkách s dvěma popisy. Nachází se v něm mnoho kódu, který akorát krátký jednoduchý kód znepřehlední a udělá složitější na ladění programu.

Velkou výhodu tento návrhový vzor přináší ve větších projektech, kde nám umožní snadnou kontrolu nad daty, která se posílají do View. Protože místo udržování několika míst v aplikaci, kde danou proměnnou měním a následně o tom informuji View, zde mám jen jednu metodu, která se stará o upravení hodnoty z Modelu do prezentovatelné podoby.

3.2 Grafické rozhraní WPF

Grafické rozhraní WPF je textový soubor, který pomocí jazyku XAML popisuje, jak má vypadat GUI. Je přehledný, většina komponent se jmenuje stejně jako ve Windows Form Application, avšak je podstatně rychlejší, modernější a lépe pracuje s DPI.

Důvodů proč přejít ze staršího Windows Forms na Windows Presentation Foundation je hned několik. Hlavní výhodou je, že na rozdíl od absolutního umísťování prvků, je zde programátor veden k umisťování prvků do mřížky o různých velikostech, která se přizpůsobí velikosti okna. Další obrovskou výhodou je vektorové vykreslování za pomocí Direct3D, což znamená převedení výkonu potřebného k vykreslení obsahu na grafickou kartu.

Nevýhodou může být opět použití v aplikacích, kde figuruje jedno tlačítko a nic víc, anebo při výuce začátečníků.

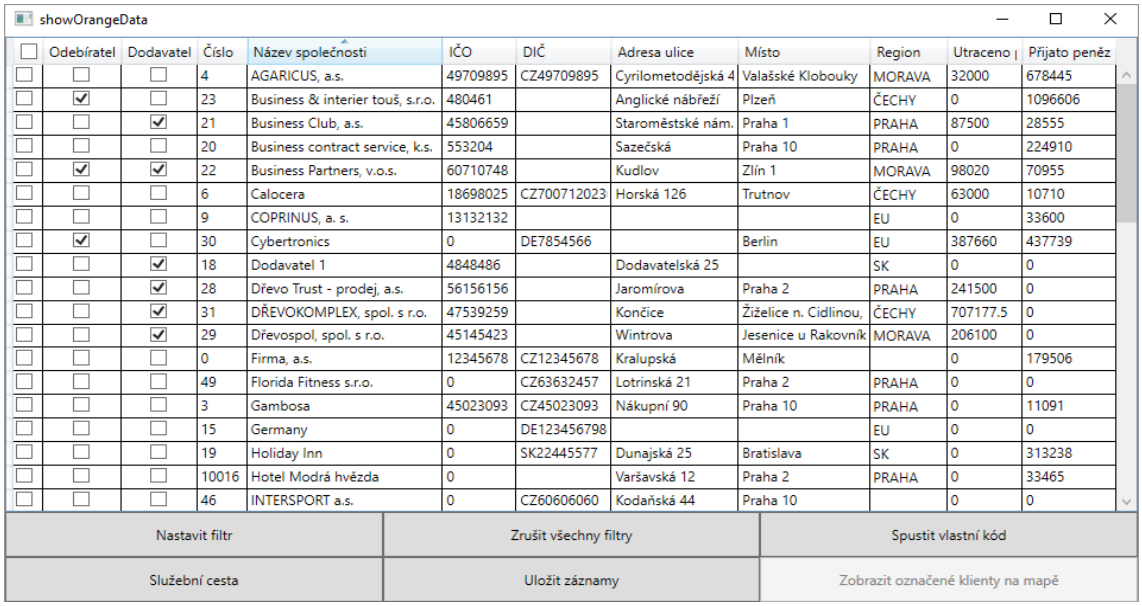
4 Helios Extender C#

Helios Extender je název mé aplikace napsané v jazyce C# za pomocí již zmíněného návrhového vzoru MVVM.

4.1 Úvodní okno aplikace

Úvodní okno aplikace zobrazuje seznam všech klientů uložených v databázi systému Helios a šest tlačítek umožňujících provádění různých operací nad těmito daty.

- *Nastavit filtr* vyvolá dialogové okno pro výběr filtru
- *Spustit vlastní kód* poskytuje možnost vybrat vlastní kód v textovém dokumentu, který bude zkompilován, spuštěn a výsledek zobrazen
- *Zrušit všechny filtry* zruší jak nastavené filtry, tak vymezení dat, které vzniklo z kódu v textovém souboru a zobrazí znovu všechny zákazníky
- *Služební cesta* naplánuje optimální trasu na mapě mezi označenými klienty
- *Uložit záznamy* aktualizuje do databáze zákazníky, u kterých byly pozměněny údaje
- *Zobrazit označené klienty na mapě* zobrazí vybrané zákazníky (i všechny) na mapě od mapy.cz

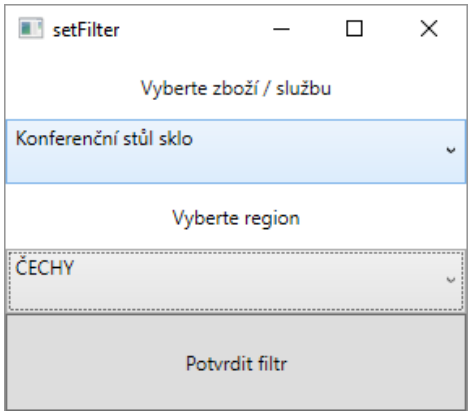


Obrázek 1: Úvodní okno mé aplikace napsané v jazyce C#

Při načítání seznamu zákazníků zároveň procházím všechny vydané anebo přijaté faktury. Podle toho, komu nebo od koho byla faktura vypsána, přičtu její hodnotu danému klientovi. Dále procházím všechny položky této faktury a uložím si také, jaké zboží daný klient nakupuje či prodává pro pozdější agregování a filtrování dat.

4.2 Filtrování dat

Jednou z funkcí mé aplikace je filtrování zobrazených dat neboli zákazníků. Jako zajímavé možnosti, podle čeho je možné filtrovat, jsem si vybral následující. Podle zboží nebo služby, kterou si zákazník někdy v minulosti koupil anebo podle regionu.



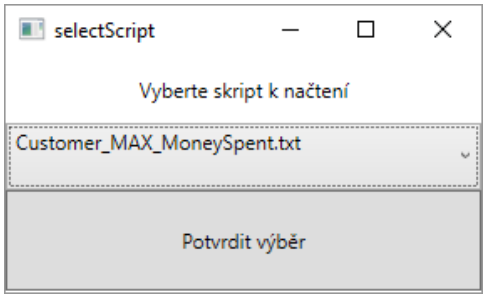
Obrázek 2: Nastavení filtrů pro výběr konkrétních zákazníků

Abych mohl uživateli nabídnout možnost filtrování podle služby, zboží nebo regionu, musím si nejdříve tato data získat z databáze. Z tabulek TabRegion a TabKmenZbozi si načtu informace o všech dostupných možnostech filtrování a uložím do kolekcí, které bindují ComboBoxy ve View. Tlačítko *Potvrdit filtr* se zpřístupní poté, co uživatel vybere alespoň jednu položku v libovolném ComboBoxu.

Obrázek 3: Výsledek aplikování filtru z Obrázek 2

4.3 Kompilace uživatelského kódu

Jaká data bude konkrétní firma nebo konkrétní uživatel požadovat, je velice těžké předpovědět a není prakticky možné připravit aplikaci na všechny možnosti. Z tohoto důvodu jsem naprogramoval možnost kompilace kódu za běhu programu. Nejprve je uživatel vyzván k výběru skriptu, který chce spustit.



Obrázek 4: Ukázka výběru skriptu

4.3.1 Použité technologie

Pro kompilaci kódu za běhu programu používám následující dvě třídy: CSharpCodeProvider a CompilerParameters. První zmíněná se stará o samotnou kompilaci, kdy pro zavolání její metody CompileAssemblyFromSource jsou vyžadovány dva parametry.

Prvním parametrem je druhá, již zmíněná, třída CompilerParameters, která má za úkol připravit všechny knihovny a objekty potřebné pro vykonání kompilace kódu. V mém případě předávám kompilátoru knihovny potřebné pro práci s kolekcemi List<T> (mscorlib.dll) a ObservableCollection<T> (System.dll). Dále jej seznamuji s mými vytvořenými objekty. Například přidání mého objektu Customer docílím následujícím způsobem.

```
parameters.ReferencedAssemblies.Add(typeof(Customer).Assembly.Location);
```

Zdrojový kód 2: Jak přidat do kompilátoru vlastní objekt "Customer" [21]

Jako druhý parametr metoda vyžaduje kód v datovém formátu string. V kódu nesmí chybět odkazy na již přidané knihovny jako například:

```
using System.Collections.Generic
```

Zdrojový kód 3: Jak umožnit v kompilovaném kódu používání kolekce List<T>

pro přidání kolekce List<T>. Následuje název jmenného prostoru, třídy a vytvoření metody Main, která sice nemusí nic obsahovat, ale bez její přítomnosti se kompilátor neobejde. A nakonec metoda s parametry, jež bude mít uživatel přístupné, do které se bude vkládat kód načtený z textového dokumentu [22].

4.3.2 Textový dokument jako skript

Spouštěný kód musí mít formát jedné metody. Konkrétně text, který uživatel vytvoří, bude vložen v následující ukázce kódu namísto slova „func“. To znamená, že v textovém dokumentu se může napsat kód, libovolně pracující se zákazníky aktuálně vypsanými na hlavním oknu aplikace. Jakékoliv podmínky, cykly, průchody datových struktur, inicializování vlastních proměnných a další klíčová slova, která běžně nalézáme uvnitř metod jsou vítána.

Logickou podmínkou je, že kód musí mít návratovou hodnotu. Musí tedy obsahovat klíčové slovo return a vracet nějaký objekt, jak už hlavička metody v následující ukázce napovídá.

```
public static object Function
(ObservableCollection<Customer> customers)
{
     func
}
```

Zdrojový kód 4: Hlavička metody, do které bude vložen uživatelský kód

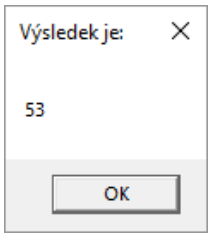
Pokud uživatel vrátí libovolný z primitivních datových typů, bude mu zobrazeno vyskakovací okno s výsledkem, který navrátil.

Například pokud si bude chtít zjistit kolik má registrovaných zákazníků uložených v databázi, napíše do textového souboru:

```
return customers.Count;.
```

Zdrojový kód 5: Nezbytná vlastnost každého uživatelského kódu – návratová hodnota za klíčovým slovem "return"

Soubor uloží, načte v aplikaci a bude mu zobrazeno následující vyskakovací okno.



Obrázek 5: Výsledek po zkompilování uživatelského kódu z ukázky Zdrojový kód 5

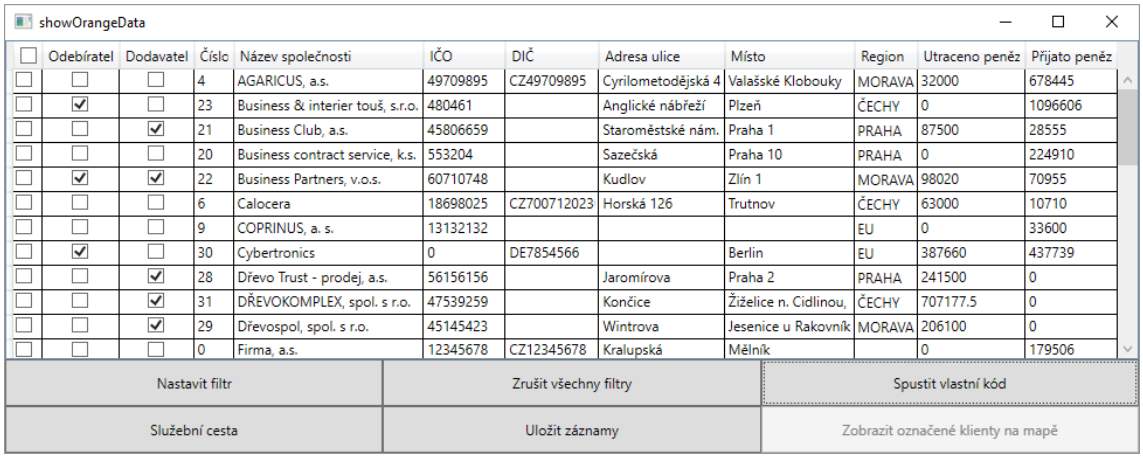
Jestliže si bude chtít autor kódu vybrat jednoho zákazníka (například toho s největším obratem) nebo skupinu klientů, kterou si vybere na základě jeho aktuálních podmínek, jednoduše navrátí objekt Customer nebo List<Customer>. Tito zákazníci se mu poté objeví v hlavním okně aplikace.

Například pokud bude chtít zákazníky, se kterými je firma finančně aktivní, napíše si následující skript.

```
List<Customer> cust = new List<Customer>();
foreach(Customer c in customers)
{
     if(c.MoneySpent > 0 || c.MoneyReceived > 0)
     {
          cust.Add(c);
     }
}
return cust;
```

Zdrojový kód 6: Ukázka komplexnějšího kódu, navrácení kolekce List<Customer>

Po nahrání skriptu do aplikace se do kolekce, která je bindovaná z View načtou zákazníci, které jsem dostal z textového dokumentu, poté bude okno vypadat následovně.



Obrázek 6: Výsledek po zkompilování ukázky kódu z Zdrojový kód 6

5 Souřadnice WGS84 a práce s api.mapy.cz

Aplikace umožňuje mezi vybranými zákazníky naplánovat služební cestu. Jednoduše stačí zaškrtnout klienty, které uživatel potřebuje v rámci služební cesty navštívit, a kliknout na příslušné tlačítko. Abych mohl naplánovat takovou cestu, potřebuji mapový podklad. Pro možnost práce s mapovým podkladem a nejasnými adresami potřebuji souřadnicový systém.

5.1 Souřadnicový systém WGS84

WGS84 je souřadnicový systém pro orientaci na planetě Zemi. Jak už číslovka v názvu napovídá, tento systém byl vyvinut v roce 1984 a ve velké míře se používá do dnes, včetně užití na jedněch z nejrozšířenější mapových podkladů od společnosti Google.

Byl vyvinut ministerstvem obrany Spojených států amerických a od roku 1998 je součástí Armády České republiky při kooperacích s armádami Severoatlantské aliance NATO [9].

Typická zeměpisná šířka je na našem území v rozmezí 48–51 stupňů a délka 12–18 stupňů [13]. A neobyčejný bod 0 stupňů severní nebo jižní zeměpisné šířky a 0 stupňů východní nebo západní zeměpisné délky, neboli střed na Zemi, leží u Afriky v Guinejském zálivu.

5.2 Převod WGS84 na double

Například budova A Technické univerzity v Liberci leží na souřadnicích: 50°46'22.121"N, 15°4'19.468"E. Kde stupně, minuty a následně vteřiny s jejich desetinnou částí detailně popisují bod na mapě. Pro představu, přičtení nebo odečtení jedné vteřiny (nikoliv její desetinné části) změní pozici přibližně o 31 metrů [20]. Jako další v zápisu souřadnice se nachází písmena „S“ („south“ neboli jižní polokoule) nebo „N“ („north“ neboli severní polokoule) pro zeměpisnou šířku a písmena „E“ („east“ neboli východní polokoule) a „W“ („west“ neboli západní polokoule) pro zeměpisnou délku.

S tímto typem zápisu souřadnic je velice složité pracovat v programování nebo v matematice. Například u výpočtu vzdálenosti mezi dvěma body. Pro tyto účely si

souřadnice převádím do desetinné podoby následujícím způsobem.

```
decimal výsledek = Stupně + (Minuty / 60) + (Vteřiny /
3600)) * (Stupně < 0 ? -1 : 1));
```

Zdrojový kód 7: Přepočet souřadnic ze stupňů, minut a vteřin na jedno desetinné číslo

Jelikož si nechci k takovému číslu ještě ukládat, na které polokouli se daná souřadnice nachází, používají se kladná čísla pro severní a východní polokouli a čísla záporná pro polokouli jižní a západní.

5.3 Výpočet vzdálenosti mezi dvěma body

K tomu, abych mohl naplánovat ideální trasu, tedy poskládat zákazníky v určitém pořadí, které bude znamenat nejkratší ujetou vzdálenost, potřebuji zjistit vzdálenosti jednotlivých klientů od sebe.

K tomuto účelu jsem si vybral Haversinův vzorec, který pracuje s takzvanou ortodromou [15], což je spojnice dvou bodů na kouli. To znamená, že tento vzorec nám nedává ten nejpřesnější možný výsledek, a to dokonce z více důvodů. Tím prvním problémem je, že Země není kulatá. Tento algoritmus tedy nepracuje s přesným poloměrem Země podle toho, kde se zrovna nalézáme. Pracuje totiž s takzvaným průměrným poloměrem Země, který je přibližně 6371 km. Další překážkou je, že tento algoritmus na naší trase nepočítá s nerovným terénem, který samozřejmě délku trasy také mění [14].

Výhodou však je, že je nenáročný na výkon (alespoň vzhledem k algoritmu, který by každý metr počítal s jinou nadmořskou výškou) a že jeho chybovost se v průměru pohybuje pod 0,5 %.

5.4 Api.mapy.cz

Po nastudování, jak se souřadnicemi WGS84 pracovat, je stačí získat na základě adresy zákazníka, která je uložená v databázi sytému Helios. Společnost Seznam.cz umožnila komukoliv práci s jejich mapami. Jedná se o online JavaScriptovou knihovnu, kterou si lze do svého HTML souboru přidat pomocí následujících dvou řádků.

```
<script type="text/javascript"
src="https://api.mapy.cz/loader.js"></script>
<script type="text/javascript">Loader.load();</script>
```

Zdrojový kód 8: Přidání online API od společnosti Seznam.cz

5.4.1 Získání souřadnic – geokódování

Geokódování znamená získání souřadnic (v tomto případě WGS84) ze zadané adresy. Zavolám konstruktor třídy Geocoder a předám mu prostý řetězec, který obsahuje adresu. V následující ukázce kódu se jedná o proměnnou add.

```
var geocoder = new SMap.Geocoder(add, function() {
     var vysledky = geocoder.getResults()[0].results;
});
```

Zdrojový kód 9: Práce s třídou SMap.Geocoder

Nyní proměnná vysledky obsahuje všechny výsledky pro danou adresu. Bohužel je možné, že pro některou adresu, která specifikuje dané místo dokonale a na první pohled není pochyb o tom, kde adresa leží na mapě, vrací výsledků více. Moje aplikace v případě vícenásobného výsledku (ať už zadáním chybné, nepřesné adresy nebo nastáním problému, který jsem popsal výše) umožňuje uživateli vybrat přehledně z mapy, které místo je to pravé.

5.4.2 Získání adresy – reverzní geokódování

Reverzní geokódování neboli získání adresy z WGS84 souřadnic tento problém nemá. Navrátí vždy jeden výsledek (třeba jenom název města nebo kraje, pokud je to bod uprostřed pole).

```
var coords = SMap.Coords.fromWGS84("14.41790", "50.12655");
var geocoder = new SMap.Geocoder.Reverse(coords, function()
{         var results = geocoder.getResults();      });
```

Zdrojový kód 10: Práce s reverzním geokódováním

6 Helios Extender – JavaScript

Abych mohl po stisknutí tlačítka *Služební cesta* naplánovat trasu mezi vybranými zákazníky, zobrazit lokaci všech klientů dané firmy na mapě, nebo i jenom získat souřadnice z adres, potřebuji, aby moje aplikace napsaná v C# dokázala komunikovat s API od společnosti Seznam.cz napsaném v jazyce JavaScript.

6.1 WebSocket

WebSocket je pokročilá technologie, která umožňuje otevřít interaktivní obousměrné spojení mezi uživatelským prohlížečem a serverem. S tímto protokolem je možné posílat a přijímat zprávy od serveru událostmi řízenou komunikací. To vše bez nutnosti obnovovat stránku prohlížeče nebo žádání serveru o odpověď.

6.1.1 Server C#

Jako komunikační server jsem si vybral knihovnu websocket-sharp dostupnou na GitHubu pod licencí „The MTI licence“ [7].

6.1.2 Instalace

Přidání této knihovny do svého projektu ve Visual Studiu není nic složitého. Stačí si otevřít NuGet Package Manager a napsat do něj tento příkaz:

```
PM> Install-Package WebSocketSharp -Pre
```

Zdrojový kód 11: Instalace WebSocket serveru

Následně stačí u tříd, kde se jej chystám použít, vložit:

```
using WebSocketSharp;
using WebSocketSharp.Server;
```

Zdrojový kód 12: Přidání reference na WebSocketSharp

6.1.3 Inicializace serveru C#

Vytvoření serveru spočívá v zavolání konstruktoru třídy WebSocketServer a předání komunikačního portu.

```
public WebSocketServer wssv;
public WebSocket()
{
      wssv = new WebSocketServer(9000);
      wssv.AddWebSocketService<getWGS84>("/getWGS84");
      wssv.AddWebSocketService<selectedWGS84>("/selectedWGS8
      4");
      wssv.AddWebSocketService<showAll>("/showAll");
      wssv.AddWebSocketService<makeRoute>("/makeRoute");
}
```

Zdrojový kód 13: Inicializace serveru WebSocketSharp

Následně se přidávají tzv. WebSocketService, což jsou jednotlivé názvy tříd, které dědí od třídy WebSocketBehavior a mají možnost implementovat čtyři události: OnOpen, OnMessage, OnClose, OnError.

OnOpen se vyvolá pouze jednou, když spojení bylo úspěšně navázáno a můžou se posílat data, OnMessage pokud přijde zpráva, OnError pokud nastala nějaká chybová událost a OnClose pokud spojení bylo ukončeno.

```
public class getWGS84 : WebSocketBehavior{
      protected override void OnMessage(MessageEventArgs e){
            MessageBox.Show(e.Data.ToString())
      }
      protected override void OnOpen(){
            Send("Hello world“);
      }
}
```

Zdrojový kód 14: Ukázka implementace třídy dědící od WebSocketBehavior

6.1.4 Klient JavaScript

Pro použití komunikačního protokolu WebSocket v jazyce JavaScript nemusím nic instalovat či nastavovat. Stačí následující ukázka kódu (viz Zdrojový kód 15) a jsem připraven komunikovat se serverem [7].

Ve většině internetového dění se slovo localhost rovná adrese 127.0.0.1, avšak pokud chci daný JavaScript spustit v prohlížeči Internet Explorer musím jako URL napsat přímo adresu 127.0.0.1. Jinak tento kód nebude fungovat z toho důvodu, že Internet Explorer vyvolá chybu SecurityError.

```
window.WebSocket = window.WebSocket || window.MozWebSocket;

websocket = new WebSocket('ws://127.0.0.1:9000/showAll');

websocket.onmessage = function (message)

{

     alert(message.data);

}
```

Zdrojový kód 15: Ukázka implementace klienta protokolu WebSocket v jazyce JavaScript

6.2 Singleton C#

Jelikož je WebSocket událostmi řízená služba, není jednoduché zpracovávat přijatá data a připravovat data k odeslání. Rozhodl jsem se tedy určitá data implementovat za pomocí návrhového vzoru Singleton, který mi umožní mít libovolné informace uložené jako globální proměnné. To znamená, že z libovolného místa ve své aplikaci mohu přistupovat a zapisovat do jakékoliv datové struktury.

6.2.1 Vytvoření návrhového vzoru Singleton

Abych si nemusel předávat jednu instanci objektu mezi několika konstruktory, vytvořím si novou veřejnou třídu, která obsahuje následující prvky:

- Privátní konstruktor, který nemá žádnou funkcionalitu
- Neveřejnou proměnnou, která bude sloužit jako uložiště instance. Musí být tedy stejného datového typu jako třída

- Veřejný „getter“, který zkontroluje, zdali instance neodkazuje na null. Pokud ano, zavolá konstruktor, uloží instanci a navrátí ji. Pokud ne, instance je již vytvořena a může ji pouze navrátit
- Vlastní veřejné proměnné libovolných datových typů a nejrůznějších kolekcí

```
public Process process { get; set; }
private static SingletonCMD _instance;
private SingletonCMD(){}
public static SingletonCMD Instance{
     get{
          if (_instance == null)
          _instance = new SingletonCMD();
          return _instance;
}}
```

Zdrojový kód 16: Použití návrhového vzoru Singleton pro uložení informace po celou dobu běhu aplikace

6.3 Spuštění http-serveru pro práci s JavaScriptovými soubory

Pro práci s programovacím jazykem JavaScript je nutné si opatřit server. Tuto nezbytnost si moje aplikace napsaná v jazyce C# spouští, řídí i ukončuje sama. Dociluje toho tak, že po spuštění programu spustí skrytou příkazovou řádku CMD a napíše příkaz pro spuštění HTTP serveru napsaném v node.js.

6.3.1 Node.js

Node.js je systém napsaný v JavaScriptu postavený na V8 JavaScriptovém základu od Googlu [4]. Jeho balíčkovací systém NPM je největším systémem open source knihoven na světě [5].

6.3.2 Instalace

Instalace samotného Node.js není žádný problém. Z oficiálních stránek si stačí zdarma stáhnout instalační program a řídit se zobrazovanými instrukcemi. Je důležité si dát pozor na to, aby se node.js zaregistroval do systémové proměnné PATH.

Pro spuštění mé aplikace je ještě vyžadován NPM balíček s názvem http-server. Ten lze získat snadno otevřením CMD a zadáním následujícího příkazu:

```
npm install http-server -g.
```

Zdrojový kód 17: Instalace NPM balíčku http-server

6.3.3 Použití

O použití se nemusí uživatel starat, vše probíhá automaticky přes příkazovou řádku po spuštění HeliosExtender.exe. Aplikace totiž sama vyvolá nový proces cmd.exe, který nejprve přejde to složky „server“ a následně vykoná příkaz:

```
http-server -p 45222,
```

Zdrojový kód 18: Spuštění HTTP serveru na portu 45222

kde http-server je název příkazu. Za ním může následovat cesta, odkud má být server hostován, ale jelikož jsem se již přesunul do požadovaného umístění s HTML soubory, je za příkazem ihned parametr -p, který nastavuje číslo portu, v tomto případě 45222, který je již bezpečně mimo kategorii registrovaných portů.

```
Process process = new Process();
ProcessStartInfo startInfo = new ProcessStartInfo();
startInfo.WindowStyle = ProcessWindowStyle.Hidden;
startInfo.FileName = "cmd.exe";
startInfo.Arguments = "/C cd server & http-server -p
45222";
process.StartInfo = startInfo;
process.Start();
SingletonCMD.Instance.process = process;
```

Zdrojový kód 19: Spuštění příkazové řádky, ze které nastartuji HTTP server

6.3.4 Příkazová řádka systému Windows

Aplikace má za úkol zajistit, aby se proces „cmd.exe“, ale i jeho potomci, po ukončení aplikace ukončili a nezůstali tak mezi spuštěnými procesy sytému Windows. Proto si aplikace po vytvoření procesu a nastartování HTTP serveru uloží ID vytvořeného procesu, aby při ukončování aplikace mohla díky tomuto ID najít proces samotný, ale i jeho potomky.

```
try {
Process pcs =
Process.GetProcessById(SingletonCMD.Instance.process.Id);
IEnumerable<Process> list = GetChildProcesses
(Process.GetProcessById(SingletonCMD.Instance.process.Id));
foreach (Process p in list) {
     KillProcessByID(p.Id, true);
}
KillProcessByID(SingletonCMD.Instance.process.Id, true);
}
```

Zdrojový kód 20: Ukončení příkazové řádky a všech potomků

6.4 getWGS84.html

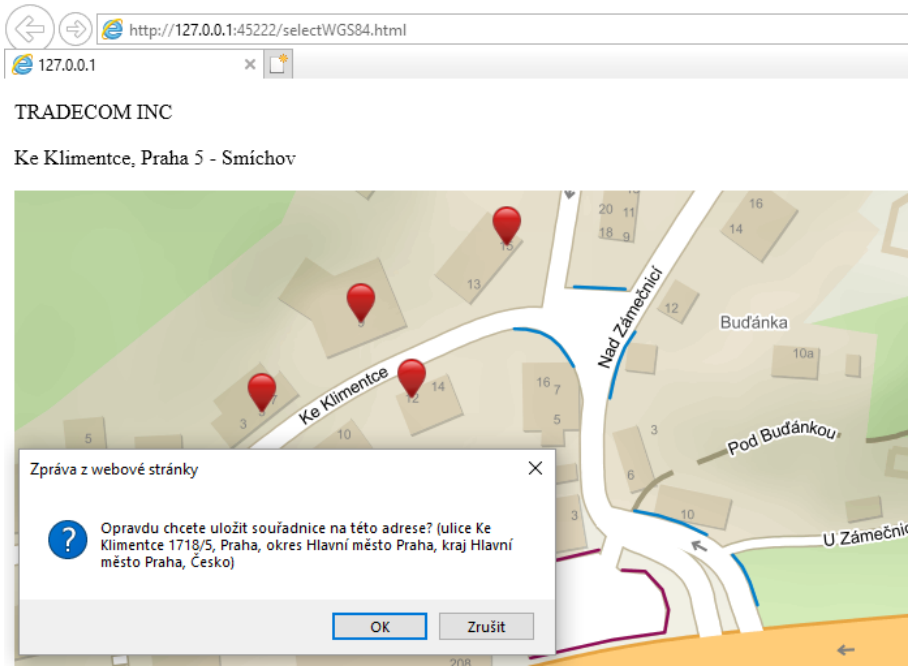
Skript napsaný v souboru getWGS84.html slouží k nalezení souřadnic WGS84 z adres, které mu jsou doručeny přes WebSocket. Pro každou adresu zavolá SMap.Geocoder a všechny výsledky, které na dané adrese dostane (může jich být nula a více), si připraví k odeslání zpět do C# aplikace. Až poté, co projde všechny požadované adresy, odešle hromadně výsledky.

V aplikaci tento soubor otevírám při každém startu, pokud existuje zákazník, který nemá v databázi vyplněné souřadnice. Pokud se stane, že se vrátí více výsledků k jednomu klientovi, zavolám soubor selectWGS84.html

6.5 selectWGS84.html

Při volání skriptu selectWGS84.html se neposílá přes WebSocket pouze adresa zákazníka, jejíž výsledky po použití SMap.Geocoder chci zobrazit na mapě, aby si uživatel mohl vybrat o jaké místo se konkrétně jedná. Posílá se také název firmy, která má tu nejednoznačnou adresu, aby uživatel přehledně vše viděl v okně aplikace Internet Explorer.

Oproti předešlému souboru, tento skript pracuje navíc s vrstvou se značkami. Každá značka má své souřadnice, svoje unikátní ID, uloženou adresu v textové podobě a dále navázanou událost, co má dělat, pokud na ni někdo klikne. V tom případě vyskočí dialogové okno, kde se uživatel dozví přesnou adresu značky, na kterou kliknul, a může se rozhodnout, zdali je to opravdu to místo, kde se firma nalézá.



Obrázek 7: Uživatel vybírá, kde se přesně nachází firma TRADECOM INC

Až po kliknutí na tlačítko *OK* se odešlou příslušné souřadnice zpátky do C# aplikace a okno internetového prohlížeče se automaticky uzavře.

6.6 showAllWGS84.html

Skript jménem showAllWGS84.html se z mé C# aplikace volá v případě, že uživatel vybral určité klienty (nebo všechny) a stiskl tlačítko *Zobrazit vše na mapě*. Do tohoto skriptu už se neposílají adresy, nýbrž konkrétní WGS84 souřadnice. S tím souvisí, že tento skript jako jediný používá reverzní geokódování, aby mohl jednotlivým značkám na mapě přidat popisek v podobě té nejdetailnější adresy, jaká je k dispozici.

Uživatel tedy dostane přehled o tom, v jakých koutech republiky má firma nejvíce zákazníků, kde je největší prodej konkrétního druhu zboží a podobně. Po kliknutí na značku se zobrazí název firmy a adresa, která se nachází na uložených souřadnicích.

6.6.1 Shlukování bodů

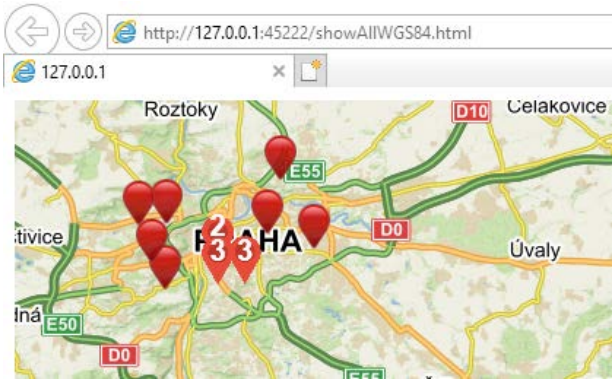
Aby nenastala situace, že například v Praze mám přes 1000 zákazníků, a tedy by se můj skript snažil na mapě v okolí Prahy vykreslit 1000 značek, implementoval jsem následující třídu.

```
SMap.Marker.Clusterer(m, 15)
```

Zdrojový kód 21: Konstruktor třídy SMap.Marker.Clusterer

Jedná se o vrstvu, která se stará o shlukování bodů, kde m je odkaz na instanci objektu SMap neboli mapy jako takové, a číslo 15 představuje velikost jednotlivých oblastí shluku, do kterých následně spadají různé body na mapě. Dokumentace bohužel neobsahuje informaci, o jaké jednotky se jedná.

Naneštěstí dokumentace nedisponuje ani informací o tom, jak si upravit styl dané značky – daného shlukovače. Naštěstí na fórum k API se nachází příspěvek, který se o danou problematikou zajímá [19]. Do toho příspěvku jeden z vývojářů od Seznamu umístil krátkou ukázku, jak se dá čistě za pomocí CSS upravit vzhled shlukovače.



Obrázek 8: Ukázka shlukování bodů v Praze [28][34]

6.7 makeRoute.html

API od společnosti Seznam.cz umožňuje dva způsoby plánování trasy. První pracuje přímo v aplikaci, kterou si programátor vytvoří, tak jako geokódování a reverzní geokódování. Neumožňuje však plynulý pohyb průjezdních míst, je vcelku nepřehledné, má složitou práci s itinerářem a o spojení se svým účtem a možnosti uložení trasy ani nemluvě. Proto jsem se rozhodl pro alternativní možnost. API umí přímo vytvořit URL na stránky mapy.cz přímo s vaší naplánovanou trasou.

6.7.1 Práce s SMap.URL.Route

Čtvrtým a zároveň posledním souborem obsahující JavaScript, který jsem naprogramoval, je makeRoute.html. I do tohoto skriptu přichází WGS84 souřadnice přes

protokol WebSocket, avšak zde již není potřeba žádné geokódování. V tomto souboru se pracuje s následující třídou.

```
SMap.URL.Route();
```

Zdrojový kód 22: Konstruktor třídy SMap.URL.Route

To je objekt, který umí vytvořit URL na základě počátečního, cílového a libovolného množství průjezdních bodů (zadaných v souřadnicích WGS84). Ty stačí otevřít ve webovém prohlížeči a mám naplánovanou cestu, a k tomu dostupné všechny další služby od společnosti Seznam.cz jako různé podklady map, panorama, baťůžek, itinerář, a mnohé další.

```
var smapUrl = new SMap.URL.Route();
var start = SMap.Coords.fromWGS84("14.41790","50.12655");
smapUrl.addStart(start);
smapUrl.addDestination(start);
adresy.forEach(function(element)
{
      prujezd = SMap.Coords.fromWGS84("jiné","jiné");
      smapUrl.addWaypoint(prujezd);
});
var url = smapUrl.toString();
```

Zdrojový kód 23: Ukázka vytvoření URL pro získání naplánované trasy na web mapy.cz

6.8 Knihovna Internet Explorer

Když už mám nastartovaný HTTP server a mám naprogramované JavaScriptové soubory, které umí pracovat s mapami od Seznam.cz a mám s nimi zajištěnou komunikaci přes WebSocket, musím už pouze zajistit, abych ty soubory mohl otevřít, a měl způsob, jak je prezentovat uživateli. K tomu jsem si zvolil následující knihovnu.

```
SHDocVw.InternetExplorer.
```

Zdrojový kód 24: Knihovna pracující s internetovým prohlížečem Internet Explorer

Jedná se o integrovanou třídu v jazyce C#, která umožňuje spustit Internet Explorer a kompletně jej kontrolovat. Dovolí například posouvat v historii prohlížení dopředu a dozadu, přesměrovat uživatele na domovskou stránku, obnovit aktuální URL nebo navázat události například na začátek nebo dokončení stahování, vytvoření nového okna, změnu velikosti okna a mnohé další.

Pro mne je důležité, že mám kontrolu nad tím, kdy se okno zavře a že jsem získal možnost si nastavit, zdali okno prohlížeče bude viditelné nebo ne. Například pro dotázání souboru getWGS84.html není vůbec důležité vidět okno prohlížeče. Je důležité spustit JavaScriptový soubor, který vykoná požadovanou práci, odešle přes WebSocket výsledky a skončí.

```
InternetExplorer IE = new SHDocVw.InternetExplorer();
IE.BeforeNavigate2 += new
SHDocVw.DWebBrowserEvents2_BeforeNavigate2EventHandler(
e.OnBeforeNavigate2);
IE.Visible = false;
IE.Navigate2(ref URL, ref Empty, ref Empty, ref Empty, ref
Empty);
```

Zdrojový kód 25: Vytvoření objektu InternetExplorer a vyvolání prohlížeče na webové adrese uložené v proměnné URL

7 Naplánování služební cesty

Jednotlivé průjezdné body služební cesty nemohu jen vzít a poslat do souboru makeRoute.html, který naplánuje trasu. Mohlo by se jednoduše stát, že by trasa vedla z Liberce do Prahy, pak zpátky do Jablonce, pak do Karlových Varů a poté zpátky do Liberce. Jedná se o známý problém obchodního cestujícího, kdy hledám nejkratší možnou cestu mezi všemi body na mapě.

Tyto průjezdné body je nutné uspořádat a existuje pro to několik řešení. Ta se dělí do dvou skupin: přesná a heuristická. Přesná řešení po dokončení poskytnou to nejlepší možné řešení, ale zato jejich náročnost na výpočet a paměť je neúnosná už od dvoumístného počtu průjezdných bodů (záleží na použitém algoritmu). Zatímco heuristická nám nemusí ve všech případech dát ideální řešení, ale jsou nenáročné na procesor a paměť a lze s nimi spočítat větší množství bodů na trase.

Velice často se také setkáme s použitím velmi jednoduchého heuristického řešení pro výpočet základního odhadu celkové délky trasy, na které navazuje jedno z přesných řešení. To má již od začátku orientační vzdálenost, kterou pokud nějaká cesta přesáhne, nemá smysl s ní dále počítat.

7.1 Algoritmus nearest neighbour

Heuristický algoritmus „nejbližšího souseda“ byl jedním z prvních vůbec, který se zabýval problematikou obchodního cestujícího. Je neuvěřitelně rychlý, nezatěžuje paměť ani procesor, ale jeho výsledek je velmi často vzdálen od ideálního řešení. I tak jako způsob získání přibližného odhadu slouží dobře, poněvadž zase nemá sklony k tomu, aby nalézal úplně nesmyslné cesty [16].

7.1.1 Princip

Začínám na startovací adrese, kterou si označím jako aktuální a projdu všechny cesty, které z ní vedou (o jeden méně průchodů, než je celkové množství adres). Vyberu tu s nejmenší vzdáleností a uložím si ji jako aktuální. Zároveň smažu nebo označím adresu jako již navštívenou. Poté si znovu projdu všechny cesty, které z nového bodu vedou (tentokrát o dvě méně než celkový počet adres) a tak dále, dokud neprojdu všechny body.

7.2 Algoritmus brute force

Přesný algoritmus „hrubá síla“ je pravděpodobně tím nejznámějším a nejuniverzálnějším algoritmem vůbec. Avšak neznám případ, kdy by se jeho implementace doporučovala či byla nějakým způsobem efektivní. Jeho základní vlastností je opravdu velká náročnost na výpočetní výkon, již při nízkém množství vstupních dat. Na druhou stranu jeho velmi přívětivou vlastností je 100 % úspěšnost nalezení toho nejlepší řešení.

Bohužel problém obchodního cestujícího patří do skupiny takzvaných NP náročných úloh. To znamená, že nemáme algoritmus, který by nám přinesl řešení pro všechny sady vstupních dat o libovolném množství za rozumný čas. A to je důvod, proč se tímto algoritmem brute force vůbec zabývám.

7.2.1 Princip

Tento algoritmus má spoustu různých menších vylepšení implementace vzhledem k danému problému, kde se používá. Ale základ je vždy následující. Snažíme se projít úplně všechna možná řešení, což v mém případě je počet iterací algoritmem roven:

$$x = \frac{(a-1)!}{2}$$

Rovnice 1: Počet iterací algoritmem brute force bez dalších vylepšeních

Kde *x* symbolizuje výsledný maximální počet iterací a *a* počet průjezdných bodů vstupujících do algoritmu. Již při jedenácti adresách se blížím až ke dvěma miliónům iterací.

7.2.2 Konkrétní implementace

Jako první si na zažádanou sadu adres spustím algoritmus nearest neighbour a vzdálenost, kterou si spočítal si uložím jako nejlepší dosavadní řešení. Následně ze startovací adresy projdu všechny možnosti, kam se mohu vydat. Pokud celková cesta do nového místa není již delší než dosavadní nejlepší možné řešení a pokud jsem ještě neprošel všechny body, tak si každou z těchto nových cest uložím do zásobníku. Přiřadím ji také informaci o tom, kudy jsem se do toho bodu dostal (v prvním průchodu algoritmem

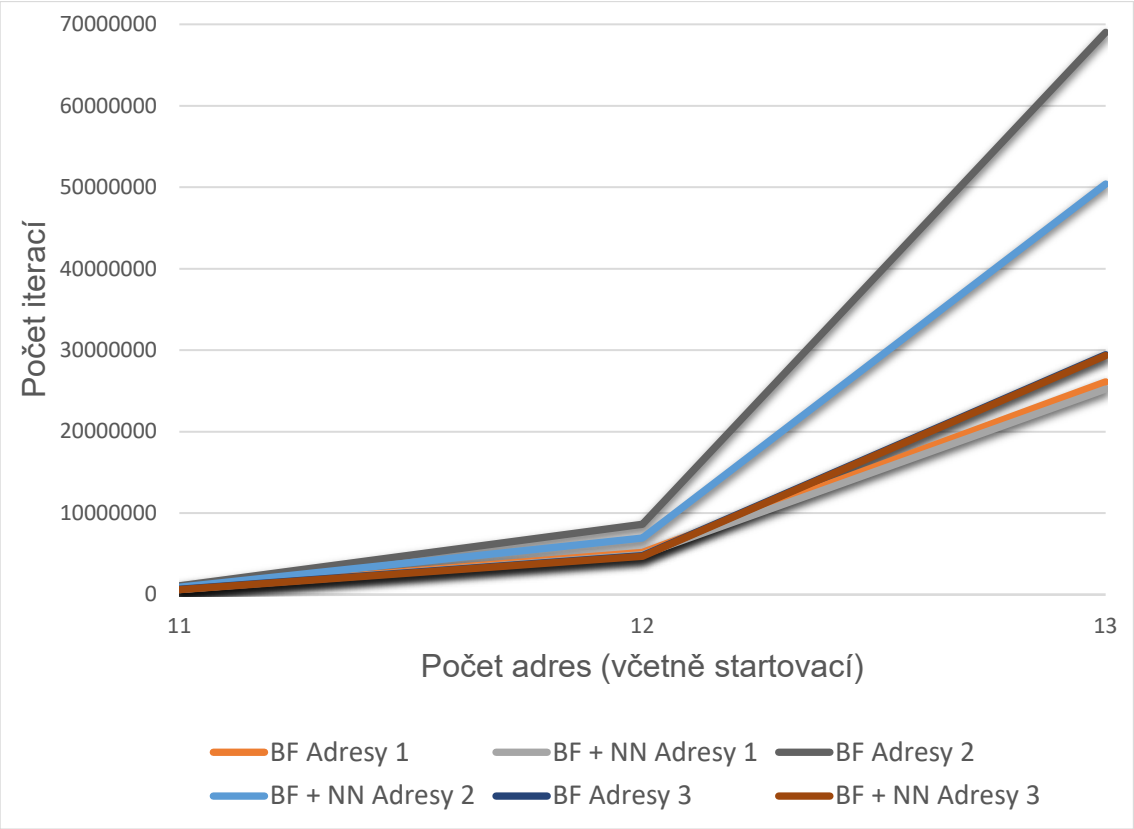
ze startovací adresy). A takto tento cyklus opakuji, dokud se nacházejí v zásobníku nějaké cesty podezřelé z toho, že by jimi mohla vést nejkratší cesta mezi všemi adresami.

Pokud celková vzdálenost je již větší než nejlepší řešení, co zatím mám, vím jistě, že tato cesta už nemá možnost se stát tou ideální, a proto informaci o ni smažu. Druhou možností je, že cesta je již v cíli a jedná se o nejkratší řešení, pak si ji uložím jako nejlepší dosavadní cestu, s kterou budu porovnávat všechny nové podezřelé cesty. Zároveň projdu všechny již uložené cesty a zkontroluji, zdali mají uraženou vzdálenost stále menší i jak nové zjištěné nejlepší řešení.

7.2.3 Výsledky

Následující graf ukazuje počet iterací v závislosti na počtu adres při použití algoritmů brute force s a bez použití algoritmu nearest neighbor pro vytvoření odhadu pro tři různé adresové sady. Je patrné, že sadě číslo dvě při výpočtu předem získaný odhad pomohl. Ušetřil ve výsledku dokonce až dvacet milionů iterací, zatímco u zbylých dvou sad adres prakticky neměl žádný efekt.

Tyto ušetřené iterace také ušetřily při výpočtu dvacet vteřin. I tak bohužel výpočet trvá stále vysokých 58 vteřin při sadě o velikosti 13 adres.



Graf 1: Počet iterací algoritmem v závislosti na počtu adres, BF = brute force, NN = nearest neighbor

7.3 Algoritmus branch and bound

Druhým přesným algoritmem, který jsem implementoval, je algoritmus „větví a hranic“ nebo „větví a mezí“. I tato metoda se používá pro řešení NP náročných úloh a je postavena na již zmíněném algoritmu brute force. S tím souvisí, že v nejhorším případě tento algoritmus může mít stejnou efektivitu. I zde platí 100 % úspěšnost nalezení ideálního řešení, a i tento algoritmus rád využívá systémové prostředky ve velkém, ale to je bohužel u řešení NP obtížných úloh běžné. K rozhodování, jakou cestou se vydat dále používám prioritní binární haldu.

7.3.1 Prioritní binární halda

Binární halda je taková struktura, kde rodič má uložené odkazy na maximálně dva potomky, které musí být větší anebo menší, podle toho, jakou prioritu máme. Je nutné tuto datovou strukturu indexovat od čísla 1, aby byly zachovány odkazy na potomky v podobě:

$$potomek1 = index \cdot 2, \qquad potomek2 = (index \cdot 2) + 1$$

Rovnice 2: Výpočet odkazu na dva potomky v binární haldě

A odkaz zpátky na rodiče potom vypadá pro oba potomky následovně:

$$rodič = \frac{potomek12}{2}$$

Rovnice 3: Výpočet odkazu na rodiče v binární haldě

Toto je jeden z mála případů, kdy opravdu chci, aby celočíselné dělení navrátilo celočíselný výsledek. Je důležité, aby proměnná *rodič* byla datového typu integer. Snažit se přistoupit na desetinný index kolekce by určitě nedopadlo dobře.

Nový prvek do této struktury přidám tak, že jej umístím na poslední index binární haldy a následně provedu takzvané probublání směrem nahoru. To spočívá v tom, že se u nově přidaného prvku podívám, jestli není menší, jak jeho rodič. Pokud ano, tak je prohodím a takto pokračuji, dokud nacházím větší rodiče anebo dokud nedosáhnu na vrchol haldy.

Velice častou operací je požadavek na získání nejmenšího prvku. To je vlastně důvod, proč jsem binární haldu implementoval, protože složitost na vrácení nejmenší hodnoty je $O(1)$ a k tomu musím opravit haldu s náročností $O(\log_2 n)$. To se provádí tak, že poslední prvek se přesune na vrchol haldy (na index 1) a probublá směrem dolů následujícím způsobem. Podívám se na potomky nově přidaného vrcholu, prohodím ho s tím menším ze dvou potomků a takhle to pokračuje, dokud nacházím menší hodnoty nebo dokud nenarazím na konec haldy.

Další operací nad daty v haldě, kterou používám, je smazání prvku na určitém indexu. Postup je velice podobný jako u navrácení nejmenšího prvku s tím rozdílem, že zde daný prvek nevracím, a že se neprobublává od vrcholu haldy, nýbrž od indexu mazaného prvku.

7.3.2 Princip algoritmu branch and bound

Před začátkem algoritmu si musím vyplnit matici vzdáleností. Jedná se o čtvercovou matici o velikosti závisející na počtu adres vstupujících do algoritmu. Na

diagonále se nastaví nekonečno (cesta z bodu A do dobu A, z bodu B do bodu B atd. nedává smysl). Nyní si v této matici na každém řádku naleznu minimum, a to z příslušného řádku odečtu. Jako další naleznu minimální hodnoty ve sloupcích a ty opět odečtu a zároveň si všechny odčítané hodnoty (*n* počet řádků a *n* počet sloupců, kdy *n* je počet adres) posčítám a uložím do proměnné. Toto číslo představuje přibližný počet kilometrů, jaký bude cesta obsahovat. Toto číslo možná už nebude větší, ale určitě už nebude nižší, takže pokud to opět porovnám s nejlepším dosavadním výsledkem, je tu možnost, že ho nebudu vůbec ukládat. Avšak pokud je menší, do objektu si uložím upravenou matici vzdáleností, uraženou vzdálenost, kolikátý bod na cestě to je a kudy jsem se do něj dostal (zde pro počáteční stanici nebude nic uloženého). Tento objekt vložím do prioritní binární haldy.

Bod, z kterého budu cestovat, získám tak, že si vezmu vrchol haldy a z tohoto bodu se vydám všemi možnými cestami, které dávají smysl (v prvním kroku algoritmu ze startovací adresy). Načtu si matici vzdáleností z adresy, ze které jsem se do aktuálního bodu dostal, a nastavím ji jeden řádek a jeden sloupec na nekonečno. O jaký se jedná záleží na tom, z jakého bodu a kam jdu. Pokud jedu například z bodu 1 do bodu 2 nastavím první řádek vzdálenostní matice a druhý sloupec na nekonečno a k tomu ještě hodnotu na druhém řádku a prvním sloupci na nekonečno (na této cestě už nebudu chtít jít z bodu 2 do bodu 1 – neboli zpátky).

A dále pokračuji, jak už jsem popisoval výše. Najdu a odečtu nejmenší řádky a sloupce a spočítám si uraženou vzdálenost. Tu dostanu tak, že posčítám: všechny odčítané řádky a sloupce, dosavadní uloženou vzdálenost z předchozího bodu a z jeho matice ještě hodnotu na prvním řádku a druhém sloupci. Nyní porovnám tuto hodnotu s nejlepším dosavadním výsledkem a pokud bude vyšší, tato cesta pro mě skončila. Pokud ne, uložím si ji do binární haldy. Takhle pokračuji tak dlouho, dokud mám v binární haldě nějaké adepty na probádání [29][30].

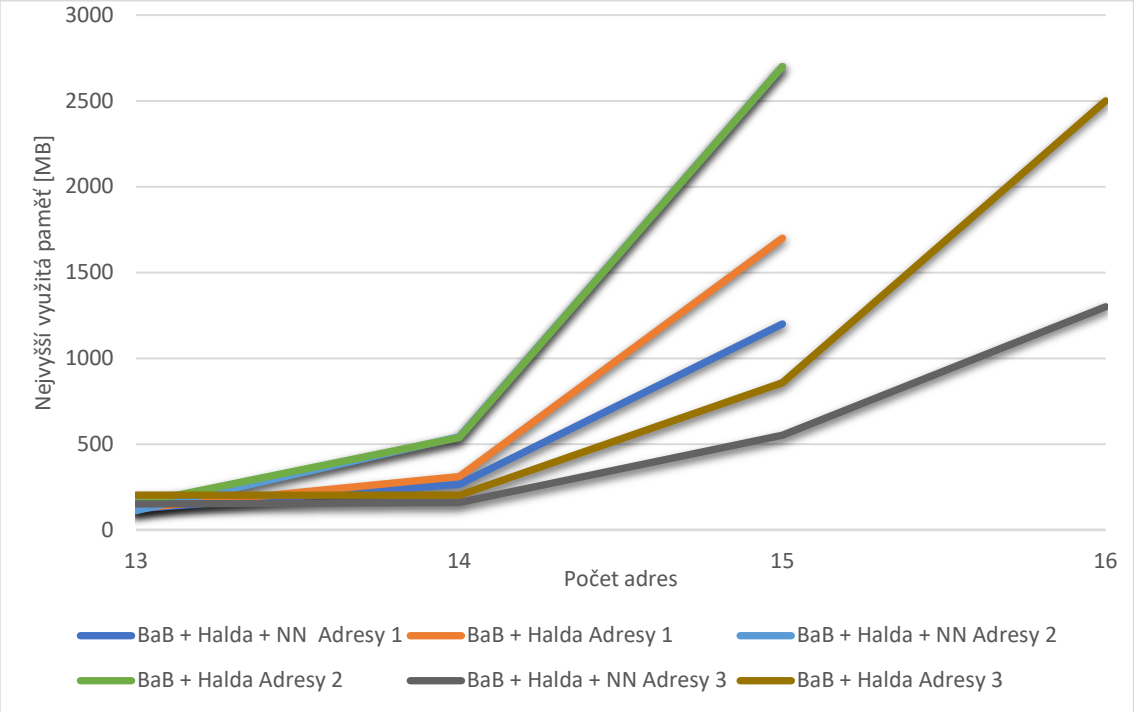
7.3.3 Konkrétní implementace

Před začátkem algoritmu si spustím na danou sadu adres algoritmus nearest neighbour a vzdálenost, kterou mi najde si uložím jako dočasně nejlepší. Pro každý průchod algoritmu si vytvořím novou matici a do ní zkopíruju vzdálenostní matici z přechozího bodu.

Pokud jdu například z adresy 1 do adresy 2, podívám se na druhý řádek a první sloupec, jestli tam není nekonečno. Jestliže ano, znamenalo by to, že se snažím jít cestou někam, kde už jsem byl, a tuto cestu tedy nezkoumám. Pokud ne, tak pokračuji s výpočty. Zkontroluji, jestli už nejsem v cíli nebo jestli mi do něj nechybí už jen jeden bod. Pokud již v cíli jsem, porovnám to s nejlepším dosavadním řešením a jestliže mi chybí do cíle už jen jeden krok, tato cesta předběhne všechny uložené v haldě a půjde na řadu další kolo algoritmu. Čím dříve získám cesty, které vedou až do cíle, tím dříve mohu uvolňovat paměť. Pokud totiž získám nový nejlepší výsledek, projdu haldu a odmažu cesty, které již nemají šanci být ideální. Také jakmile projdu všechny cesty z nějaké adresy, odmažu její vzdálenostní matici, abych uvolnil paměť.

7.3.4 Výsledky

Následující graf ukazuje využití paměti RAM v závislosti na počtu adres při použití algoritmů branch and bound s a bez použití algoritmu nearest neighbour pro vytvoření odhadu. Je zde vidět, že u sady adres číslo 2 heuristický algoritmus nearest neighbour nenalezl dobré řešení, a proto má stejnou náročnost na paměť RAM, a to podstatně vyšší, jak u zbylých dvou sad.



Graf 2: Využití paměti RAM v závislosti na počtu adres, BaB = branch and bound, NN = nearest neighbor

Ačkoliv operační paměť umí ušetřit dobrým způsobem, počtu iterací, tedy času stráveným na procesoru nepomůže, jelikož maže z paměti pouze přebytečné cesty umístěné v haldě hluboko. Nestane se tedy, že by se někdy dostalo na nějakou z těchto cest na řadu, pouze je nemusím mít uložené tak dlouho.

Náročnost na paměť je tak vysoká z důvodu ukládání maticí vzdálenosti. Pro každý bod, který ještě není v cíli musím mít uloženou obrovskou matici, například pro 16 adres to bude matice o 256 prvcích, plná čísel typu double

7.4 Shrnutí

Zatímco u brute force vypočítání odhadovaného výsledku za pomocí nearest neighbour nepomohlo v žádné oblasti, u branch and bound algoritmu to dokázalo ušetřit až 500 MB paměti. Obecně díky jeho implementaci jsem schopen naplánovat služební cestu za rozumnou dobu mezi 13 adresami, namísto 11 při použití brute force.

Další výhodou implementace branch and bound je nízké množství iterací algoritmem. Bohužel tyto iterace jsou na druhou stranu velice složité. Pro představu jeden průchod algoritmem je přibližně 20krát časově náročnější než u brute force.

8 Závěr

Zprvu v mé bakalářské práci představuji informační systém Helios. Popisuji, k čemu takový systém slouží, jak s ním a SQL serverem pracuji a jaká data od něj, z jeho 2000 tabulek, potřebuji. Dále představuji moji aplikaci naprogramovanou v jazyce C#. S tím souvisí seznámení s návrhovým vzorem „Model, View, ViewModel“, ve kterém jsem ji napsal, s grafickým rozhraním WPF, které jsem zvolil pro vytvoření uživatelského rozhraní a s metodami jako „binding“ a „commands“, které jsou nedílnou součástí práce s WPF.

V navazující kapitole se věnuji filtrování dat. Představuji mnou navržené předvolby výběru filtrů a možnost napsat si vlastní filtr podle potřeb uživatele, s čímž souvisí kompilace uživatelského kódu za běhu programu.

Jako další popisuji programování v JavaScriptu s API od společnosti Seznam.cz. Ukazuji, jakým způsobem za pomocí třídy geokódování si získat souřadnice na základě uložené adresy, jak tyto souřadnice převádět do podoby vhodné pro výpočty, nebo jak spočítat vzdálenost mezi dvěma body v těchto souřadnicích. Dále jak zobrazit adresy, které si uživatel označí, jako body na mapě pro vytvoření přehledu, kde firma nejvíce prosperuje. Nebo jak vytvořit URL, za pomocí třídy SMap.Route přidáváním jednotlivých adres na služební cestě, s naplánovanou trasou na web mapy.cz.

Následně představuju vytvoření HTTP serveru, k čemuž mi pomáhá systém node.js, konkrétně jeden NPM balíček s názvem http-server. O zajištění událostmi řízené komunikace mezi aplikací napsanou v jazyce C# a zmíněnými JavaScriptovými soubory se mi stará protokol WebSocket. V poslední řadě se zabývám optimalizací návrhu služební cesty tak, aby cesta byla co nejkratší. K tomu používám přesné algoritmy branch and bound, brute force, heuristický algoritmus nearest neighbour a jejich kombinace.

Výsledkem mé práce bylo zjištění, jaké množství adres lze použít pro naplánování služební cesty za pomocí zmíněných algoritmů, jaké množství systémových prostředků si nárokují a jak jim v těchto oblastech pomůže výpočet odhadované délky trasy za pomocí algoritmu nearest neighbour.

Do budoucna je možné tuto práci rozšířit o další heuristické algoritmy, buďto pro přesnější výpočet odhadu, nebo přímo pro vytvoření naplánované trasy zobrazené uživateli.

Literatura

[1] In: *Client–server model* [online]. David Vignoni, 2011 [cit. 2018-05-13]. Dostupné z: https://en.wikipedia.org/wiki/Client%E2%80%93server_model

[2] HELIOS Orange / Asseco Solutions – ERP – Informační systém – Lomax. *Youtube* [online]. Bořetice: HELIOS / Asseco Solutions, 2015 [cit. 2018-05-13]. Dostupné z: https://www.youtube.com/watch?v=eEi53QTZrQ0

[3] HELIOS Green / Asseco Solutions – ERP – Informační systém – Seznam.cz. *Youtube* [online]. Praha: HELIOS / Asseco Solutions, 2014 [cit. 2018-05-13]. Dostupné z: https://www.youtube.com/watch?v=y3wsUcthyZc

[4] Chrome V8. Google Developers [online]. California: Google, 2017 [cit. 2018-05-13]. Dostupné z: https://developers.google.com/v8/

[5] NPM [online]. California: Isaac Z. Schlueter, 2017 [cit. 2018-05-13]. Dostupné z: https://www.npmjs.com/

[6] Node.js [online]. Ryan Dahl, 2009 [cit. 2018-05-13]. Dostupné z: https://nodejs.org/

[7] WebSockets. Mozilla Developer Network [online]. California: Mozilla Developer Network members, 2017 [cit. 2018-05-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[8] Websocket-sharp. *GitHub* [online]. STA, 2010 [cit. 2018-05-13]. Dostupné z: https://github.com/sta/websocket-sharp

[9] World Geodetic System. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2017 [cit. 2018-05-13]. Dostupné z: https://cs.wikipedia.org/wiki/World_Geodetic_System

[10] Vzdálenost dvou zeměpisných bodů. *PHP triky* [online]. Jakub Vrána, 2007 [cit. 2018-05-13]. Dostupné z: https://php.vrana.cz/vzdalenost-dvou-zemepisnych-bodu.php

[11] Úvod do WPF. *ITnetwork.cz* [online]. David Čápka [cit. 2018-05-13]. Dostupné z: https://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace/

[12] Convert WGS84(Degrees, Minutes, seconds) To Decimal Degrees [duplicate]. In: *Stack Overflow* [online]. Raika, 2012 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/12871087/convert-wgs84degrees-minutes-seconds-to-decimal-degrees

[13] Jak nezabloudit aneb jak na WGS, S42, JTSK, ... *BERUNA WEB* [online]. Beruna, 2011 [cit. 2018-05-13]. Dostupné z: http://www.beruna.cz/text-jak-nezabloudit-aneb-jak-na-wgs-s42-jtsk/

[14] Stack Overflow. *Calculate distance between two latitude-longitude points? (Haversine formula)* [online]. Chuck, 2014 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula?page=1&tab=votes#tab-top

[15] Ortodroma. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-05-13]. Dostupné z: https://cs.wikipedia.org/wiki/Ortodroma

[16] Nearest neighbour algorithm. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-05-13]. Dostupné z: https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

[17] Branch and bound. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-05-13]. Dostupné z: https://en.wikipedia.org/wiki/Branch_and_bound

[18] Travelling salesman problem. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-05-13]. Dostupné z: https://en.wikipedia.org/wiki/Travelling_salesman_problem

[19] Vlastní styl shlukovače. *Seznam.cz nápověda* [online]. tpialek, 2015 [cit. 2018-05-13]. Dostupné z: https://napoveda.seznam.cz/forum/viewtopic.php?f=31&t=23148&sid=c5c96482dd81bb30eb6a86952bb66fe0

[20] Calculating how long arc second is for given long/lat? [closed]. *Geographic Information Systems* [online]. Robert Buckley, 2017 [cit. 2018-05-13]. Dostupné z:

https://gis.stackexchange.com/questions/24975/calculating-how-long-arc-second-is-for-given-long-lat

[21] How do I add another project (assembly) from my solution to the set of CompilerParameters.ReferencedAssemblies?. *Stack Overflow* [online]. sjs, 2012 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/13424751/how-do-i-add-another-project-assembly-from-my-solution-to-the-set-of-compilerp

[22] Compiling C# Code at Runtime. *CODE PROJECT* [online]. Lumír Kojecký, 2014 [cit. 2018-05-13]. Dostupné z: https://www.codeproject.com/Tips/715891/Compiling-Csharp-Code-at-Runtime

[23] PLÍVA, Z., J. DRÁBKOVÁ, J. KOPRNICKÝ a L. PETRŽÍLKA. Metodika zpracování bakalářských a diplomových prací. 2. upravené vydání. Liberec: Technická univerzita v Liberci, FM, 2014. ISBN 978-80-7494-049-1. Dostupné z: doi:10.15240/tul/002/2014-11-002

[24] Stack Overflow. *Switch case: can I use a range instead of a one number* [online]. Steve G., 2017 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/20147879/switch-case-can-i-use-a-range-instead-of-a-one-number

[25] Binding DataGridComboBoxColumn SelectedIndex. *Stack Overflow* [online]. ravi, 2014 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/14466026/binding-datagridcomboboxcolumn-selectedindex

[26] How to propagate property change notifications of objects within collections. *Stack Overflow* [online]. David Schwartz, 2013 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/20846161/how-to-propagate-property-change-notifications-of-objects-within-collections

[27] SQL update statement in C#. *Stack Overflow* [online]. Habib, 2013 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/15246182/sql-update-statement-in-c-sharp

[28] Map marker Icon vector. In: *Brand EPS* [online]. Paomedia [cit. 2018-05-13]. Dostupné z: https://www.brandeps.com/icon/M/Map-marker-02

[29] Travelling salesman Problem using Branch and Bound Method. In: *YouTube* [online]. Mukesh Rajput, 2017 [cit. 2018-05-13]. Dostupné z: https://www.youtube.com/watch?v=X_CDan9Lqc4

[30] Brian Ray: Traveling Salesman (Branch and Bound in 10 min). In: *YouTube* [online]. Code Fellows, 2015 [cit. 2018-05-13]. Dostupné z: https://www.youtube.com/watch?v=JQW-0d1-Ttw

[31] Check for column name in a SqlDataReader object. *Stack Overflow* [online]. Jasmine, 2015 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/373230/check-for-column-name-in-a-sqldatareader-object

[32] Beginners guide to accessing SQL Server through C#. *Code project* [online]. Matt Newman, 2004 [cit. 2018-05-13]. Dostupné z: https://www.codeproject.com/Articles/4416/Beginners-guide-to-accessing-SQL-Server-through-C

[33] Metoda SqlCommand.ExecuteNonQuery (). *Microsoft Developer Network* [online]. [cit. 2018-05-13]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.data.sqlclient.sqlcommand.executenonquery(v=vs.110).aspx

[34] Center a large image of unknown size inside a smaller div with overflow hidden. *Stack Overflow* [online]. Evan Layman, 2015 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/19673895/center-a-large-image-of-unknown-size-inside-a-smaller-div-with-overflow-hidden

[35] DataGridComboBoxColumn data binding. *Stack Overflow* [online]. Fredrik Hedblad, 2011 [cit. 2018-05-13]. Dostupné z: https://stackoverflow.com/questions/7050286/datagridcomboboxcolumn-data-binding

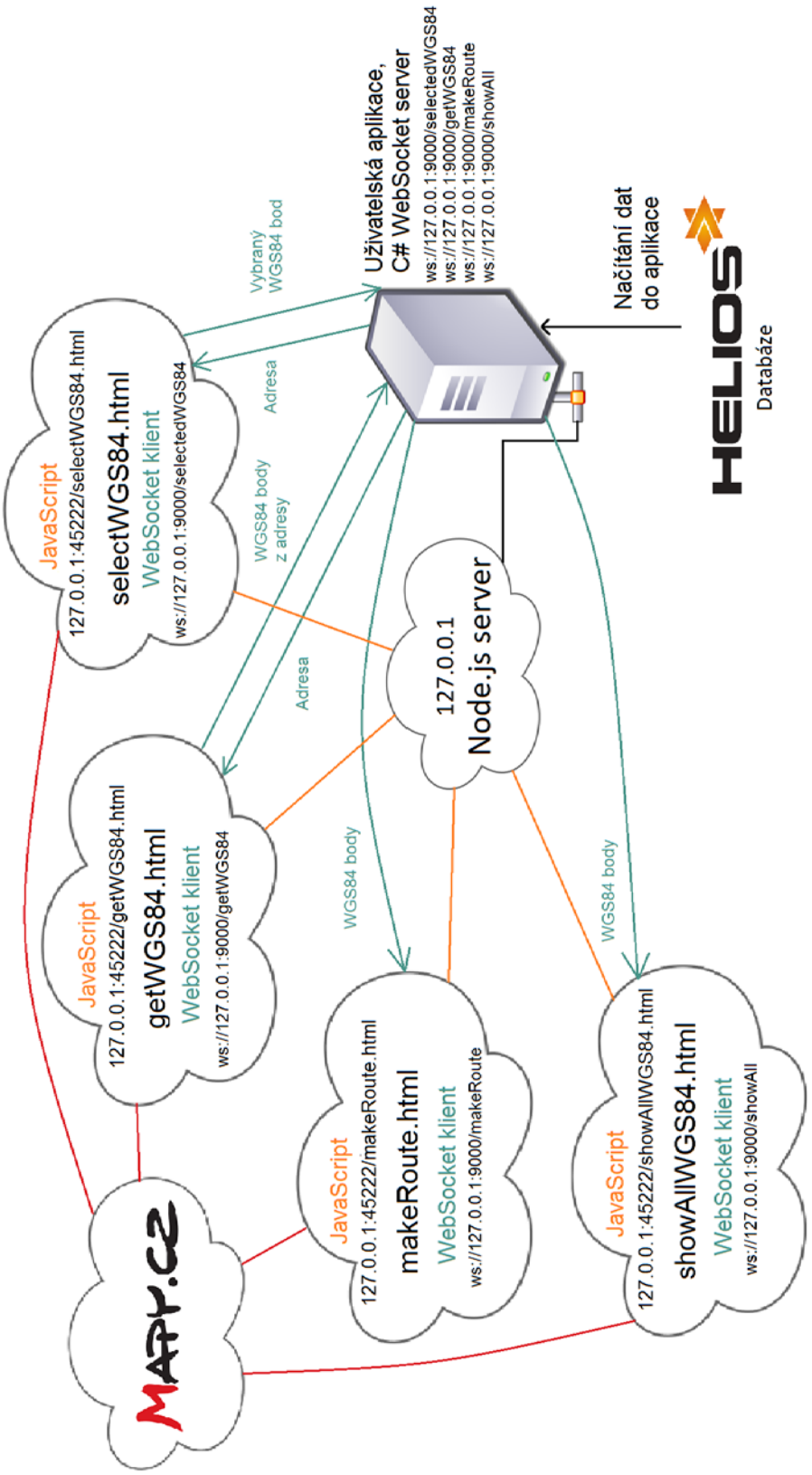
A Uživatelský manuál – jak vytvořit filtr

V mé aplikaci si uživatel může vyfiltrovat zákazníky jakým způsobem uzná za vhodné, jen je nutné se držet následujících pokynů.

- Psát kód, který nevyžaduje importování dalších knihoven nebo definování jmenných prostorů
- Výjimkou jsou kolekce ObservableCollection<T> a List<T>
- Kód musí obsahovat klíčové slovo return, následované návratovou hodnotou následujících datových typů:
    - Kolekce ObservableCollection<Customer> a List<Customer>
    - Samotný objekt Customer
    - Datový typ int, string, double, long, float, decimal, bool, char
- V případě navrácení kolekce nebo samotného objektu Customer, budou tito zákazníci uživateli vypsáni na hlavním okně aplikace
- V případě navrácení hodnoty v primitivním datovém typu výše zmíněném, bude uživateli zobrazeno nové okno aplikace s výsledkem
- Uživatel má přístup ke všem zákazníkům, kteří jsou aktuální na hlavním okně aplikace v podobě ObservableCollection<Customer> customers. Objekt Customer má následující sktrukturu:
    - bool IsChecked, zdali má zákazník zaškrtnutý příslušný CheckBox
    - string Number, unikátní identifikační číslo zákazníka
    - string CompanyName, název firmy
    - int IČO
    - string DIČ
    - double MoneySpent, celková hodnota, kterou zákazník utratil
    - double MoneyReceived, celková hodnota, kterou zákazník obdržel

- List<Ware>, seznam zboží, které si někdy zákazník koupil nebo prodal
  - Objekt Ware obsahuje následující proměnné:
    - int ID, unikátní číslo
    - string Name, název produktu
- Address Address, adresa zákazníka
  - Objekt Address obsahuje následující proměnné:
    - string Street, ulice
    - string City, město
    - string Country, region
- WGS84 Coordinates, souřadnice získané z adresy zákazníka
  - Objekt Coordinates obsahuje následující proměnné
    - double Degrees, stupně
    - double Minutes, minuty
    - double Seconds, vteřiny

B Schéma aplikace



C Obsah přiloženého CD

- text bakalářské práce
  - bakalarska_prace_2018_Tomas_Kosek.pdf
  - bakalarska_prace_2018_Tomas_Kosek.docx
  - kopie_zadani_bakalarska_prace_2018_Tomas_Kosek.pdf
- zdrojový kód programu
  - HeliosExtender (aplikace pro PC v programovacím jazyce C#)
    - server (komunikace s mapy.cz soubory napsané v JavaScriptu)
    - scripts (uložené uživatelem vytvořené filtry)