VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Modul optimalizace přeprav pro dopravní monitorovací systém

Transport Optimisation Module for Traffic Monitoring System

2015 Jiří Friml

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Friml**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Modul optimalizace přeprav pro dopravní monitorovací systém
Transport Optimisation Module for Traffic Monitoring System

Zásady pro vypracování:

Cílem bakalářské práce je vytvoření modulu optimalizace přeprav pro dopravní monitorovací systém vyvíjený společností GX Solutions ve spolupráci s VŠB-TU Ostrava. Tento modul bude určen pro ovládání polo-automatizované optimalizace přeprav, jejíž algoritmus je vyvíjen v rámci projektu RODOS na VŠB-TU Ostrava. V rámci bakalářské práce se musí vytvořit uživatelské rozhraní pro zadávání vstupu do tohoto algoritmu a pro práci s jeho výstupy. Při vytváření bakalářské práce student obdrží základní knihovny pro komunikaci s databází a dalšími existujícími částmi systému.

Jednotlivé body bakalářské práce jsou:
1. Analýza současného stavu problematiky optimalizace přeprav.
2. Implementace databázové vrstvy pro konfiguraci optimalizátoru přeprav a pro uložení výsledků.
3. Vytvoření uživatelského rozhraní pro práci s optimalizátorem přeprav.
4. Testování vytvořeného modulu s využitím reálných dat.

Seznam doporučené odborné literatury:

[1] GX Solutions. Manuál na obsluhu SW Truck Data Memory 2010 v2.2
[2] jsprit, open source toolkit for solving rich vehicle routing problems:
https://github.com/jsprit/jsprit/wiki/More-Examples

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

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

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Chtěl bych na tomto místě poděkovat vedoucímu práce Ing. Janu Martinovičovi, Ph.D. a také zadavateli práce Ing. Marku Boberovi za vedení a za všechny připomínky a cenné rady, které vedly k úspěchu při vytváření této práce a k jejímu zkvalitnění.

Abstrakt

Cílem této práce bylo vytvoření modulu optimalizace přeprav, jehož součástí je rozhraní pro práci s algoritmy knihovny Jsprit a pro zadání vstupů do těchto algoritmů z prostředí frameworku .NET a také práce s mapou OpenLayers – vykreslování bodů a tras do mapy. Knihovnu Jsprit jsem pomocí nástroje IKVM zpracoval tak, aby ji bylo možné použít z prostředí .NET, což se povedlo.

Klíčová slova: Optimalizace, bakalářská práce, knihovna Jsprit, MVC

Abstract

The aim of this thesis was to create transport optimisation module, that consists of interface for working with algorithms from the Jsprit library and for inputing values into these algorithms from .NET framework and also of working with OpenLayers map – drawing points and strokes to the map. I treated the Jsprit library by IKVM tool in order to be able to use it from the .NET environment and I was successful.

Keywords: Optimisation, bacelor thesis, Jsprit library, MVC

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
AJAX	–	Asynchronous Javascript and XML
CSS	–	Cascading Style Sheet
DB	–	Databáze
DBMS	–	DataBase Management System
GUI	–	Graphical User Interface
IDE	–	Integrated Development Environment
JDK	–	Java Development Kit
JSON	–	JavaScript Object Notation
MVC	–	Model-View-Controller
OL	–	OpenLayers
POI	–	Point Of Interest
SOAP	–	Simple Object Access Protocol
TSP	–	Travelling Salesman Problem
URL	–	Unified Resource Locator
VRP	–	Vehicle Routing Problem
WSDL	–	Web Services Description Language
XML	–	eXtensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Optimalizace je jedním ze způsobů, jak výrazně snížit náklady a výdaje za provoz. Když se řekne optimalizace, spoustě lidí se jistě vybaví přepravy a logistika. Oblast logistiky je skutečně oblastí, kde se dá tento postup účelně aplikovat a kde se aplikuje snad nejčastěji. Výdaje za každý jednotlivý kamion nebo nákladní vozidlo, za pohonné hmoty a mýtné na silnicích, za pracovní dobu řidiče (přesčasy) a za další záležitosti spojené s provozem vozového parku jsou pro logistické firmy značné, proto se těmto firmám jistě vyplatí do optimalizace investovat.

Oblast logistiky ale jistě není jedinou oblastí, kam je optimalizace aplikovatelná. Optimalizační úlohy se řeší dnes a denně v mnoha dalších oblastech a odvětvích. Můžeme si například představit továrnu, ve které pracuje stroj, jehož hlava operuje na relativně velkém prostoru a pohyb hlavy je omezený, takže se pohybuje jen velmi pomalu. I zde se jistě z hlediska úspory času a energií vyplatí shromáždit data o plánovaném pohybu hlavy, o místech, kde bude hlava operovat, a nechat je zpracovat optimalizátorem, který tuto posloupnost míst upraví tak, aby celková délka přesunů hlavy byla co nejnižší.

Cílem této práce byla průběžná implementace stanovených úkolů a funkcionalit, které byly zadávány pracovníkem firmy, pro niž je tento systém implementován. Stanovené úkoly se v převážné většině vztahovaly na práci s mapou, proto se této oblasti týká i převážná většina implementace.

První kapitola této práce se zabývá analýzou produktů na trhu, které se věnují optimalizaci, a také produktů určených pro sledování flotily vozidel. Nechybí ani jejich vzájemné srovnání.

Ve druhé kapitole se podrobněji rozepíši o některých knihovnách a technologiích, které jsem použil při implementaci systému. Zejména jde o knihovnu Jsprit a knihovnu OpenLayers.

Další kapitola pak obsahuje rozbor architektury implementovaného systému a jeho jednotlivých součástí, od databázové vrstvy až po vrstvu prezentační.

Poslední kapitola se týká implementace – popisuji v ní jednotlivé části mnou implementovaného modulu systému a některé postupy při implementaci. Většina zadaných úkolů v průběhu implementace se týkala práce s mapou, proto se právě touto oblastí budu zabývat nejvíce. Na řadu ale přijde i část ohledně optimalizátoru.

2 Analýza konkurence

V rámci přípravy na implementaci praktické části této práce jsem analyzoval a porovnal několik vytipovaných systémů na trhu, které se zabývají sledováním flotily vozidel, analýzou jejich pohybu a zpracováním různých statistik a výstupů. Z porovnávaných systémů optimalizátor přeprav neobsahoval žádný, nicméně jedna z firem, jejichž systémy jsem analyzoval, vytvořila optimalizační nástroj jako svůj druhý produkt.

2.1 Optimalizátory

Zmíněnou firmou je firma Digitech, a její produkt Plantour. Proti tomuto nástroji stojí ve srovnání druhý optimalizační nástroj, který vyvinula firma Rinkai. Firma Rinkai se na rozdíl od ostatních analyzovaných firem nevěnuje vývoji monitorovacího systému a sledování vozidel, ale pouze vývoji a implementaci optimalizačního algoritmu, který firma využívá v produktu Rinkai Routing.

Plantour [6] je dispečerský plánovací systém, který připravuje efektivní plány dopravy a při plánování zároveň vyhodnocuje rentabilitu dodávek. Má také analytické možnosti - umožňuje definovat budoucí potřeby vozového parku, rozložení dep apod.

Plantour při vytváření plánů zohledňuje zákaznická omezení (např. časová okna zákazníků) i omezení v dopravě (aktuální omezení silniční sítě a parametry vozidla). Zároveň se snaží maximalizovat využití vozového parku a minimalizovat přepravní náklady. Výsledné plány zobrazí na mapě, v tabulce a na časové ose.

Nasazení systému má smysl pro firmy všech velikostí - výrobce uvádí snížení nákladů na dopravu po nasazení systému o 15 - 30 %.

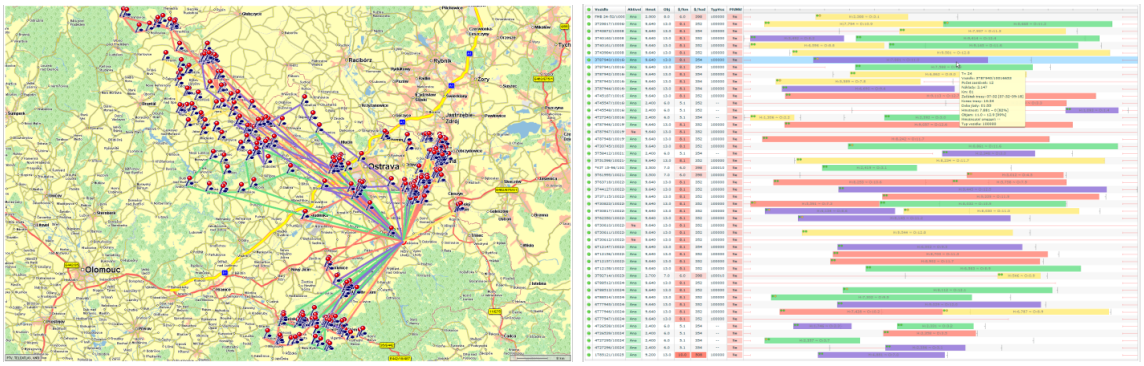
Optimalizační nástroj **Rinkai Routing** [3] umožňuje vytvořit efektivní plán dopravy podle zákaznických objednávek, který respektuje přepravní a zákaznická omezení. Při tvorbě plánu se RiRo snaží o maximální využití dostupného vozového parku a minimalizaci přepravních nákladů. Nasazení tohoto systému má smysl pro všechny velikosti firemního vozového parku.

Společnost Rinkai uvádí, že při použití produktu RiRo je možné dosáhnout snížení přepravních nákladů o 10 - 20 %. Systém také dokáže odhadnout dopad změny struktury vozového parku.

Na Obrázku 1a [4] můžeme vidět příklad zobrazení rozvozových tras na mapě a na Obrázku 1b [5] rozpis rozvozů na časové ose.

2.2 Porovnání optimalizátorů

Jak ukazuje Tabulka 1, porovnávané optimalizátory jsou si velmi podobné, liší se pouze v několika málo ohledech. Co se týče tvorby plánu, snaží se oba akceptovat požadavky zákazníka a také omezení provozu. Plantour ale oproti RiRo nabízí navíc analýzu budoucích potřeb vozového parku a jeho udávaná úspora je o něco vyšší než v případě Rinkai Routing. Možná proto výrobce nástroje Plantour uvádí, že tento produkt je nejčastěji nasazovaným nástrojem pro plánování tras.



(a) Rozvozové trasy na mapě (b) Rozvozy na časové ose

Obrázek 1: RiRo – Ukázky ze systému

	Plantour	**RiRo**
Omezení braná v úvahu při tvorbě tras		
Časová okna zákazníků	Ano	Ano
Omezení na určitý typ vozidla	Ano	Ano
Dostupnost vozidel	Ano	Ano
Kapacita vozidla	Ano	Ano
Omezení silniční sítě	Ano	Ano
Další parametry		
Analýza budoucích potřeb	Ano	Ne
Udávaná úspora po nasazení systému	15 - 30 %	10 - 20 %

Tabulka 1: Porovnání Plantour a Riro

2.3 Monitorovací systémy

Monitorovacích systémů je dnes na trhu celá řada. V základních funkcích jsou si všechny velmi podobné, některé z nich jsou ale koncipovány jednodušeji, takže u sady základních funkcí také zůstávají a jsou tak vhodné pro všechny nenáročné zákazníky, kteří chtějí např. pouze sledovat vozidla a generovat knihu jízd a nic jiného nepotřebují.

Další systémy jsou složitější, obsahující sadu pokročilejších funkcí, od imobilizéru (možnost z dispečerského okna na dálku zastavit vozidlo), přes vyhodnocování stylu jízdy řidiče až po sledování provozu externích čidel (např. teplota v nákladovém prostoru). Podle informací od výrobců je většina systémů škálovatelných, takže je lze přizpůsobit přesně podle potřeb zákazníka.

V následujících dvou tabulkách (Tabulka 2 a Tabulka 3) uvádím srovnání všech analyzovaných systémů podle vybraných kritérií. Do první tabulky jsem zařadil kritéria, která jsou podle mě z hlediska provozu systému a dispečera důležitější.

	Zpětné vykreslení trasy na mapě	Identifikace řidiče	Jazykové mutace systému	Komunikace s řidičem
GPSDozor	Ano	Ano	Ano	Ne
AutoGPS	Ano	Ano	Ano	Ne
Webdispečink	Ano	Ano	Ano	Ano
Fleetware	Ano	Ano	Ne	Ne
ONI system	Ano	Ano	Ne	Ano
LogisCarE	Ano	Ano	Ne	Ano
CarNet	Ano	Ano	Ne	Ano
SherlogTrace	Ano	Ano	Ano	Ne
Digitech	Ano	Ano	Ano	Ano
GX Solutions	Ano	Ano	Ano	Ano

Tabulka 2: Srovnání monitorovacích systémů, část 1

	Nejen webová aplikace	Vyhodnocování stylu jízdy řidiče	Zabezpečení vozidla	Imobilizér
GPSDozor	Ano	Ano	Ne	Ano
AutoGPS	Ano	Ne	Ano	Ano
Webdispečink	Ano	Ano	Ne	Ne
Fleetware	Ano	Ne	Ano	Ne
ONI system	Ne	Ano	Ano	Ne
LogisCarE	Ano	Ne	Ano	Ne
CarNet	Ne	Ano	Ano	Ano
SherlogTrace	Ne	Ano	Ne	Ne
Digitech	Ano	Ano	Ano	Ano
GX Solutions	Ano	Ano	Ne	Ne

Tabulka 3: Srovnání monitorovacích systémů, část 2

3 Analýza systému

Systém se skládá ze tří serverů (viz Obrázek 2) – databázového, aplikačního a webového – a webového prohlížeče jakožto nástroje pro prezentaci dat. V architektuře implementovaného systému se kombinují prvky třívrstvé architektury a modelu MVC.

Třívrstvá architektura se uplatňuje při přístupu k databázi - objekty prezentační vrstvy nemohou přímo ukládat data do databáze, musejí o to požádat objekty aplikační vrstvy, a MVC model se uplatňuje při komunikaci v prezentační vrstvě. Jde o webovou aplikaci, kde části Model a Controller jsou reprezentovány třídami v odpovídajících složkách na webovém serveru a část View jsou webové stránky, které jsou také uloženy na webovém serveru a z něj se posílají do prohlížeče, odkud se zobrazují uživateli.

3.1 Vrstva přístupu k datům

Databází, na kterou je implementovaný systém napojen a ze které čerpá data, je databáze PostgreSQL [9]. PostgreSQL je objektově-relační DBMS šířený pod licencí, která umožňuje tento open-source produkt využívat, modifikovat a distribuovat v libovolné formě, např. jako databázi v rámci nějaké průmyslové aplikace.

Po více než 15 letech usilovného vývoje se PostgreSQL řadí mezi nejrobustnější DBMS vůbec. Kolem PostgreSQL se během té doby vytvořila poměrně široká uživatelská komunita. Tato databáze je primárně vyvíjena pro unixové systémy, ale podporuje všechny běžně rozšířené operační systémy (Linux, Solaris, FreeBSD, Mac OS, Windows).

K databázi se přistupuje pomocí knihovny Npgsql [2], která aplikaci postavené na .NET frameworku poskytuje spojení a komunikaci (přenos dat) s databází PostgreSQL, pokud databázový server, na kterém databáze běží, naslouchá TCP spojením.

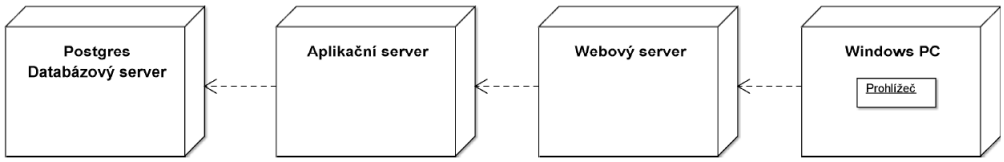
3.2 Aplikační vrstva

Aplikační vrstvu (nebo také Bussiness vrstvu) tvoří zejména aplikační server, nicméně i na webovém serveru probíhá část aplikačních výpočtů, proto jsem jej do aplikační vrstvy zahrnul také. Software na obou serverech je postaven na frameworku .NET, psán je konkrétně v jazyce C#.

3.2.1 Aplikační server

Aplikační server tedy slouží k aplikačním výpočtům v jednotlivých modulech (např. modul hlídání událostí, modul dispečerská plachta) a také obsahuje rozhraní pro volání metod pro přístup k databázi.

Aby všechny tyto metody aplikačního serveru mohly být využity, funguje na portu 8733 webová služba, která vystavuje rozhraní – množinu metod, pomocí kterých se lze dostat k metodám v modulech a také k metodám pro přístup k DB a které může využít každá aplikace, která se pomocí servisní reference k uvedenému portu připojí a tím pomocí jazyka WSDL zjistí, jaké metody aplikační server nabízí, a pomocí protokolu SOAP je může využít.



Obrázek 2: Architektura systému

Co se týče přístupu k databázi, je samozřejmě nutné přistupovat k DB s objekty, které DB zná, které má nadefinované. Proto je třeba objekty, které jsou používané v aplikaci, převést na databázové objekty. K těmto převodům mezi ekvivalentními objekty (směrem do databáze i z databáze) slouží třída Mapper.

3.2.2 Webový server

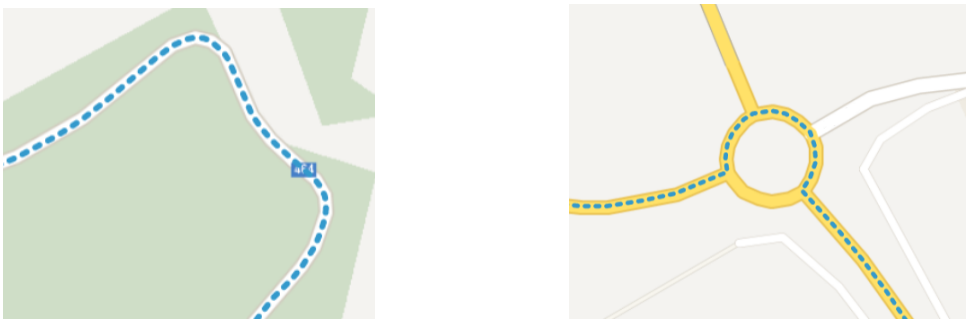
Webový server sice slouží primárně k obsluze požadavků od uživatele (z prohlížeče), nicméně obsahuje také část aplikační logiky. Aplikační logiku při obsluze požadavku od klienta využije jedním ze tří způsobů:

- Pokud ví, jak na požadavek odpovědět, pak sám odpověď zpracuje a odešle (zde se jedná o požadavky typu např. zobrazení stránky).
- Pokud se dotaz týká routingu nebo geolokace, zavolá odpovídající službu. Té předá potřebná data a počká na odpověď. Jakmile ji dostane, zpracuje ji do potřebného formátu a odešle zpět uživateli do prohlížeče, kde odpověď zpravidla projde ještě několika fázemi úprav v javascriptu, načež je zobrazena uživateli.
- Pokud je nutné se pro vytvoření odpovědi na požadavek připojit k databázi a získat z ní nějaká data nebo webový server nezná algoritmus pro zpracování odpovědi, zašle pomocí připojené služby svůj požadavek na aplikační server. Ten jej zpracuje (podle potřeby se připojí k databázi, ze které získá potřebné informace) a vytvoří odpověď, kterou odešle zpět webovému serveru. Webový server odpověď potřebným způsobem zpracuje a pošle uživateli do prohlížeče konečný výstup (HTML kód stránky).

3.2.3 Služby

Jak jsem v předchozím bodě zmínil, k webovému serveru je pomocí referencí připojeno několik služeb běžících na vzdálených serverech:

1. Služba pro práci s aplikačním serverem - služba, která umožňuje využít všechny metody vystavené pomocí rozhraní aplikačním serverem. Při své práci jsem tuto službu využil zejména pro připojení k databázovému serveru pro získání dat.
2. Routovací služba - služba, která momentálně běží na serveru superpočítačového centra IT4Innovations a která umožňuje vyhledat trasu mezi dvěma body (dvěma



(a) Zatáčky (b) Kruhový objezd

Obrázek 3: Plynulost geometrie na mapě

místy) na zemi. Služba přijímá jako parametry zeměpisné souřadnice obou bodů (výchozího a cílového), údaj o placenosti trasy (zda má služba trasu vyhledávat také přes placené úseky silnic, např. dálnice nebo některé silnice 1. třídy), údaj o typu vozidla, pro který se trasa hledá (osobní, tir), a s ním spojené "truck parametry"(maximální rychlost, výška a celková váha vozidla) a údaj, jestli má být vyhledána trasa nejkratší nebo nejrychlejší.
Výstupem pak je vyhledaná trasa, přesněji její podrobná geometrie (jednotlivé body geometrie velmi přesně kopírují silnici zobrazenou na mapě, zvláště v zatáčkách – viz Obrázek 3a a na kruhových objezdech – viz Obrázek 3b. Zde je třeba dodat, že body geometrie jsou spojeny pouze rovnými čarami, takže i když to na vizuálním výsledku není úplně patrné, jedná se pouze o spojnici mnoha rovných čar). Spolu s geometrií trasy je vrácena i její délka a čas, jak dlouho trvá cesta po této trase.

3. Geolokační služba - tato služba funguje na serveru společnosti GX Solutions a slouží ke zjištění informací o místě daném GPS souřadnicemi. Potřebnými parametry pro získání výsledků jsou obě souřadnice místa a uživatelský klíč. Služba vrátí jednotlivé složky kompletní adresy, pokud jsou pro dané místo k dispozici (ulice, č.p., město, PSČ, stát), a také další údaje jako maximální rychlost nebo údaj o zpoplatnění silnice. V této práci z navrácených hodnot využívám pouze adresu místa a maximální rychlost.

3.2.4 Optimalizátor

Optimalizátor je důležitou součástí aplikační vrstvy, jelikož jednou z funkcionalit tohoto systému je také plánování objednávek a optimalizace rozvozových tras. Podrobněji viz Kapitola 5. V této aplikaci jsou optimalizace řešeny pomocí algoritmů knihovny Jsprit 4.3.

3.3 Prezentační vrstva

Mezi prvky prezentační vrstvy se dá zařadit zaprvé webový server, který má v sobě uložena data k prezentování (webové stránky) a tato data renderuje a odesílá zpět do prohlížeče jako odpověď na požadavek. A za druhé také webový prohlížeč s povoleným javascriptem (pokud je totiž javascript v prohlížeči zakázaný, stránky nemohou řádně fungovat), protože ten prezentuje data uživateli.

3.3.1 MVC

Jak jsem se již lehce zmínil v úvodu analýzy systému (viz Kapitola 3), je prezentační vrstva na webovém serveru postavena na modelu MVC, konkrétně na technologii ASP.NET MVC verze 5. Tato technologie umožňuje vytvářet dobře strukturované a jednoduše udržovatelné aplikace [13].

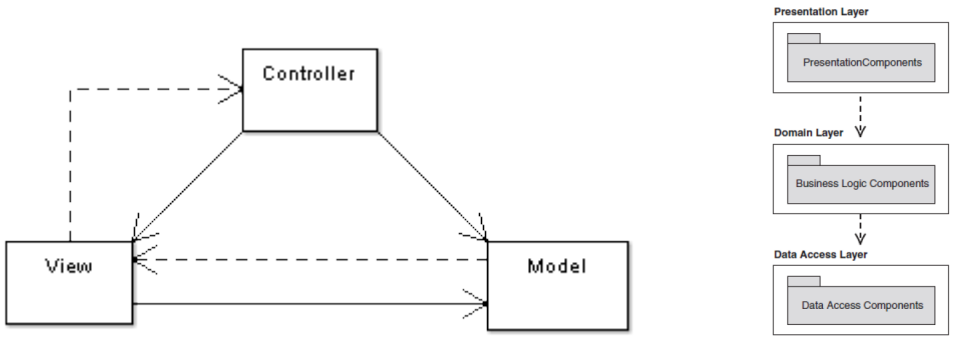
Model MVC se od modelu třívrstvé architektury liší v tom, že není lineární – jeho komponenty leží jakoby na jedné úrovni a mohou se odkazovat vzájemně jedna na druhou (viz Obrázek 4a a Obrázek 4b).

Aplikace postavené na modelu MVC obsahují tři části:

- Model
- View
- Controller

Do **modelu** [10] patří třídy, které reprezentují data aplikace. V systému jsou jednak proto, aby prezentační vrstva měla kam sáhnout pro data – často v nich jde o distribuci dat z databáze (využití metody modelu, která zavolá metodu na aplikačním serveru, aby získala data z DB) a jednak aby je mohlo View zpracovat a zobrazit.

Model provádí svou akci (myšleno přístup do databáze) vždy až tehdy, když mu k tomu Controller dá pokyn, a jeho volání bývá často spjato s vytvářením View – jeho data jsou ve View prezentována uživateli.



(a) MVC (b) 3vrstvá architektura

Obrázek 4: Porovnání modelu MVC a 3vrstvé architektury

Část **View** jsou soubory, které obsahují kód zobrazovaných webových stránek. V systému jsou soubory View řešené jako Razor Partial View a mají oproti běžně používaným souborům poměrně neobvyklou příponu *.cshtml*. Tato přípona napovídá, že se soubor bude skládat ze směsice úryvků kódu jazyka C# a jazyka HTML.

Razor [11] je jednoduchá programovací syntaxe pro vkládání kódu serveru do webových stránek. Je založená na syntaxi ASP.NET a umožňuje do jednoho souboru vkládat úseky HMTL (podobně jako do klasického `.html` souboru) a zároveň úseky kódu jazyka C#. Tyto úseky ale mají v Razor souboru několik omezení:

- Bloky kódu musí začínat znakem @ a musí být uzavřeny ve složených závorkách
- Inline výrazy v závorkách být nemusí, ale musí rovněž začínat znakem @
- Proměnné se deklarují klíčovým slovem `var`

Kromě těchto omezení se syntaxe kódu řídí standardními pravidly. Zde je ukázka kódu se zmíněnými omezeními.

```
@{
    ViewBag.Title = "Home Page";
}

<p>Now: @DateTime.Now</p>

@{
    var first = 1;
    var second = 2;
    var result = first + " + " + second + " = " + ( first + second);
}
```

Výpis 1: Ukázky Razor syntaxe v souboru .cshtml

Soubory View jsou často využívány pro zobrazení nejen vzhledu, ale také nějakých dat. Pokud je tedy třeba uživateli na stránce zobrazit data (kolekci hodnot, řetězec, číslo apod.), pak se někam do souboru (nejlépe na začátek) uvede konstrukce `@model`, za níž se uvede typ modelu (dat), který se na stránce má zobrazit, např. takto:

```
@model IEnumerable<WebApp.NameModel.Surnames>
```

Skládání stránek z jednotlivých Partial View je velmi jednoduché, do stránky stačí napsat jednu ze čtyř konstrukcí:

- `@Html.RenderPartial`
- `@Html.RenderAction`
- `@Html.Partial`
- `@Html.Action`

Každou z těchto metod je vhodné použít v truchu jiném případě, obecně ale platí, že metody, které obsahují sufix *Action*, volají controller a v něm metodu, jejíž název mají jako parametr, ostatní dvě načítají View podle názvu uvedeného v parametru.

V systému se metody s prefixem *Render* obvykle volají tam, kde se View načítá inline jako součást nějaké UI komponenty (jednotlivá okna v levém navigačním panelu, obě strany splitteru atd.), metody bez tohoto prefixu pak obvykle tam, kde se View načítá přímo do HTML kódu.

Pokud se načítají soubory View, které v sobě obsahují příkaz na renderování dalšího View souboru (ať už jako součást UI komponent nebo v HTML kódu), načte tento příkaz uvedený soubor View a vyrenderují se všechny další soubory View, které v něm jsou uvedeny. Takto rekurzivně načítá, až dojde k souboru, který v sobě žádný příkaz na renderování dalšího View neobsahuje. V systému jsou jako náplň souborů View velmi často voleny komponenty společnosti DevExpress.

Controller je část aplikace, na kterou směřují požadavky z prohlížeče. Oproti technologiím jako WebForms nebo klasické ASP.NET, kde každá URL adresa je reprezentována jedním souborem, který požadavek přijme, zpracuje a vrátí odpověď, v modelu MVC jsou URL adresy reprezentovány jednotlivými metodami ve třídách, které dědí ze třídy ControllerBase. Pokud tedy uživatel zadá do prohlížeče URL adresu a stiskne enter, pak de facto zavolá metodu ve třídě.

V implementovaném systému jsou metody Controlleru volány dvojím způsobem:

1. Callback DevExpress komponent
2. Z javascriptu pomocí jQuery konstrukce $.ajax (API pro použití technologie AJAX v jQuery) nebo z View rovněž pomocí AJAXu

Podle typu požadavku tato metoda v Controlleru zavolá odpovídající aplikační logiku. Pokud se jedná o volání prvního typu, má volaná metoda návratový typ ActionResult a vrací některý z typů, které z ActionResult dědí (`RedirectToActionResult`, `ViewResult` atd.), pokud o volání druhého typu, pak metoda v Controlleru vrátí string (různé formáty – XML, JSON nebo jednoduchý text) a s vrácenou hodnotou se dá v JavaScriptu dále pracovat.

3.3.2 Webový prohlížeč

Webový prohlížeč je velmi důležitou součástí celé aplikace, protože skrz něj probíhá veškerá interakce mezi uživatelem a systémem. Komunikace mezi prohlížečem a serverem (Controllerem) je postavená na technologii AJAX. AJAX umožňuje měnit pouze části stránek a pracuje na pozadí - odeslání požadavku, příjem a zpracování odpovědi, to vše se děje na pozadí, navíc se nemění URL adresa v prohlížeči, takže si uživatel nemusí ani všimnout, že proběhla nějaká interakce se serverem.

Ale prohlížeč tady není jenom od toho, aby na pozadí komunikoval se serverem, ale zároveň aby např. stahoval obrázky, které jsou na dané stránce umístěny, a další soubory, které stránka využívá, jako např. šablony CSS stylů nebo zdrojové kódy JavaScriptu, pomocí kterých prohlížeč poskytuje uživateli interaktivitu na stránce.

V tomto projektu se používá několik desítek zdrojových souborů jazyka JavaScript, některé jsou napsané námi, programátory, další jsou volně dostupné knihovny zaměřené na určitou oblast. Mezi tyto knihovny patří např. i knihovna jQuery nebo knihovna OpenLayers. O těchto knihovnách bude zmínka v další kapitole.

4 Použité knihovny a technologie

V této kapitole uvádím knihovny a technologie, s nimiž jsem pracoval v průběhu implementace. Dvěma knihovnami jsem se zabýval více - knihovnou Jsprit kvůli optimalizátoru a knihovnou OpenLayers kvůli práci s mapou, tyto dvě knihovny tedy rozvedu podrovněji.

4.1 jQuery

JQuery je JavaScriptová knihovna, která dosahuje maxima v poměru funkcionalita / velikost. JQuery podporuje a zednodušuje práci s HTML DOM, CSS vlastnostmi, událostmi, animacemi a také s AJAXem, kterému poskytuje velmi jednoduše použitelné API.

V projektu se knihovna hojně využívá pro manipulaci s HTML DOM (vytváření a úprava HTML elementů), pro nastavování CSS vlastností a událostí a také pro využívání AJAXu.

4.2 OpenLayers

OpenLayers je rovněž JavaScriptová knihovna určená pro zobrazení a zpracování map ve webovém prohlížeči, která poskytuje API pro tvorbu kvalitních webových mapových aplikací. Funguje čistě na straně klienta, není tedy závislá na serverové části. Je open-source a zdarma.

Po dlouhém vývoji byla na konci srpna minulého roku (2014) vydána nová verze knihovny s označením OpenLayers 3.0. Tato nová verze přinesla oproti starším verzím s označením 2.x spoustu nových možností – jednodušeji použitelné API, vylepšené vykreslování vektorů a manipulaci s obrázky a další [12]. To je také jeden z hlavních důvodů, proč knihovna OL verze 3.x a záležitosti s ní spojené ještě nejsou příliš diskutovány na různých IT fórech (Stackoverflow, gis.stackexchange apod.) a je proto mnohdy obtížné získávat podporu při různých problémech.

Knihovna se i nadále vyvíjí a přibývají nové funkcionality – od srpna minulého roku do současnosti (květen 2015) bylo vydáno několik nových verzí, momentálně aktuální má označení 3.4.0.

Mapa, kterou je pomocí OpenLayers možné na stránce zobrazit, je dynamická – knihovna umožňuje mapu volně posouvat, přibližovat a oddalovat, novější verze pak dokonce i otáčet. Knihovna OL k mapě poskytuje i dynamické měřítko, které se mění podle úrovně přiblížení – mění se zobrazené jednotky i velikost měřítka.

4.2.1 Vrstvy

Knihovna OpenLayers nemá svůj název náhodou - součástí knihovny (součást, která se v systému hojně využívá pro zobrazení objektů na mapě) jsou totiž mapové vrstvy a práce s nimi. V projektu jsou na mapě použity dva různé typy vrstev – Tile a Vector.

Vrstva Tile (česky dlaždice) zobrazuje předpřipravené mapové podklady (obrázky) jako dlaždice a skládá je vedle sebe.

Vrstva Vector pak na tuto podkladovou vrstvu zobrazuje různé mapové objekty – obrázky, tečky, čáry atd. Vrstvě Vector lze přiřadit styl, který je poté aplikován na všechny objekty v dané vrstvě, pokud není přepsán definicí stylu u samotného objektu. Každá mapová vrstva musí mít svůj zdroj, ze kterého čerpá zobrazovaná data – vrstva Tile má dlaždice, vrstva Vector má objekty typu Feature.

4.2.2 Feature

Objekt Feature je významným objektem knihovny OL. Je to základní objekt, který lze přiřadit do zdroje mapové vrstvy a tím jej zobrazit.

Feature obsahuje property Geometry, která udává, jak a kam bude objekt Feature na mapě zobrazen. Dále, jak už jsem uvedl v předchozím odstavci, může být objektu Feature přiřazen styl. Pokud přiřazen není, pak se použije defaultní styl dané vrstvy. Objektu Feature lze definovat také vlastní dodatečné parametry, které mohou sloužit k identifikaci nebo ke zobrazení v bublině na mapě u odpovídajícího objektu Feature.

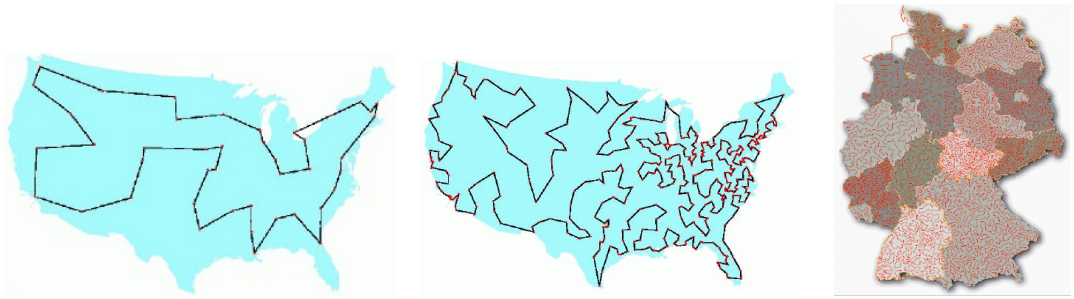
4.3 Jsprit

Jsprit [1] je knihovna v jazyce Java, která slouží k řešení složitých optimalizačních problémů. Knihovnu vytvořil Stafan Schröder a se svými spolupracovníky ji stále udržuje. Stefan měl dvojí motivaci k jejímu vytvoření: Za prvé si uvědomil, že routovací problémy jsou všude, kde se něco přemisťuje z místa na místo, takže není náhodou, že již existuje dalekosáhlý seznam algoritmů a řešení, jak se s těmito problémy vypořádat.

Nicméně existuje pouze velmi málo řešení, které jsou implementovány a jejichž implementace je volně dostupná jako open-source (tj. zdrojové kódy jsou volně ke stažení). Ještě méně je pak těch, které se dokáží vypořádat s problémy z reálného světa přinášející do zadání problémů další omezení. A druhou motivací byla Stefanova disertační práce, která se zabývala právě routovacími algoritmy a kde mohl tento projekt využít.

Knihovna Jsprit řeší optimalizační příklady vycházející ze dvou grafových problémů:

- Problém obchodního cestujícího (angl. TSP – Travelling Salesman Problem)
- Problém okružních jízd (angl. VRP – Vehicle Routing Problem)



(a) USA, 1954 (b) USA 1954 (c) Německo 2001

Obrázek 5: Příklady TSP Tour

4.3.1 Problém obchodního cestujícího

TSP [7] spočívá v tom, že je zadaná množina měst (každé město má své souřadnice) a množina silnic mezi městy, jejichž délky známe. Úkolem je nalézt nejkratší možnou trasu procházející právě jednou všemi zadanými městy a vracející se zpět do výchozího města.

Pokud bychom problém formulovali pomocí teorie grafu, pak hledáme nejkratší hamiltonovskou kružnici v úplném grafu nebo také nejkratší uzavřený sled průchodů všemi uzly. Pokud se v grafu hamiltonovskou kružnici nalézt podaří, nastává ideální situace, protože v tomto případě obchodní cestující navštíví každé město právě jednou a skončí v počátečním uzlu a tím obrazně nakreslí graf jedním tahem.

Existuje hra založená na řešení problému obchodního cestujícího s názvem TSP Tour. Jde v ní o to, že je zadán seznam měst (míst na mapě) a úkolem je najít nejkratší cestu mezi všemi těmito městy. Pointa hry spočívá v přidávání dalších míst do mapy a tím i nových silnic, takže se řešení neustále mění. Tato hra byla aplikována na města v USA, Kanadě, Německu a dalších zemích - v USA v roce 1954 na 49 měst (Obrázek 5a) a v roce 1987 na 532 měst (Obrázek 5b), v Německu pak v roce 2001 na 15 112 obcí (Obrázek 5c).

Nejpřímější možnost **řešení** TSP problému je řešení využitím "selského rozumu"neboli pomocí hrubé síly. Toto řešení zkoumá všechna možná uspořádání posloupnosti míst a délku každého uspořádání porovnává s nejnižší dosud nalezenou délkou. Pokud označíme počet možností n a počet míst x, pak platí vztah:

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

Počáteční bod se totiž do posloupnosti nepočítá, proto místo x píšeme $(x-1)$, a každá cesta je ve výsledku započítána dvakrát (např. A-B-C-D-A i A-D-C-B-A), proto se výsledek vydělí 2. Vzhledem k výpočetní složitosti (pro pouhých 16 vrcholů počet možností dosahuje čísla 650 miliard, což by při ohodnocovací rychlosti 1 cesta za milisekundu trvalo přes 20 let) je toto řešení vhodné pouze pro malý počet míst a proto se moc nepoužívá.

Mezi používané metody patří metody heuristické, které se zabývají jen vybranými trasami, jsou tedy rychlé, ale nezaručují, že výsledek je reálně nejkratší. Kritériem pro

výběr trasy je např. výběr nejbližšího souseda. Dalšími výpočetními metodami jsou metody metaheuristické, které místo lokálních minim hledají minima globální - berou se v úvahu i horší řešení - a jejich výsledky jsou reálně blízké nejkratší cestě.

4.3.2 Problém okružních jízd

Problém okružních jízd je obměnou problému obchodního cestujícího. Obměna spočívá v tom, že je definována množina míst na mapě, které je nutné při rozvážce navštívit, a mezi nimi výjezdní místo (depo). Každé místo na mapě má definováno, kolik jednotek zboží vyžaduje. Dále je definována množina vozidel, které se nachází ve skladě a které je možné pro okružní jízdy využít.

Úkolem tedy tentokrát není nalézt jeden nejkratší okruh přes všechna místa, ale navrhnout z vozidel, které jsou dostupné v depu, množinu vozidel pro okružní jízdy a ke každému vozidlu přiřadit několik míst jako okružní jízdu tak, aby celková délka okružních jízd byla co nejmenší a tím se maximálně snížily náklady na provoz. Navíc se každé vozidlo smí použít pouze jednou (omezení proti opakovanému využití vozidla s největší kapacitou nebo nejmenší spotřebou).

4.3.3 Další optimalizační problémy

Jsprit dokáže řešit i další optimalizační problémy odvozené ze dvou výše zmíněných:

- Capacitated VRP – obsluhujícím vozidlům je zadána kapacita a při řešení nesmí být kapacita žádného z vozidel překročena
- Multiple Depot VRP – je definováno více dep
- VRP with Time Windows – jsou definována a brána v úvahu časová okna zákazníků
- VRP with Pickups and Deliveries – na kterékoli zastávce se může zboží nakládat i vykládat

4.3.4 Obsah knihovny

Knihovna Jsprit je tvořena těmito čtyřmi moduly: *core*, *analysis*, *instances* a *examples* (viz Tabulka 4). Pro využití optimalizačního algoritmu z knihovny budeme ale potřebovat pouze moduly dva - *core* a *analysis*. Modul core slouží k samotnému řešení problémů včetně zadání problému (např. definice typů vozidel, jejich parametrů a nosnosti, definice zastávek a časových oken zákazníků). Modul analysis pak umožňuje interpretovat a prezentovat získané výsledky, např. výpisem do xml souboru nebo vykreslením do obrázku a grafu.

Název modulu	Význam
core	Zadávání a řešení problémů
analysis	Analýza a prezentace řešení
instances	Převedení zadání ze souborů do příkladů
examples	Předpřipravené spustitelné příklady

Tabulka 4: Moduly knihovny Jsprit a jejich význam

4.3.5 Použití knihovny

Co se týče jazyka Java, je použití knihovny velmi jednoduché. Je ale třeba mít nainstalované vývojové prostředí pro jazyk Java (např. IDE NetBeans) a JDK. Pokud jsme splnili tyto dvě podmínky, vytvoříme v IDE Maven projekt a do něj přidáme referenci na moduly jsprit knihovny, které chceme v aplikaci využít. Knihovny obsahují reference na další knihovny z JDK i mimo JDK - ty, které v JDK nejsou, se ale automaticky samy stáhnou a uloží na disk při prvním sestavení projektu.

Pro použití v jazyce C# (obecně v jazyce využívajícím .NET framework) je jedna z možných cest převedení komponent knihovny jsprit z formátu `.jar` do formátu `.dll`. Pro tento převod mezi formáty jsem využil nástroj IKVM.NET. Tento nástroj poskytuje aplikaci *ikvmc* (IKVM compiler), která se spouští prostým zavoláním z příkazového řádku a jako argumenty bere údaj, jestli chci kompilovat do aplikace (`.exe`), knihovny (`.dll`) nebo modulu, údaj o názvu výstupního souboru a samozřejmě jeden nebo více vstupních `.jar` souborů.

Pro správnou funkčnost výsledné aplikace je potřeba zvolit jako vstupní soubory všechny potřebné soubory, tj. hlavní kompilovanou knihovnu a všechny na ní závislé (z ní referencované), aby se při kompilaci přenesly zároveň všechny reference. Pokud totiž při převodu některá knihovna chybí, její reference v jiné knihovně (import v hlavičce zdrojového souboru) se nepřenese a výsledná aplikace neví, kde má danou třídu najít.

V případě kompilace knihoven *jsprit-core* a *jsprit-analysis* (i tyto dvě knihovny se musí kompilovat najednou, protože obsahují vzájemné reference. Takto jsem volání *ikvmc* použil i já, jelikož jsem potřeboval obě knihovny) vypadá příkaz do příkazové řádky následovně [1]:

```
ikvmc -target:library -sharedclassloader { jsprit-core-1.6.1.jar }
{ jsprit-analysis-1.6.1.jar } { bcmail-jdk14-138.jar } ...
{ xml-apis-1.3.04.jar }
```

Na Obrázku 6a a Obrázku 6b je vidět rozdíl v referencích v hlavičce mezi původním souborem v jazyce Java a zkompilovaným souborem v jazyce C#, pokud při kompilaci

[1] Referencovaných knihoven je v tomto případě 21, proto jsem je nevypisoval všechny.

```java
package jsprit.core.problem.vehicle;

import jsprit.core.problem.AbstractVehicle;
import jsprit.core.problem.Location;
import jsprit.core.problem.Skills;
import jsprit.core.util.Coordinate;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

(a) Zdrojový kód v Javě

```csharp
using IKVM.Attributes;
using java.lang;
using jsprit.core.problem;
using jsprit.core.util;
using System;

namespace jsprit.core.problem.vehicle
```

(b) Zdrojový kód v C#

Obrázek 6: Chybějící import

chybí knihovna *log4j-api.jar* (kromě známých rozdílů mezi jazyky Java a C# – použití klíčových slov a toho, že jazyk Java vyjmenovává všechny třídy, které chce použít, kdežto jazyk C# pouze celý jmenný prostor, ve kterém jsou dané třídy definovány, takže např. tři příkazy `import` tříd ze třídy *jsprit.core.problem* v levém obrázku jsou sloučeny do jediného příkazu `using` namespace v pravém obrázku - je mezi uvedenými zdrojovými kódy ještě jeden zásadní rozdíl – chybějící reference na namespace *org.apache.logging.log4j*).

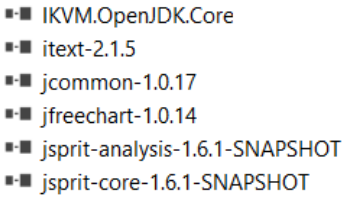
Nyní, když jsou zkompilované obě knihovny Jsprit a také všechny ostatní závislé knihovny, můžeme je v projektu referencovat. Pro zdárný běh aplikace se zkompilovanými knihovnami je třeba přidat ještě referenci na přeloženou sadu základních knihoven z JDK (Java Core API). Překlad těchto `.jar` knihoven již ale autoři nástroje IKVM provedli za nás, proto stačí referenci na knihovnu *IKVM.OpenJDK.Core.dll*, která je distribuována v rámci nástroje IKVM, takže se nemusí zvlášť stahovat, přidat do projektu (viz Obrázek 7).

4.4 DevExpress

Společnost DevExpress [14] (celým názvem Developer Express) se již přes 15 let zabývá vývojem komponent pro uživatelská rozhraní aplikací na .NET frameworku, ať už jde o aplikaci webovou, desktopovou nebo mobilní.

V projektu je využívána celá řada komponent společnosti DevExpress, mezi ty méně obvyklé patří např. `Menu`, `TreeList`, `GridView` nebo `Splitter`. Používají se ale i úplně obyčejné formulářové ovládací prvky jako `Textbox`, `Checkbox` nebo `Radiobutton`.

DevExpress komponenty podporují naslouchání na události na straně klienta, např. `Init`, `Click` nebo v případě komponenty callback panel `BeginCallback` a `EndCallback`.



Obrázek 7: Visual Studio – reference v projektu (výběr)

Na všechny tyto události resp. na jejich handlery lze také přímo napojit volání metod z javascriptu, a to při definici handlerů. Buď lze z handleru přímo zavolat jedinou funkci, pak se uvádí název funkce bez závorek, nebo prostřednictvím pomocné funkce, potom se název funkce uvádí i se závorkami.

```
settings.ClientSideEvents.Click = "doSomething";   //prime volani  funkce bez zavorek

settings.ClientSideEvents.Click = "function  ()  {  doSomething(); }";    // volani  pres pomocnou fci
```

Výpis 2: Napojení metody na událost

Při vkládání DevExpress komponent je možné také definovat některé jejich CSS vzhledové vlastnosti, a to buď přímým vyjmenováním vlastností a přiřazením hodnot (např. výška, šířka, zobrazený kurzor) nebo přiřazením CSS stylu a jeho následnou definicí v souboru kaskádových stylů.

Rovněž lze definovat a upravovat zobrazení ovládacích prvků komponenty a její chování v případě, že se některý z ovládacích prvků využije. Např. při vložení komponenty GridView můžu nastavením hodnoty property `CommandColumn` zařídit zobrazení sloupce s ovládacími prvky a definovat, které ovládací prvky se ve sloupci zobrazí, nebo třeba definovat, v jakém formátu se zobrazí formulář pro vkládání a úpravu hodnot.

Komponenty společnosti DevExpress nabízení velké množství dalších možností a funkcionalit kromě výše zmíněných, které umožňují vývojáři nastavit komponentu přesně podle jeho představ. Snad i proto má společnost DevExpress motto

> Your Next Great App Starts Here

5 Implementace

Během implementace systému jsem dostával sady úkolů týkajících se jednotlivých celků v aplikaci. V této kapitole tyto celky podrobněji rozeberu. Celky byly dohromady čtyři:

1. Levý panel – vyhledávání, plánování trasy, měření
2. Horní panel – tabulka POI a oblastí, zobrazení na mapě
3. Geolokace – zobrazení adresy a maximální povolené rychlosti na mapě
4. Optimalizace – GUI a konfigurátor

5.1 Levý panel

První částí implementace bylo vytvořit nad mapou panel, který by se po stisknutí tlačítka vysunul nad mapu a po druhém stisknutí tlačítka se opět schoval. Tohoto efektu vysouvání panelu jsem docílil nastavováním CSS vlastnosti `display: block` pro zobrazení panelu a `none` pro jeho schování. S panelem souvisí i tlačítko na jeho otevření, jehož poloha se mění pouhým nastavením odsazení zleva – CSS vlastnosti `left`.

Panel velmi úzce spolupracuje s mapou – na mapě se vizuálně zobrazují výsledky, které jsou v panelu vypsány a funguje i interakce mezi mapou a panelem, ale o tom až dále.

Panel má tři části:

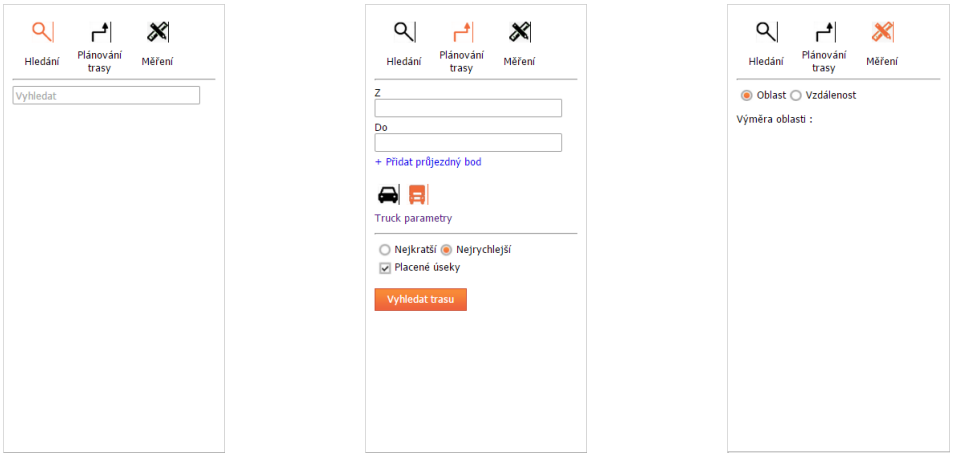
- Část pro vyhledávání míst
- Část pro vyhledání trasy
- Část pro měření a ukládání oblastí

Mezi těmito částmi lze přepínat pomocí radiobuttonů v horní části panelu, přičemž je na jednotlivé části aplikován stejný skrývací efekt, jako na celý panel - viditelná část má CSS vlastnost `display` nastavenou na `block`, ostatní dvě na `none`.

V první verzi byly radiobuttony implementovány jako klasické HTML `<input>` tagy, které byly typu `radiobutton` a byly překryty obrázkem, později jsem je přepracoval na DevExpress radiobuttony, které mají jako pozadí nastaven daný obrázek a mění se jim barva pozadí v závislosti na označení.

5.1.1 Vyhledávání míst

První část panelu je vyhledávací. Stejně jako u známých mapových serverů je zde pouze jedno vyhledávací okno (viz Obrázek 8a). Poté, co do něj uživatel vepíše vyhledávaný text (adresu nebo heslo, podle kterého se má místo hledat) a stiskne enter (počítá se s intuicí uživatele mapových serverů, žádné tlačítko "vyhledat"tady není), spustí se vyhledávání.



(a) Vyhledávání (b) Plánování trasy (c) Měření

Obrázek 8: Tři části levého panelu

Zadané heslo se nejprve vyhledává v databázi mezi POI (angl. Point Of Interest) objekty, které má daný zákazník uloženy. Pokud v databázi není nalezen žádný záznam, je využita připojená služba Google API pro překlad adresy na GPS souřadnice (tomuto překladu se říká také geocoding, opačnému pak reverzní geocoding). V obou případech se vrátí GPS souřadnice vyhledávaného bodu (jeden nebo více párů souřadnic – podle počtu výsledků vyhledávání).

Tyto GPS souřadnice lze posléze po transformaci (GPS souřadnice jsou v projekci EPSG:4326 – jednotky jsou desetinné stupně, pro zobrazení na mapě je ale třeba mít souřadnice v projekci EPSG:3857) pomocí objektu Point (objekt Point je určen pro zobrazení jednoho bodu – na rozdíl od objektů LineString a Polygon obsahuje pouze jeden pár souřadnic) přiřadit property geometry objektu Feature. Poté, co objektu Feature přiřadíme geometrii, už tento objekt ví, kam se má na mapě zobrazit, a tak jej zbývá přidat do mapové vrstvy a tím ji zobrazit na mapě.

```
var point = new ol.geom.Point([coordinateX, coordinateY]);
var features = new Array();
var feature = new ol.Feature({
    geometry: point
});
features.push(feature);

source.addFeatures(features);
```

Výpis 3: Přidání Feature do vrstvy

Jak jsem již dříve zmínil, každá mapová vrstva (jinak také objekt knihovny OpenLayers `ol.layer.Vector`) musí mít vlastní zdroj mapových prvků, aby je bylo možné zobrazit na mapě. Tento zdroj je typu `ol.source.Vector` a jako svou náplň přijímá javascriptové pole naplněné objekty typu Feature, proto mapový objekt z předchozího

odstavce musíme ještě přidat do pole a toto pole poté přiřadit do zdroje dané mapové vrstvy.

Bod se zobrazí na mapě (pokud je to jediný vyhledaný výsledek, pak uprostřed mapy, jinak se mapa vycentruje a přiblíží tak, aby byly v okně viditelné všechny výsledky) zvýrazněný bublinou a jeho textový popis (adresa a GPS souřadnice) se zobrazí v panelu vlevo. Takto vyhledaného výsledku je možno využít i pro plánování trasy kliknutím pravým tlačítkem myši na bublinu označující vyhledaný výsledek na mapě nebo na textový popis daného výsledku v levém panelu. Zobrazí se kontextové menu, ve kterém uživatel vybere, kterému bodu trasy chce tento bod přiřadit.

Již zde padla zmínka o jakési interakci mezi bočním panelem a vyhledaným výsledkem na mapě, resp. bublinou, která jej označuje. Pokud totiž vyhledám nějaké místo a poté přejedu myší v levém panelu po jeho popisu, zvětší se mi odpovídající bublina na mapě pro znázornění, že tento výsledek a tato bublina charakteritují stejné místo. Stejný efekt funguje i opačně - při přejetí bubliny na mapě se vyznačí položka v seznamu vyhledaných výsledků.

5.1.2 Plánování trasy

Druhá část panelu slouží k vyhledání trasy. Layout vypadá opět podobně, jako u mapových serverů (viz Obrázek 8b). Skládá se z několika oken pro adresy bodů trasy, voleb parametrů trasy a tlačítka pro vyhledávání. Poněkud neobvyklou součástí je seznam truck parametrů, které je třeba nadefinovat, pokud se plánuje trasa pro nákladní dopravu - mimo jiné obsahují parametry jako výška a váha vozidla a také jeho maximální ryhlost.

Je implementováno několik způsobů, jak definovat body trasy.

- Využije se výsledku vyhledávání (viz předchozí odstavec)
- Kliknutím do mapy, pokud je aktivní některé ze vstupů bodů trasy (je v něm kurzor)
- Využití kontextového menu nad mapou (menu, které se otevře po kliknutí pravým tlačítkem), funguje pouze pokud je levý panel otevřený
- Využití vozidla v panelu nad mapou v záložce "Online pozice"– kliknout pravým a zvolit "Navigovat od", funguje ale pouze pokud je mezi zařízeními v levém panelu vybráno nějaké vozidlo
- Vepsání adresy do okna

Spustit vyhledávání trasy je pak možné stisknutím tlačítka nebo enteru (opět pouze pokud je aktivní některé z oken bodů trasy – událost je registrována na ona okna). Při vyhledávání GPS souřadnic jednotlivých bodů trasy (pokud ještě nejsou zjištěny) se postupuje stejně jako při vyhledávání míst – nejprve se podle názvu místa (textu ve vstupním okně) hledá mezi POI objekty v databázi a poté pomocí Google API. Pokud jsou zjištěny souřadnice všech bodů trasy, zašlou se spolu s parametry trasy routovací službě.

5.1.2.1 Zobrazení na mapě

Pro zobrazení na mapě je využito dvou mapových vrstev. Jedna vrstva je určena pro body definující trasu (výchozí, cílový a průjezdné), do které se body přidávají stejně jako při vyhledávání. Prvky této vrstvy se na mapě vyobrazují jako barevné bubliny zapíchnuté do daného místa. Barva bubliny je vždy určena hodnotou dodatečného parametru `class` objektu Feature – podle jeho hodnoty přiřadí mapová vrstva objektu Feature styl – obrázek zelené, oranžové nebo červené bubliny.

Druhá vrstva je pro samotnou geometrii trasy, do které se přidá Feature s geometrií Linestring – tato geometrie je určena pro zobrazení čar na mapě. Tloušťku, barvu a další atributy čar v této vrstvě lze definovat ve stylech vrstvy nebo samotného objektu Feature. Lze také definovat, jestli čára bude přerušovaná – property `linedash`. Její hodnota je pole, kde první hodnota je délka čáry a druhá hodnota je délka čáry a mezery dohromady.

```
routePointsLinestrLayer = new ol.layer.Vector({
    source: someSource,
    style: new ol.style.Style({
        stroke: new ol.style.Stroke({
            color: '#3399CC',
            width: 4.5,
            lineDash: [4, 8]
        }),
    })
})
```

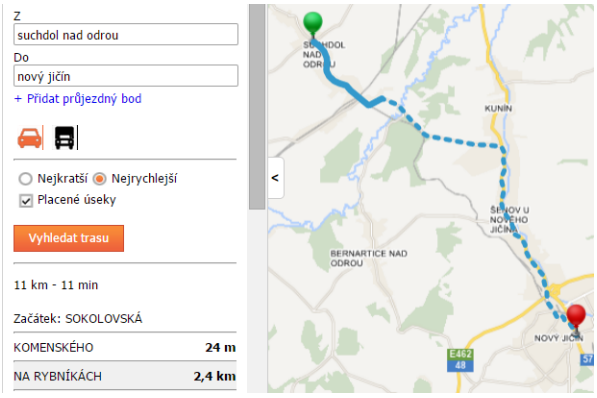
Výpis 4: Definice stylu vrstvy

Jak jsme zvyklí u známých mapových serverů, stačí pouze přesunout některý z bodů trasy někam jinam a trasa se automaticky přeroutuje a zobrazí se nová. Zde v mapě to funguje úplně stejně. Zajišťuje to událost, která zaregistruje puštění levého tlačítka myši nad mapou, přičemž je ale zjištěno, zda se před puštěním tlačítka skutečně přesouval objekt Feature po mapě nebo ne.

5.1.2.2 Itinerář

Na webovém serveru je uložen soubor, ve kterém jsou informace z NavTequ (NavTeq je poskytoval elektronických mapových dat) o tom, jak se mapují ID silničních segmentů na silniční úseky a názvy silnic – jeden silniční úsek vždy obsahuje alespoň jeden segment. Při startu aplikace na webovém serveru je tento soubor načten do slovníku, kde klíče jsou ID segmentů.

Když se poté ze služby routingu vrací kolekce trasových segmentů, každý z těchto segmentů má přiřazeno unikátní ID a ze slovníku se pomocí těchto ID zjistí, kterým segmentem jedna silnice končí a druhá začíná a zaznamená se název silnice, která skončila, kilometráž daného úseku a také údaj o placenosti úseku, všechna tato data se posléze zobrazí v řádku itineráře.



Obrázek 9: Zvýraznění úseku trasy

5.1.2.3 Zvýraznění úseku

Navíc se během průchodu kolekcí segmentů získaných ze služby k jednotlivým silničním úsekům a jejich datům zaznamenávají souřadnice cílového bodu každého segmentu, takže se získá celá geometrie daného úseku a ta se poté pomocí JavaScriptu vloží do skrytého pole u daného řádku itineráře.

Když poté uživatel přejede myší přes některý řádek itineráře, načtou se všechny souřadnice ze skrytého pole do geometrie Linestring a objekt Feature s touto geometrií se zobrazí na mapě. Vrstva, do které je přiřazen, má definován styl s čárou větší šířce, než je šířka čáry ve vrstvě pro vykreslení trasy, a nepřerušovanou, takže jde dobře vidět, o který úsek trasy jde.

Když pak uživatel z položky itineráře myší odjede, obsah mapové vrstvy se vymaže a celá trasa zůstane vykreslená opět přerušovanou čarou.

5.1.3 Měření

Třetí část panelu slouží primárně k měření. Podobně jako mapový server Mapy.cz poskytuje systém měření vzdálenosti, ale oproti němu poskytuje i měření výměry oblasti.

Abychom ale mohli něco změřit, musíme si v mapě vytvořit útvar (polygon/lomenou čáru). Vytváření oblastí a čar na mapě je v doslovném překladu terminologie objektů knihovny OL interakce s mapou, neboť se jedná o objekt `ol.interaction.Draw`. A jde opravdu o interakci, neboť uživatel přidává další body tím, že jezdí myší po mapě a kliká a na mapě tímto vzniká buď vyplněný polygon v případě měření výměry oblastí nebo prostá čára v případě měření vzdálenosti, záleží na parametru `type` konstruktoru objektu Draw. Pokud je typem `Polygon`, kreslí se mnohoúhelník, pokud `LineString`, kreslí se lomená čára.

5.1.3.1 Oblast

Pokud měřím oblast, výsledek se objeví až tehdy, když se polygon uzavře. Je to logické, protože bez uzevřeného polygonu není definován útvar, jehož výměra se má měřit.

Uživatel by měl sice nakreslit polygon tak, aby se jeho hrany nijak nekřížily, nicméně výpočet obsahu polygonu jedno překřížení "toleruje"– v místě křížení vytvoří nový bod (který tam původně nebyl) a souřadnice vedlejších bodů obou křížených čar (pouze na jedné straně od křížení) se vymění. Takto se dosáhne výsledku, že už se polygon nikde nekříží a výpočet jeho výměry může proběhnout. Jakmile je polygon dokončen, zavolá se metoda z knihovny geographicLib, která spočítá jeho výměru ze zadaných bodů obrysu a výsledek vypíše do panelu. Spolu s výsledkem se zobrazí také dvě tlačítka - na smazání a uložení oblasti. Pokud stisknu tlačítko na smazání, vyhledá se odpovídající objekt Feature v mapové vrstvě podle pomocné property `id` a vyjme se z ní a tím se také vymaže z mapy.

Pokud chci oblast uložit, objeví se okno, ve kterém definuji její název a barvu a uložím. Tak se oblast se vším, co jsem jí definoval, uloží do databáze a zároveň se jí na mapě změní barva na barvu zvolenou při ukládání a doprostřed se umístí její název.

5.1.3.2 Vzdálenost

Vzdálenost se začíná měřit hned, jakmile je co měřit – jakmile se vytvoří první dva body a mezi nimi čára, Pro výpočet vzdálenosti je opět použita metoda z knihovny geographicLib.

Geometrie LineString má při kreslení výbornou vlastnost – když vymažu některý její bod, tento bod se hned vymaže i z mapy a novou čárou se spojí dva body, které jsou před a za tím vymazaným a zároveň se přepočte celá délka změněné čáry od smazaného bodu až do konce. Takto lze zpětně zrušit a z mapy vymazat i celou nakreslenou čáru.

5.2 Horní panel

Další úkol spočíval v přidání záložek POI objektů a oblastí do panelu nad mapou a jejich propojení se zaškrtávacími políčky v TreeListu v levém panelu a s mapou. Když se zaškrtne v TreeListu checkbox u oblastí nebo u POI, objeví se odpovídající záložka v panelu nad mapou a po odškrtnutí opět zmizí.

V obou těchto záložkách, podobně jako v číselnících, je DevExpress komponenta gridview. Oproti číselníkům je tady ale rozdíl – oblasti ani POI se nedají upravovat nebo mazat, dají se pouze zobrazovat. A zobrazí se tak, že se zaškrtne v gridview políčko na odpovídajícím řádku.

Pokud se jedná o POI, pak se na tento pokyn vytáhnou z tabulky zobrazené GPS souřadnice, transformují se a přidají se jako geometrie objektu Feature, který má tentokrát jako styl obrázek modré bubliny. A pokud se jedná o oblast, pak se vytáhne pouze ID oblasti v daném řádku (jelikož souřadnice jsou u oblasti uloženy jako řetězec s GPS souřadnicemi všech bodů, jednak by se do tabulky nevešly a jednak by se v ní zobrazovaly nepěkně) a pomocí něj se z databáze vytáhne celý řetězec všech GPS souřadnic, název oblasti a barva. Souřadnice se v JavaScriptu rozdělí na jednotlivé páry a ty se poté přiřadí geometrii Polygon (jelikož jde o mnohoúhelník). Feature s touto geometrií dostane název a do stylu barvu a zobrazí se na mapě.

Odškrtnutí políčka v gridview znamená, že se odpovídající POI nebo oblast odebere z pole objektů Feature v dané vrstvě a tím se vymaže z mapy.

Obrázek 10: Geolokace nad mapou

5.3 Geolokace

Následujícím úkolem bylo zprovoznit geolokaci nad mapou. Uživatel pohybuje myší nad mapou a když myš zastaví, mají se zobrazit data o místě, na které myš ukazuje – konkrétně jeho adresa a maximální povolená rychlost v tom místě.

K tomu jsem využil JavaScriptovou metodu `setInterval` – tato metoda spustí uvedenou funkci vždy po uvedeném časovém intervalu. Aby ale neběžela pořád a nezpomalovala tak celý systém, zapíná se až při stisknutí tlačítka "Maximální rychlost"nebo "Adresa"a pokud se obě tlačítka vypnou, vypne se i metoda setInterval, funguje tedy pouze pokud je některé z tlačítek aktivní.

Při zavolání funkce v metodě setInterval se získají souřadnice na mapě, kam ukazuje myš, a pomocí AJAXu se pošlou do Controlleru. Controller je přepošle geolokační službě a ta mu vrátí data, ze kterých si Controller vybere pouze označení silnice, město a stát a maximální rychlost a tyto údaje pošle zpátky do JavaScriptu jako AJAX response. JavaScript je poté zpracuje a zobrazí nad mapou.

5.4 Optimalizace

Posledním úkolem bylo vytvořit v systému rozhraní pro optimalizátor a konfigurátor optimalizátoru. Optimalizátor pracuje v první řadě s objednávkami – při optimalizaci se přiřazují vozidlům na okružních jízdách. Každá objednávka má své povinné parametry, mezi ně patří datum, údaj o vyřízenosti objednávky, adresa a GPS souřadnice. Dále má každý zákazník možnost si k objednávkám definovat vlastní parametry.

Protože se v optimalizaci pracuje i s hmotností zakázky a s časovými okny zákazníků a protože tyto hodnoty nejsou mezi povinnými parametry objednávky, je třeba definovat, které volitelné parametry objednávky daného zákazníka je budou reprezentovat – toto přiřazení parametrů (který volitelný parametr bude brán za tonáž a který za časové okno) zákazník provede v konfigurátoru.

Druhým objektem, se kterým se v optimalizaci pracuje, jsou vozidla. Jelikož ale zákazník nemusí chtít, aby se všechna jeho vozidla účastnila rozvozů, může u jednotlivých vozidel nastavit příznak, zda je dané vozidlo aktivní pro optimalizaci. Poté se do optimalizace berou všechny vozidla, která mají tento příznak nastaven na `true`.

Jakmile jsou připraveny všechny potřebné údaje o objednávkách, může se začít optimalizovat. Optimalizátor poskytuje dvě funkce:

- Ruční plánování
- Automatické plánování

Pokud uživatel zvolí ruční plánování, z objednávek se vyberou pouze označené, vybere se označené vozidlo a pro toto vozidlo proběhne naplánování rozvozu, jehož výsledek se zobrazí na mapě a mezi trasami.

Pokud uživatel zvolí automatické plánování, opět se vyberou pouze označené objednávky, k nim se ale vyberou tentokrát všechna vozidla a pro tato data proběhne optimalizace, jejíž výsledek se rovněž zobrazí na mapě a mezi trasami.

6 Závěr

Cílů této práce, které byly vytyčeny na začátku a později v průběhu implementace, bylo většinou úspěšně dosaženo. V každé etapě zadané práce jsem se snažil odvést maximální výkon ke spokojenosti mé i zadávajícího pracovníka a již při implementaci počítat s tím, že, jak se říká v IT světě, "Uživatel je schopen všeho". Samozřejmě se vždy najde nějaká chyba, ale to je u softwarových produktů běžnou praxí a proto bývají vydávány opravné balíčky (tzv. Service pack) nebo záplaty (tzv. Patch). I já počítám s tím, že má práce na tomto systému ještě neskončila, že budou v budoucnu probíhat opravy a úpravy a že se budou některé části systému ještě dokončovat.

Během své práce na modulu jsem získal mnoho nových poznatků a zkušeností, ať už z hlediska programování nebo co se týče nově poznaných technologií. Dostal jsem se k práci na reálném projektu, jehož výsledek bude zanedlouho sloužit reálným zákazníkům, což je pro mě velmi dobrá zkušenost. Osobně mohu potvrdit, že platí známé pořekadlo "Praxe je nejlepší učitelka"a že vědomosti a schopnosti získané z praxe jsou k nezaplacení.

7 Reference

[1] *Home · jsprit/jsprit Wiki · GitHub* [online]. [cit. 2015-05-01]. Dostupné z: https://github.com/jsprit/jsprit/wiki, 1. 5. 2015

[2] *User Manual · npgsql/npgsql Wiki · GitHub* [online]. [cit. 2015-05-01]. Dostupné z: https://github.com/npgsql/npgsql/wiki, 30. 4. 2015

[3] RINKAI ROUTING. *Rinkai Routing - Rinkai s.r.o.: Produkty* [online]. [2009] [cit. 2015-04-27]. Dostupné z: http://www.rinkai.cz/products.aspx

[4] Zobrazení všech naplánovaných tras. *Rinkai Routing - Rinkai s.r.o.: Produkty* [online]. [2009] [cit. 2015-04-27]. Dostupné z: http://www.rinkai.cz/products.aspx

[5] Report využití vozidel. *Rinkai Routing - Rinkai s.r.o.: Produkty* [online]. [2009] [cit. 2015-04-27]. Dostupné z: http://www.rinkai.cz/products.aspx Název obrázku. In: Název zdroje [typ média]. Vlastník webové stránky a rok vydání (nebo copyright). [datum citování]. Dostupnost.

[6] DIGITECH. *Plánování trasy a optimalizace rozvozových tras | PLANTOUR* [online]. [cit. 2015-04-27]. Dostupné z: http://www.digitech.cz/plantour, 27. 4. 2015

[7] RADA, Václav. *CW05 - TEORIE ROZHODOVACÍCH PROCESŮ, CW13 - LOGISTIKA* [online]. [2014] [cit. 2015-04-25]. Dostupné z: www.fce.vutbr.cz/tst/rada.v/ROZHPROC/w-cw05-rpr-pr26-tspcli.ppt

[8] MÜLLEROVÁ, Michaela, PODRUH, Michal. *Problém obchodního cestujícího a příbuzné úlohy* [online]. [2007] [cit. 2015-04-25]. Dostupné z: www.fd.cvut.cz/projects/k611x1p/VRP.pps

[9] *PostgreSQL: About* [online]. [1996] [cit. 2015-05-02]. Dostupné z: http://www.postgresql.org/about/

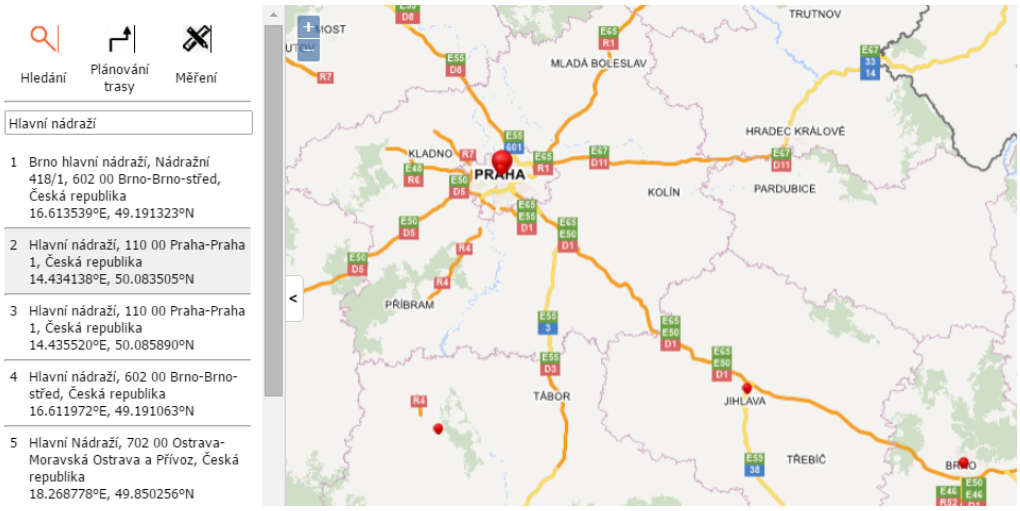
[10] MICROSOFT. *Learn ASP.NET Online – Microsoft Virtual Academy* [online]. [cit. 2015-05-03]. Dostupné z: http://www.microsoftvirtualacademy.com/training-courses/introduction-to-asp-net-mvc

[11] W3CSCHOOLS. *ASP.NET Razor C# Syntax* [online]. [cit. 2015-05-03]. Dostupné z: http://www.w3schools.com/aspnet/razor_syntax.asp

[12] PENATE, Rolando. *OpenLayers 3.0 Released! - Boundless* [online]. [2014] [cit. 2015-05-04]. Dostupné z: http://boundlessgeo.com/2014/08/openlayers-3-0-released/

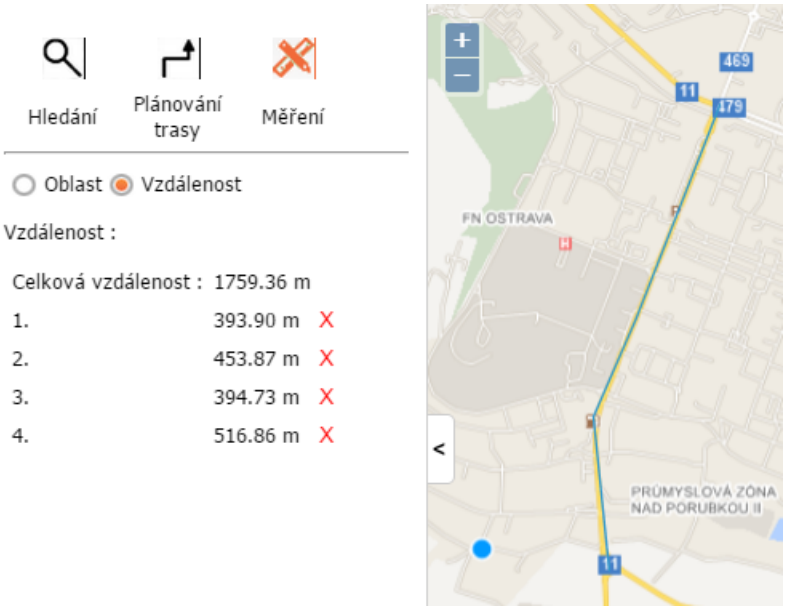
[13] ANDERSON, Rick. *Adding a Controller | The ASP.NET Site* [online]. [2013] [cit. 2015-05-02]. Dostupné z: http://www.asp.net/mvc/overview/getting-started/introduction/adding-a-controller

[14] DEVEXPRESS. *About Us, Our Mission* [online]. [cit. 2015-05-05]. Dostupné z: https://www.devexpress.com/Home/Mission.xml

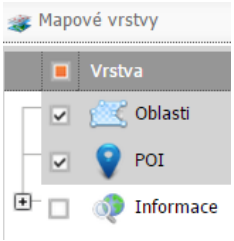


Obrázek 11: Zobrazení výsledků vyhledávání s interakcí

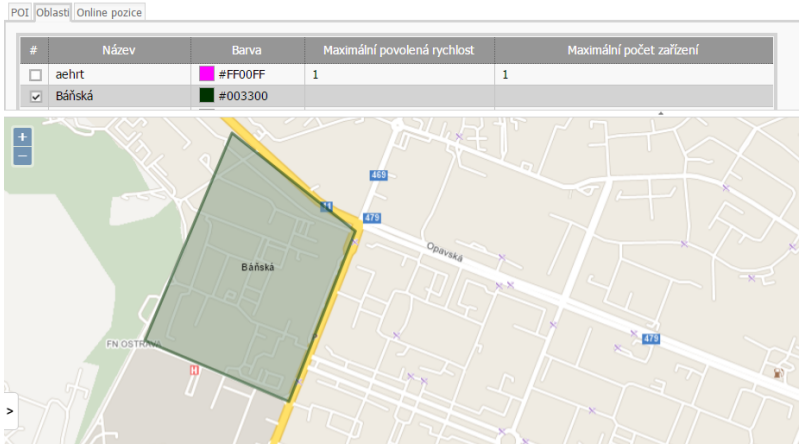
Obrázek 12: Oblast na mapě uložení do databáze



Obrázek 13: Změřená trasa s jednotlivými úseky



Obrázek 14: TreeList



Obrázek 15: Vykreslení vybrané oblasti na mapě