TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: X2612 – Elektrotechnika a informatika
Studijní obor: 1234T567 – Název studijního oboru

Webová komponenta pro práci s digitálními mapami

Web component for digital maps

Diplomová práce

Autor: **Bc. Jan Kuta**
Vedoucí práce: prof. Ing. Zdeněk Plíva, Ph.D.

V Liberci 27. 12. 2010

Prohlášení

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

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce.

Datum

Podpis

Poděkování

Chtěl bych touto cestou poděkovat panu prof. Ing. Zdeňku Plívovi, Ph.D. za to, že mi byl ochoten vést diplomovou práci na mé vlastní téma a také za jeho příkladné vedení, zejména bych chtěl pochválit jeho pohotové reakce na elektronické dotazy.

Dále bych chtěl poděkovat své přítelkyni za pomoc se závěrečnými úpravami a korekturami textu. Také bych jí chtěl poděkovat za její motivaci ve chvílích, kdy se mi zrovna pracovat nechtělo.

Abstrakt

Česky

Diplomová práce se věnuje tvorbě komponenty pro webové aplikace sloužící k ukládání, editaci a zobrazování dat v digitální mapě od společnosti Google. Data, se kterými komponenta pracuje, znázorňují trasy nebo místa zájmu (tzv. POIs – z anglického points of interest). Komponenta tato data ukládá v datovém úložišti. Jako výchozí datové úložiště používá databázový systém MySQL, ale je připravena na zavedení podpory i jiných úložišť.

Práce je rozdělena do tří kapitol. V první autor popisuje teoretické základy a seznamuje čtenáře s potřebnou technologií. Dále zde navrhuje základní rozvrstvení komponenty. Další část se zabývá implementací této komponenty. V poslední kapitole vytvořenou komponentu použije v ukázkové aplikaci a popíše možnosti jejího nastavení.

Klíčová slova:

Webová komponenta, Mapy, Google Maps JavaScript API Verze 3, AJAX, PHP

Abstarct

English

This graduation theses attends to create a component for web aplications, which should be used for storing, editing and for displaying data in digital maps from Google company. The data, which is the component working with, displays routes or points of interest (POIs) and later are saved in data storage. Default data storage is database system My SQL, but the component is already prepared for applying of other storages.

Thesis is divided to three chaptors. In the first one author describes theoretical background and introduces the reader with the necessary technology. Furthermore, there is suggested a basic stratification of the component. The component is used in a sample application and it is described possibilities of settings in the last chapter.

Keywords:

Web component, Maps, Google Maps JavaScript API Version 3, AJAX, PHP

Obsah

ÚVOD

Pro svou diplomovou práci jsem si vybral téma tvorby webové komponenty pro práci s digitálními mapami. Důvodem, proč jsem si vybral toto téma, bylo vydání nové (již třetí) verze Google Maps JavaScript API v květnu roku 2010. Tato technologie mě velice oslovila, a tak jsem s ní začal trochu experimentovat. Zajímal jsem se o její případné možnosti využití, a tak jsem objevil i pár velice povedených webových aplikací[1] využívajících druhou verzi Google Maps JavaScript API. Po bližším zkoumání jsem zjistil, že obě jsou postaveny na produktu Maptoolkit německé společnosti Toursprung[2]. Využívání tohoto produktu je ale zpoplatněno, marně jsem se pokoušel najít nějakou alternativu, která by byla zdarma. V tu chvíli mě napadlo, že bych se o něco podobného mohl pokusit sám.

Cíle

Cílem této diplomové práce je vytvořit webovou komponentu pro práci s digitálními mapami, která by mohla být neplacenou alternativou zpoplatněného nástroje Maptoolkit od německé společnosti Toursprung. Na vyvíjené komponentě si ukážeme základní principy pro vytváření webových komponent a seznámíme se s potřebnými technologiemi. Hotová komponenta bude následně použita v ukázkové webové aplikaci.

Členění práce

Práce se dělí do tří kapitol. První kapitola se zaměřuje na teoretické znalosti potřebné pro vyvíjení webových komponent. Navrhneme si zde základní rozvrstvení komponenty a seznámíme se s potřebnými technologiemi.

Druhá kapitola pojednává o implementaci komponenty. Podrobně se proberou jednotlivé vrstvy a vysvětlí se jejich význam. V této kapitole se podíváme i na některé zasadní úseky kódu.

V poslední kapitole se naučíme vytvořenou komponentu začlenit do webové aplikace a nastavovat ji dle vlastních požadavků.

---

1 Toursprung. *Bikemap.net - Tvoje trasy online* [online]. [cit. 2010-12-27]. URL: <http://www.bikemap.net/>.
SPD. *RADFALLE Konstanz* [online]. [cit. 2010-12-27]. URL: <http://konstanz.radfalle.de>.

2 Toursprung. *Toursprung* [online]. [cit. 2010-12-27]. URL: <http://www.toursprung.com/>.

1 Teoretická část

V této kapitole se podíváme na návrh webové komponenty pro práci s mapami. Nejprve si povíme něco málo o aktuálních trendech v oblasti návrhu. Načrtneme si návrh naší komponenty v jejich duchu a seznámíme se se všemi technologiemi, které k vývoji budeme potřebovat. Podíváme se na jejich vlastnosti a naučíme se je používat.

1.1 Maptoolkit

Produkt Maptoolkit je vzorem pro naši vvvíjenou komponentu. Proto se na něj nyní trochu zaměříme, abychom později mohli posoudit, jak moc se od něho naše komponenta liší.

Maptoolkit je postaven jako JavaScriptové API[3]. Vývojář ho využívá tak, že volá rozhraním nabízené javaScriptové funkce a nastavuje různé parametry, čímž získá požadovanou funkčnost. Komponenta podporuje přibližně dvacet světových jazyků včetně češtiny.

Dále je nabízena uživatelům rozsáhlá databáze bodů zájmu (tzv. POIs) počínaje fotkami, památkami, aktuálním počasím až turistickými trasami konče. Samozřejmostí je možnost přidávání vlastních bodů zájmu. Body lze v mapě full-textově vyhledávat nebo určité body vyfiltrovávat. Pokud je na mapě příliš velké množství bodů na malém území, dochází automaticky ke klastrování (shlukování bodů pod jeden zástupný).

Pro přidávání vlastních bodů zájmu je pro uživatele nápomocný editor geodat. Pokud například požadujeme vložení nějaké oblíbené trasy, spustíme tento editor a v mapě danou trasu nakreslíme. Je zde i podporovaná funkce přichytávání trasy k cestám a dopravním komunikacím.

Administrátorská úprava nastavení komponenty probíhá pomocí několika málo kliknutí v průvodci nastavení.

Maptoolkit je optimalizován pro vysoký počet bodů zájmu a vysokou návštěvnost. Odborně řečeno je velice dobře škálovatelný (to znamená, že je zaručena schopnost práce programu i při významném vzrůstu vytížení[4]). Díky tomu se nestává, že

3 Application Programming Interface – rozhraní pro programování aplikací

4 ProZ.com. *Překladatelské služby, poptávky po překladech a nezávislí překladatelé* [online]. [cit. 2010-12-27]. URL: <http://ces.proz.com/kudoz/czech_to_english/computers:_software/2776676-%C5%A0k%C3%A1lovateln%C3%BD.html>.

by pod návalem žádostí aplikace zamrzala. Mimojiné je toho docíleno různými cachovacími algoritmy.

1.2 Návrh

Při návrhu komponenty by se mělo postupovat v duchu moderního programování (tj. víceúrovňový návrh a s tím spojené oddělení úrovní), aby byla komponenta flexibilnější, snadno konfigurovatelná a upravovatelná.

Víceúrovňový návrh, též známý jako vícevrstvý návrh nebo vícevrstvá architektura, se v poslední době stal mezi vývojáři rozšířený a hojně využívaný při vývoji středně velkých a velkých projektů, kde dodržování jeho pravidel přináší mnoho pozitiv, mezi jinými například:

- větší přehlednost kódu
- lepší udržovatelnost
- větší flexibilita kódu
- vývoj lze rozdělit mezi více vývojářů (každý z nich se může specializovat pouze na určitou část vývoje)
- jednodušší zásahy do projektu (dodatečné úpravy a změny)

To je ovšem na druhou stranu vykoupeno nutností psát více kódu, proto se tento způsob většinou nepoužívá u malých jednoúčelových projektů, ve kterých se vývojář dobře orientuje a dodržování pravidel víceúrovňového návrhu by pouze prodlužovalo vývoj.

My budeme vyvíjet komponentu podle klasického čtyřvrstvého modelu (viz [1], s. 72-89):

- Datové úložiště: slouží k uchovávání dat, jedná se většinou o nějakou relační databázi. U statických projektů (tj. tam, kde se data téměř nemění) se pro uložení dat mohou používat XML-soubory, případně obyčejné textové soubory.
- Vrstva přístupu k datům (Data Access Layer): obsahuje kód pro získávání dat z datového úložiště a manipulaci s nimi (vkládání, úprava, mazání). Tato

vrstva zabaluje data získaná z datového úložiště do objektové podoby. Stále se však jedná o surová data. Přestože jsou zabalena do objektů, nic dalšího (jako např. nějaká logika) k nim není přidáno. Tyto objekty se dále posílají vrstvě aplikační logiky.

- Vrstva aplikační logiky (Business Logic Layer): transformuje získané objekty od předchozí vrstvy do abstraktnější a intuitivnější podoby. Tato podoba skrývá podrobnosti na nižší úrovni (např. schéma uložení dat v datovém úložišti atd.) a přidává veškerou validační logiku, která zajišťuje bezpečný a konzistentní vstup dat. Dále se na této úrovni přidávají odvozené vlastnosti a metody pro snazší a intuitivnější práci.
- Prezentační vrstva (User Interface): (též uživatelské rozhraní) definuje, jakým způsobem získaná data (objekty) reprezentovat na obrazovce (např. v podobě různých animací, formátovaných tabulek atd.)

V jiných zdrojích byste mohli nalézt také třívrstvou architekturu, jedná se však o stejnou věc, pouze datové úložiště a vrstva přístupu k datům se uvádí jako jediná datová vrstva[5].

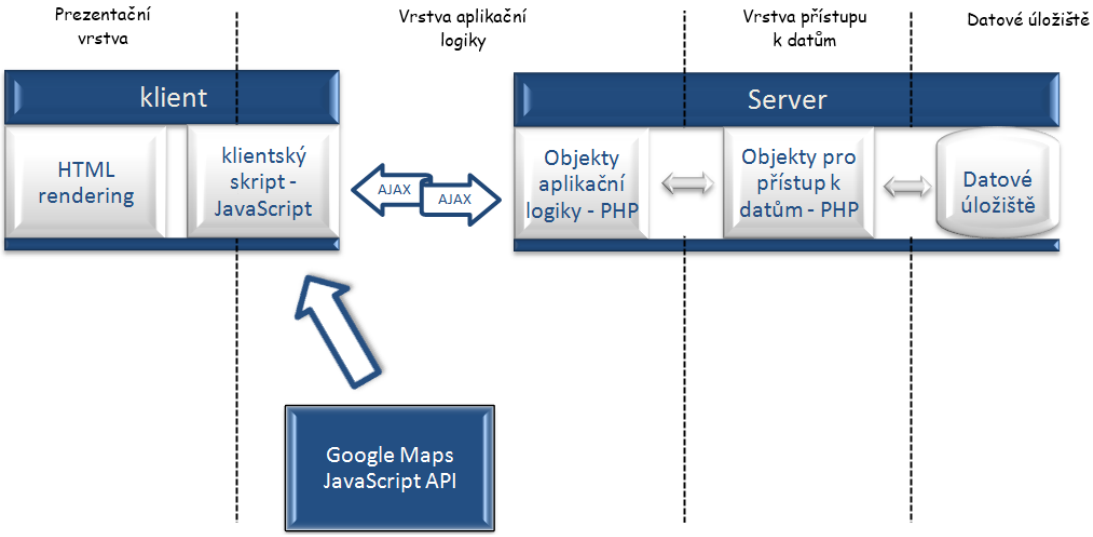
S vrstvením programu také blízce souvisí nutnost oddělení úrovní. Jednotlivé úrovně by měly být uzavřenými celky, které se sousedními vrstvami komunikují pouze přes předem definované rozhraní. Tím se projekt rozdrobí na menší části, na kterých se lépe pracuje.

U velkých projektů se ještě mohou jednotlivé úrovně rozdělovat. Naopak u malých projektů se některé vrstvy slučují, obyčejně se slučuje např. vrstva přístupu k datům s vrstvou aplikační logiky.

Naše komponenta se bude používat na webových stránkách. Tím se rozdělí na dvě části. Jedna se bude vykonávat na straně klienta (prezentační vrstva a část vrstvy aplikační logiky) a ta druhá na straně serveru (zbylé vrstvy). Ke komunikaci mezi těmito částmi budeme používat protokol HTTP, konkrétně pro ni využijeme technologii AJAX (viz kapitola 1.2.3). Pro názornější představu o rozdělení a rozvrstvení

5 ZONER software, a.s. *Používáme návrhové vzory v .NET - Abstract Factory | Interval.cz* [online]. [cit. 2010-12-27]. URL: <http://interval.cz/clanky/pouzivame-navrhove-vzory-v-net-abstract-factory/>.

komponenty se podívejme na následující obrázek.



Obr. 1: Rozdělení a rozvrstvení komponenty (zdroj: autor)

Na straně klienta se nachází prezentační vrstva. Její základem je renderovací jádro prohlížeče, které na základě došlého HTML-dokumentu a souboru popisující jeho styly data vykreslí na obrazovku. U klienta se ještě dále vykonává část aplikační logiky prostřednictvím klientských skriptů psaných v jazyce JavaScript. Definují se v nich reakce komponenty na podněty (akce) vyvolané uživatelem. Ve velké míře se zde pro práci s mapami využívá JavaScriptové API od společnosti Google.

Zbytek komponenty se nachází na serveru. Pro serverové skripty je použit jazyk PHP. Pro komunikaci mezi klientskou a serverovou částí vrstvy aplikační logiky se používá technologie AJAX, která je založena na zasílání požadavků a odpovědí přes protokol HTTP. Vysoce abstraktní objekty aplikační logiky komunikují s těmi primitivnějšími určenými pro přístup k datům. Objekty pro přístup k datům získávají data z datového úložiště prostřednictvím dotazovacích jazyků (např. SQL). Na obrázku je znázorněno sice datové úložiště na stejném serveru jako dvě předchozí vrstvy, ale nemusí tomu nutně tak být a může se nacházet na zvláštním serveru.

Než přistoupíme k samotné implementaci, musíme se naučit požívat všechny výše zmíněné technologie.

1.2.1 JavaScript

JavaScript se řadí mezi interpretované[6] dynamicky typované[7] skriptovací programovací jazyky. Syntakticky se řadí do velké skupiny "céčkovských" jazyků[8]. Skripty jsou prováděny ve webovém prohlížeči na straně klienta. Bohužel pro prohlížeče neexistuje žádný standard, který by při práci s JavaScriptem musely dodržovat, ba hůře existují dvě skupiny prohlížečů (prohlížeče Internet Explorer a ty ostatní), které využívají různé postupy. Vytvořit kód kompatibilní s oběma tak stojí značné úsilí. To spolu s komplikovaným laděním kódu znesnadňuje vývoj. Naštěstí existují nástroje, které nám tyto překážky pomáhají překonat.

Problém s laděním kódu se nechá snadno vyřešit použitím debuggeru JavaScriptu. Jedním z nejuznávanějších je rozšíření pro prohlížeč Firefox s názvem FireBug (existují i odlehčené verze pro ostatní prohlížeče). Kromě zmíněného debuggeru navíc obsahuje řadu nástrojů usnadňujících vývoj webových projektů jako například inspektor HTML-kódu, který dovoluje i editaci přímo za běhu, dále nástroje pro sledování síťové aktivity, pro zobrazení DOM[9], pro hlášení chyb a mnoho dalších nástrojů. Vřele doporučuji FireBug nainstalovat[10] a používat, určitě si ho zamilujete.

Co se týče problému JavaScriptu a jeho kompatibility s prohlížeči, máme několik možností. Buď vyvineme značné úsilí pro napsání kompatibilního kódu využitím větvení pro jednotlivé skupiny prohlížečů, nebo použijeme nějaký Framework pro JavaScript, jenž vše vyřeší za nás. U mě jednoznačně vyhrává druhá alternativa. Frameworků v dnešní době ovšem existuje celá řada. Po přečtení různých porovnání[11] jsem došel k závěru, že nejlepší volbou je JQuery. Zaprvé má velkou fungující komunitu, dále propracovanou dokumentaci a také řadu tutorialů[12], z nichž můžeme čerpat inspiraci.

6 kód překládá postupně interpretr až v době jeho spuštění
7 proměnná může za svůj život obshovat data různých datových typů (integer, string atd.), typová kontrola probíhá až za běhu
8 dále sem patří Java, C, C#, PHP atd.
9 objektový model dokumentu (Document Object Model)
10 stáhnout ho můžete na http://www.getfirebug.com/
11 Wikimedia Foundation, Inc. *Comparison of JavaScript frameworks - Wikipedia, the free encyclopedia* [online]. [cit. 2010-12-27]. URL: <http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks>.
mootools.net. *SlickSpeed Selectors Test* [online]. [cit. 2010-12-27]. URL: <http://mootools.net/slickspeed/>.
12 The jQuery Project. *JQuery JavaScript Library* [online]. [cit. 2010-12-27]. URL: <http://docs.jquery.com/Main_Page>.

JQuery využijeme zejména pro práci spojenou s obsluhou událostí a k asynchronnímu zasílání požadavků na server (viz kapitola 1.2.3). Nyní si ale představme způsob, jakým budeme simulovat v JavaScriptu objektové programování, na které jsme zvyklí. V JavaScriptu jsou totiž objekty podporovány na velice nízké úrovni. Spíše než objektům se podobají asociativním polím[13], která mohou v sobě obsahovat jak proměnné, tak i odkazy na funkce a objekty. Třídy můžeme definovat zápisem konstruktoru, čímž můžeme vytvářet instancí třídy kolik jen chceme (viz následující ukázka kódu).

```
1    function auto(nazev, text) {
2      var text = text;
3      var getText = function(){ return text;};
4      this.nazev = nazev;
5      this.nastartovat = function() {
6        document.write(this.nazev + getText());
7      }
8    }
9
10   var bmw = new auto("BMW X5", " nastartovalo!");
11   var fiat = new auto("Fiat Punto", " nenastartovalo!");
12   bmw.nastartovat();
13   document.write(fiat.nazev+fiat.text);
```

Kód 1: Jednoduchá ukázka objektu v JavaScriptu[14]

V tomto příkladu vidíme všechny možnosti, které nám JavaScript pro objekty nabízí. Na prvním řádku se nachází předpis konstruktoru, který později voláme na řádku 10 a 11. Na druhém řádku vidíme soukromou vlastnost[15], která se inicializuje hodnotou druhého parametru předaného konstruktoru. Na dalším řádku se vytváří soukromá metoda[16], která pouze navrací již zmíněnou soukromou vlastnost (`text`). Všimněte si volání soukromé vlastnosti. Narozdíl od volání té veřejné se používá pouze její jméno, stejně je tomu tak i u volání soukromých funkcí (viz řádek 6). Na čtvrtém řádku máme ukázku veřejné vlastnosti. Uvnitř objektu se k ní přistupuje zápisem `this.verejna_vlastnost` (viz řádek 4 a 6), zvenčí objektu pomocí zápisu `jmeno_objektu.verejna_vlastnost` (viz řádek 13). Na pátém řádku začíná

13 pole k jehož položkám se přistupuje pomocí jména

14 převzat a podle potřeby upraven z:
Tomáš Bobek. *JavaScript - 11. lekce - JavaScript, formulář, DOM, document, object, model, metody, funkce, JS* [online]. [cit. 2010-12-27]. URL: <http://programujte.com/?akce=clanek&cl=2008062900-javascript-11-lekce>.

15 je přístupná pouze uvnitř objektu

16 volání příkazu "fiat.getText();" by vyvolalo výjimku říkající, že metoda neexistuje

definice veřejné funkce. Stejně jako u veřejné vlastnosti k ní přistupujeme uvnitř objektu přes ukazatel `this`[17] a zvenčí přes název objektu (viz řádek 12). Na posledním řádku se pokoušíme vypsat veřejnou a soukromou vlastnost objektu fiat. Kdybychom skript spustili, zjistili bychom, že zápis `fiat.text` vyvolá výjimku (vlastnost není definována).

Druhou možností je vytváření objektů přes anonymní třídy. U tohoto způsobu zápisu ztrácíme možnost vytvořit více instancí a zároveň nemůžeme definovat soukromé vlastnosti ani metody.

```
1    var bmw = {
2      nazev : "BMW X5",
3      majitel : {jmeno : "Jan", prijmeni : "Novak"},
4      nastartovat : function() {
5        document.write(this.nazev + " nastartovalo!");
6      }
7    }
8
9    bmw.nastartovat();
```

Kód 2: Ukázka vytvoření instance anonymní třídy

Za povšimnutí stojí třetí řádek, kde do vlastnosti majitel vkládáme anonymní objekt, u vložených anonymních objektů se mohou využívat uzávěry, o nichž bude řeč později. Druhý způsob zápisu se používá v případech, kde by se za normálních okolností použila statická třída (např. při aplikaci návrhového vzoru singleton[18]). Ten první naopak ve všech případech, kdy vyžadujeme soukromé vlastnosti a metody.

Zajímavou specialitou objektového programování v JavaScriptu, jíž budeme hojně využívat, jsou již zmíněné většinou učebnic utajované uzávěry (closures). Opravdu o nich většina výukových materiálů mlčí, ačkoliv se jedná o velice silnou a užitečnou vlastnost JavaScriptu. Jak je obecně známo, JavaScript používá pro uvolňování nepoužívané paměti Garbage Collector. Takovou může být mimojiné paměť po lokálních proměnných ukončených funkcí[19]. Pokud ovšem v takové funkci deklarujeme vnořenou funkci, zachová se kontext vnější funkce (tj. hodnoty všech lokálních proměnných) až do zániku jakéhokoli spojení na tu vnitřní. Jinými slovy kontext vnější funkce může přežít samotnou funkci. Možná se to zdá trochu zamotané, lépe se snad vše pochopí na malém příkladu.

17 this není nic jiného než ukazatel na objekt, ze kterého je příkaz volán
18 představuje řešení problému, kdy v celém programu má běžet pouze jediná instance třídy
19 nezapomeňte, že objekty jsou ve skutečnosti také jen určitý druh funkce

```
1   var globalni_ukazatel_na_vnitrni_funkci;
2
3   function vnejsi_funkce(){
4      var lokalni_promenna_vnejsi_funkce = "Closure funguje";
5      globalni_ukazatel_na_vnitrni_funkci = function(){
6        return lokalni_promenna_vnejsi_funkce;
7      }
8    }
9
10   vnejsi_funkce();
11   globalni_ukazatel_na_vnitrni_funkci();
12   ...
13   globalni_ukazatel_na_vnitrni_funkci = 5;
```

Kód 3: Ukázka využití uzávěru

V příkladu máme globální proměnnou (řádek 1) a jednu pojmenovanou funkci (její definice je na řádku 3). V této funkci jsme nadefinovali lokální proměnnou (řádek 4) a anonymní vnitřní funkci, která se ukládá do globální proměnné. Ona anonymní funkce vrací lokální proměnnou vnější funkce. Tak a teď pozor! Na řádku 10 voláme vnější funkci, čímž se do globální proměnné uloží vnitřní anonymní funkce. Tím vnější funkce končí, ale kontext stále zůstává, protože stále máme odkaz na vnitřní funkci v naší globální proměnné. Na 11. řádku voláme přes globální proměnnou vnitřní funkci, která opravdu navrací hodnotu lokální proměnné vnější funkce (řetězec "Closure funguje"). I po ukončení vnitřní funkce se kontext stále uchovává, ten zanikne až na řádku 13, kde do globální proměnné ukládáme novou hodnotu. Tím se zpřetrhají veškeré vazby na vnitřní funkci a kontext konečně zanikne a lokální proměnná se uvolní z paměti[20].

Tuto techniku budeme využívat při registrování posluchačů na události. Dále se bez ni neobejdeme v případech, kdy chceme ze soukromé metody přistupovat k nějaké veřejné vlastnosti či metodě. Soukromé metody totiž běží ve vnějším kontextu (kontextu objektu abalujícího tuto třídu), tudíž proměnná `this` odkazuje na rodičovskou třídu. Musíme tedy před zavoláním soukromé metody uložit odkaz na naší třídu. Ukažme si to na pozměněném příkladu, se kterým jsme se již setkali dříve (viz kód 1). Nechť soukromá metoda `getText()` vytváří a vrací celý řetězec a metoda `nastartovat()` tento řetězec pouze vypisuje.

20 pokud stále tápete a není vám cokoli ohledně uzávěrů jasné, doporučuji přejít na následující odkaz: Petr Staníček. *Javascript a oblast působnosti proměnných - díl třetí – Zdroják* [online]. [cit. 2010-12-27]. URL: <http://zdrojak.root.cz/clanky/javascript-a-oblast-pusobnosti-promennych-dil-treti/>.

```
1    function auto(nazev, text) {
2      var that = this;
3      var text = text;
4      var getText = function(){ return that.nazev + text;};
5      this.nazev = nazev;
6      this.nastartovat = function() {
7        document.write(getText());
8      }
9    }
10
11   var bmw = new auto("BMW X5", " nastartovalo!");
12   var fiat = new auto("Fiat Punto", " nenastartovalo!");
13   bmw.nastartovat();
14   fiat.nastartovat();
```

Kód 4: Jednoduchá ukázka objektu v JavaScriptu[21]

Pokud si skript spustíme, zjistíme, že vše funguje tak, jak má. Na řádku číslo 2 jsme si uložili do lokální proměnné `that` odkaz na danou instanci třídy auto. V soukromé metodě getText (respektive v anonymní funkci uložené v proměnn getText) máme k této proměnné přístup právě díky uzávěru, a tak přes ní můžeme přistoupit k veřejné vlastnosti `nazev`.

1.2.2 Google Maps JavaScript API[22]

Další technologií, se kterou se blíže seznámíme, je Google Maps JavaScript API (dále jen Google API).

Google Maps je revoluční webová mapová technologie vyvinutá firmou Google, která nadobro změnila podobu webu. Díky ní se začaly ve velké míře objevovat interaktivní mapy (ať už jednoduché orientační výřezy nebo celé aplikace kompletně založené na mapách). Jako většina webových služeb od Googlu používá ve značné míře JavaScript a XML. V dnešní době je na ni postaveno mnoho mapových služeb (např. plánovač tras na stránkách Google Maps[23]) a využívají ji mapy vložené na internetových stránkách třetích stran. Vkádání se provádí právě pomocí Google API. Jedná se o bezplatnou službu pro nekomerční účely a v současné době neobsahuje reklamy (což

21 převzat a podle potřeby upraven z:
Tomáš Bobek. *JavaScript - 11. lekce - JavaScript, formulář, DOM, document, object, model, metody, funkce, JS* [online]. [cit. 2010-12-27]. URL: <http://programujte.com/?akce=clanek&cl=2008062900-javascript-11-lekce>.

22 Google Maps. *Wikipedia : the free encyclopedia* [online]. [cit. 2010-12-27]. URL: <http://en.wikipedia.org/wiki/Google_Maps>.

23 Google. *Mapy Google* [online]. [cit. 2010-12-27]. URL: <http://maps.google.com/>.

se, jak vyplývá z podmínek použití[24], může kdykoli změnit). Dále je v podmínkách uloženo, že mapy musí být veřejně přístupny všem uživatelům bez nutnosti platit poplatky a musí zůstat viditelný odkaz na Google. Pro komerční účely existuje verze Google Maps API Premier, která je zpoplatněna.

Počátky Google Maps sahají někam před říjen roku 2004, kdy firma Google odkoupila společnost Larse a Jense Rasmussenových vyvíjejících mapový software. Původně byl program navržen v C++ a uživatelé si ho museli stáhnout. Larse a Jense napadla myšlenka čistě webového produktu, se kterou přišli do společnosti Google. Ta transformovala aplikaci do webové aplikace Google Maps, která se poprvé objevila na stránkách Googlu 8. února 2005. V tu dobu byla plná funkčnost garantována pouze v prohlížečích Internet Explorer a Mozilla. Podpora pro Operu a Safari byla přidána 25. února 2005. V současnosti je třetí verze oficiálně podporována pouze prohlížeči Internet Explorer od verze 7.0 , Firefox od verze 3.0, Safari od verze 4, Google Chrome a Android (systém pro mobilní zařízení)[25]. V Opeře se vyskytly problémy podporou JavaScriptu (např. se nevykreslovaly v mapách křivky).

V červnu 2005 Google uvolnil první verzi Google Maps API. V červenci se k satelitním pohledům Japonska přidaly i topografické mapy s plánky měst (Roadmaps). 22. července se ještě přidal "Hybridní pohled" (satelitní snímky překryté vrstvou ulic). S přidáním tohoto pohledu se projevila nutnost změny projekce satelitních snímků z platte carrée (jednoduchá délkojevná válcová projekce) do Mercatorovy projekce (úhlojevná válcová projekce)[26], čímž se snížilo zkreslení v zeměpisných šířkách vzdálenějších od rovníku.

2. ledna roku 2006 Google začlenil topografické mapy pro další území (USA, Portoriko, Kanada, Velká Británie atd.). Ve stejné době se začala používat stejná databáze satelitních snímků jako u Google Earth. Začátkem dubna 2006 vyšla druhá verze Google Maps API. 11. června byla začleněna do API služba pro "geokódování" (geocoding capalibities), která slouží k převodu mezi adresami a geografickou polohou (zeměpisná šířka a délka). Vzápětí byla zahájena komerční verze Google API pro

24 Google. *Google Maps/Google Earth APIs Terms of Service - Google Maps API Family - Google Code* [online]. [cit. 2010-12-27]. URL: <http://code.google.com/intl/cs-CZ/apis/maps/terms.html>.

25 Google. *FAQ - Google Maps API Family - Google Code* [online]. [cit. 2010-12-27]. URL: <http://code.google.com/intl/cs-CZ/apis/maps/faq.html#browsersupport>.

26 KRÁSA, Josef. *Geografické informační systémy* [online]. [s.l.], 2006. 85 s. Oborová práce. Fsv ČVUT v Praze. URL: <storm.fsv.cvut.cz/on_line/gisz/Kurz_GIS_skriptum.pdf>.

podnikání. Od tohoto okamžiku se začalo API rychle rozrůstat, zaváděly se postupně další služby a ty stávající se dále vylepšovaly, v mapách přibývaly další informace (o budovách, stanicích metra, aktuálním dopravním toku ve hlavních amerických městech atd.)

Mezi nové služby, které se v roce 2007 zasloužily o masovější používání Google API, bezesporu patří služba Street View (360° pohled na ulice ve velkých městech) a služba pro vyhledávání tras, jejíž starší verze byla dosud dostupná jen na stránkách Google Maps.

V srpnu 2007 se zjednodušilo vkládání map do internetových stránek pro méně technicky zdatné uživatele. Od této chvíle bylo možné vkládání map jednoduchým kopírováním vygenerovaného HTML kódu, čímž odpadla nutnost znalosti JavaScriptu. Také byly přidány mapy dalších 54 zemí z Latinské Ameriky a Asie

V květnu 2008 se API převedlo i do flashové (actionscriptové) verze pro webové aplikace postavené na této technologii. V srpnu 2008 bylo předěláno uživatelské rozhraní a byl představen nástroj pro vytváření mapových podkladů. Přidáním služby pro "opačné geokódování" skončila doba převratných změn a nastoupilo období pouhého sběru dalších dat. Služba Street View se rozšířila pro další města v Evropě, Americe a Austrálii. Aktualizovaly se stávající mapy (nové nebo zaniklé silnice, železnice atd.). Toto období bylo přerušeno až v květnu tohoto roku (2010), kdy vyšla třetí verze Google Maps API.

Tato verze je navržená podle vzoru MVC (Model-View-Controller), díky čemuž se zvýšila rychlost běhu aplikací. Nově jsou podporována i mobilní zařízení. V API přibyla služba pro zjištění nadmořské výšky, dále je přidán nástroj pro vytváření vlastních Street View atd. Celé API se snaží o zjednodušení použití. Zrušila se povinnost registrace domény[27] pro získání API-klíče, bez kterého dříve aplikace vůbec nefungovaly. Značného zlepšení dosáhl i přehlednější objektový návrh API.

1.2.2.1 Příklady použití Google API

Google API v naší komponentě budeme využívat celkem ke čtyřem základním věcem. Zaprvé k samotnému zobrazování mapy, dále k přidávání a zobrazování bodů

27 cílové stránky, kde se API bude používat

v mapě, k zobrazování informací v mapě a v poslední řadě k vytváření a zobrazování křivek v mapě. Abychom se poté během implementace v kódu neztráceli a rozuměli mu, předvedeme si na tomto místě pár příkladů. První z nich se zaměřuje na vytváření mapy a na vkládání ukazatelů.

```
1   <html>
2     <head>
3       <title>Google Maps JavaScript API v3 Example: Adding
  marker</title>
4       <script type="text/javascript"
  src="http://maps.google.com/maps/api/js?sensor=false"></script>
5       <script type="text/javascript">
6       var map;
7       function initialize() {
8           var myLatlng = new google.maps.LatLng(-25, 131);
9           var myOptions = {
10              zoom: 4,
11              center: myLatlng,
12              mapTypeId: google.maps.MapTypeId.ROADMAP
13          }
14          map = new
  google.maps.Map(document.getElementById("map_canvas"), myOptions);
15
16          google.maps.event.addListener(map, 'click', function
  (event) {
17              placeMarker(event.latLng);
18          });
19      }
20
21      function placeMarker(location) {
22          var marker = new google.maps.Marker({
23              position: location,
24              map: map
25          });
26
27          map.setCenter(location);
28      }
29      </script>
30    </head>
31    <body onload="initialize()">
32      <div style=" height:100%" id="map_canvas"></div>
33    </body>
34  </html>
```

Kód 5: Vytvoření mapy a vložení ukazatele v Google API

Na řádku 4 vidíme přilinkování knihovnyGoogle API. Na pátém řádku začíná vlastní skript. V inicializační funkci se nejprve zvolí nastavení mapy (řádek 9). Vidíme, že zoom má hodnotu 4, střed mapy odpovídá 25º jižní šířky a 131º východní délky a pro

vykreslení mapy se použije topografická mapa (Roadmap). Na řádku 14 se vytváří instance mapy s danými vlastnostmi, která se zobrazí v div-elementu s identifikátorem "*map_canvas*" (ten se získal pomocí metody DOM[28] getElementByID). Instance se pro další použití ukládá do proměnné *map* (její deklarace se nachází na řádku 6). Příkazem na řádku 16 registrujeme nepojmenovanou funkci, která se má vykonat po vzniku události kliknutí do mapy. V jejím těle se volá funkce placeMarker, která přidá na místo kliknutí ukazatel (řádek 22) a posune střed mapy na ukazatel (řádek 27). Inicializační funkce se volá automaticky po načtení celého dokumentu díky zápisu na 31. řádku.

V dalším příkladu si ukážeme, jak vytvářet křivky, naučíme se používat službu pro hledání cesty mezi dvěma body a podíváme se na použití informačních oken. Protože je příklad delší, rozdělím ho na dvě části.

```
1   <html>
2     <head>
3       <title>Google Maps JavaScript API v3 Example: Directions,
  polylines and
    info windows</title>
4       <script type="text/javascript"
  src="http://maps.google.com/maps/api/js?
                          sensor=false"></script>
5       <script type="text/javascript">
6         var map;
7         var start;
8         var polyline;
9         var directionsService;
10
11        function initialize() {
12          var myLatlng = new google.maps.LatLng(45, 18);
13          var myOptions = {
14            zoom: 4,
15            center: myLatlng,
16            mapTypeId: google.maps.MapTypeId.ROADMAP
17          }
18
19          map = new
  google.maps.Map(document.getElementById("map_canvas"),

    myOptions);
20          start = new google.maps.Marker({
21            position: new google.maps.LatLng(50, 15),
22            map: map
23          });
24          polyline = new google.maps.Polyline({
25            strokeColor: "#FF0000",
```

28 objektový model dokumentu

```
26            strokeWeight: 2
27          });
28          directionsService = new google.maps.DirectionsService();
29
30          google.maps.event.addListener(map, 'click', function
  (event) {
31            ShowDirection(event.latLng);
32          });
33        }
```

Kód 6: Složitější příklad v Google API: část 1 (Inicializace)

Tento výpis by již měl být s našimi znalostmi celkem dobře pochopitelný. Na začátku skriptu se vytvářejí globální proměnné, které se ve funkci `initialize` inicializují (funkce se automaticky spustí po načtení dokumentu). Inicializaci mapy už známe z minulého příkladu, vkládání ukazatele do mapy na řádku číslo 20 by taktéž nemělo činit problém. Dále se zde nastavují barva a šířka křivky (řádek 24). Na řádku 28 se vytváří instance objektu DirectionsService, který slouží k hledání cesty mezi dvěma a více místy. Poslední věcí je registrování funkce `ShowDirection`, která se má spustit jako reakce na událost kliknutí v mapě. Následující výpis ukazuje její předpis.

```
34          function ShowDirection(end) {
35              var request = {
36                  origin: start.getPosition(),
37                  destination: end,
38                  avoidHighways: true,
39                  avoidTolls: true,
40                  travelMode:
  google.maps.DirectionsTravelMode.DRIVING
41              };
42
43              directionsService.route(request, function (result,
  status) {
44                  if (status == google.maps.DirectionsStatus.OK) {
45                      polyline.setPath(result.routes[0].overview_pa
  th);
46                      polyline.setMap(map);
47                      var IW = new google.maps.InfoWindow({
48                          content: result.routes[0].copyrights,
49                          position: end
50                      });
51                      IW.open(map);
52                  }
53              });
54
55          }
56      </script>
57    </head>
```

```
58      <body onload="initialize()">
59        <div style=" height:100%" id="map_canvas"></div>
60      </body>
61    </html>
```

Kód 7: Složitější příklad v Google API: část 2 (ShowDirection)

V této funkci se vytváří požadavek pro vyhledání cesty (začíná na řádku 35). Zadává se výchozí a cílový bod cesty. Vlastnost `avoidHighways` nastavená na hodnotu `true`, omezí výsledek o možnosti využívající dálnice, stejně tak parametr `avoidTolls` o zpoplatněné úseky. Na řádku 43 se požadavek posílá zavoláním metody `route` objektu `DirectionsService`. Metodě se předávají dva parametry. Prvním je již zmíněný požadavek a druhým je anonymní funkce, která se má vykonat po návratu odpovědi. Tato funce má dva povinné parametry (výsledek odpovědi a jeho stav). Předpis funkce vidíme na řádcích 44 až 53. Nejprve v ní kontrolujeme stav odpovědi. Pokud je vpořádku, předá se křivce jako zdroj dat první nalezená trasa (ve výsledku může být totiž celé pole tras). Zápisem na řádku 46 se křivka zobrazí na mapě. Dále vytváříme informační okno obsahující copyright k nalezené trase, zobrazí se v cílovém místě trasy. Na 51. řádku se toto okno otevírá.

Tímto příkladem jsme se seznámili se všemi zbývajícími nástroji, které v komponentě využijeme. Proto můžeme přejít k technologii pro komunkaci mezi klientem a serverem.

1.2.3 AJAX

Otázka technologie komunikace mezi klientskou a serverovou částí mě trápila velice dlouhou dobu. Můj původní záměr psát zbylé vrstvy v jazyce C# za podpory .NET mi značně svazoval ruce. Dlouhou dobu jsem se snažil najít nějaký nástroj pro obousměrné propojení projektů psaných v JavaScriptu a C#[29]. Nějaké možnosti jsem nakonec našel na vývojářském fóru .NET[30] a na osobních stránkach některých vývojářů[31]. Mé pokusy o jejich aplikaci však byly bezvýsledné.

29 Bylo potřeba volat JavaScriptový kód ze C# a naopak

30 Microsoft Corporation. *Calling Javascript from C#.NET* [online]. [cit. 2010-12-27]. URL: <http://social.msdn.microsoft.com/Forums/en-US/netfxjscript/thread/3efc7bf9-730f-47d0-bc9c-7a3550cf1282>.

31 Rick Strahl. *Evaluating JavaScript code from C# - Rick Strahl's Web Log* [online]. [cit. 2010-12-27]. URL: <http://www.west-wind.com/weblog/posts/10688.aspx>.
Roger. *Calling C# from JScript* [online]. [cit. 2010-12-27]. URL: <http://www.differentpla.net/content/2007/04/calling-c-from-jscript>.

Když už jsem ztrácel naději, narazil jsem při prohlubování si znalostí JavaScriptu (viz [2]) na opravdu elegantní řešení. V knize se nachází část věnovaná AJAX. Uvědomil jsem si, že je to vlastně přesně to, co jsem celou dobu hledal.

Jak jsem se již zmínil v úvodu, tuto technologii využijeme pro komunikaci mezi klientskou a serverovou částí, pro niž je stavěná. Běžný mechanismus komunikace webových stránek se serverem je typu požadavek-stránka. Představme si webovou stránku, na níž máme nějakou anketu. Je jasné, že tato anketa zabírá minimální část na stránce. U komunikace typu požadavek-stránka by hlasování v anketě vyvolalo požadavek na server (aktualizovat výsledky ankety). Tím by se přerušila veškerá interakce uživatele se stránkou po celou dobu čekání na odpověď od serveru (doba u vytíženého serveru nemusí být zrovna úplně krátká). Tato odpověď má podobu celé aktualizované stránky, která se vrátí prohlížeči, jenž ji následně zobrazí. Jistě Vám je jasné, že znovunačtení celé stránky kvůli aktualizaci výsledků ankety nacházející se někde v rohu je naprosto zbytečné. Nejenže se tím značně zvyšuje síťový provoz, ale zároveň klesá i uživatelská přívětivost, protože se stránky nebudou chovat plynule.

Namísto toho AJAX je technologie pro vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání. Jak již z názvu vyplývá[32], používá asynchronní typ komunikace se serverem. U výše zmíněného příkladu by stránka po hlasování zaslala serveru pouze požadavek pro navrácení výsledku hlasování. Po celou dobu vyřizování požadavku by zůstala interaktivní. Až by se zpracoval, nahradily by se staré výsledky těmi aktuálními.

Vzhledem k tomu, že naše komponenta bude vysoce interaktivní, a tak bude posílat veliké množství požadavků, je pro zachování plynulosti práce na stránkách využívající naší komponentu AJAX úplnou nutností.

Klíčovou úlohu v AJAX hraje objekt XMLHttpRequest, který v současnosti podporují všechny moderní prohlížeče. Bohužel, jak tomu u prohlížečů bývá, není tato podpora jednotná. Starší verze Internet Exploreru (jmenovitě 5 a 6[33]) ho implementovaly prostřednictvím objektu ActiveX, zatímco nové verze spolu s ostatními prohlížeči už bez něj. Práci s objektem XMLHttpRequest si popíšeme na ukázkovém

32 název AJAX je zkratkou pro Asynchronní JavaScript a XML (v originálu Asynchronous JavaScript and XML)
33 podle [2]

kódu.

```
1   ...
2   try {
3       var requester = new XMLHttpRequest();
4   }
5   catch (error) {
6       try {
7           var requester = new ActiveXObject("Microsoft.XMLHTTP");
8       }
9       catch (error) {
10          var requester = null;
11      }
12  }
13  requester.open("GET", "getPollResult.php?id=1987423", true)
14  requester.onreadystatechange = zkontrolujOdpoved;
15  requester.send(null);
16  ...
17  function zkontrolujOdpoved() {
18      if (requester.readyState == 4) {
19          if (requester.status == 200 || requester.status == 304) {
20              zpracujDoslouOdpoved(requester);
21          }
22          else {
23              alert("Server se nepodařilo kontaktovat.");
24          }
25      }
26  }
27
28  function zpracujDoslouOdpoved(requestObject) {
29      alert(requestObject.responseText);
30  }
```

Kód 8:Ukázka práce s Objektem XMLHttpRequest v JavaScriptu (převzato z [2])

Na prvních dvanácti řádcích se provádí kroky nutné k vytvoření objektu XMLHttpRequest. Na řádku číslo 3 se pokoušíme o přímé vytvoření, to se podaří u všech moderních prohlížečů kromě Internet Exploreru 5 a 6 (dále IE5 a IE6). V nich a starších prohlížečích dojde k výjimce, kterou odchytáváme na řádku číslo 5. Následně se pokoušíme vytvořit XMLHttpRequest prostřednictvím objektu ActiveX (pro IE5 a IE6). Pokud i zde dojde k výjimce, znamená to, že daný prohlížeč vůbec XMLHttpRequest nepodporuje.

Tím jsme vytvořili requester. Na řádku 13 do něj vkládáme požadavek pro server. Prvním argumentem je metoda odesílaného požadavku. My jsme zvolili metodu GET, u níž se parametry vkládají přímo do url (2. parametr). Třetí parametr říká, zda se má požadavek odeslat asynchronně (téměř vždy bude mít hodnotu `true`). Metoda open

může mít ještě další dva parametry, jedná se o uživatelské jméno a heslo. Těch se využívá, pokud je dokument na serveru zabezpečený.

Když máme vytvořen požadavek, zaregistrujeme funkci (její definice začíná na řádku 17), která se má v případě změny jeho stavu spustit (řádek 14) a odešleme ho (řádek 15). Požadavek může nabývat 5 stavů (0 – Neinicializováno; 1 – Inicializováno, požadavek ale neodeslán; 2 – Požadavek odeslán; 3 – Probíhá příjem odpovědi; 4 – Odpověď přijata) podle [3]. V zaregistrované funkci kontrolujeme stav požadavku. Když nabyl stavu číslo 4, znamená to, že požadavek byl zpracován a máme k dispozici odpověď. Dále zkontrolujeme stav odpovědi (řádek 19). Hodnoty 200 a 304 říkají, že požadavek byl úspěšně vyřízen. Hodnota 200 znamená úspěch a 304 úspěch s dodatečnou informací, že se nic od posledního požadavku nezměnilo (vrácená stejná hodnota). V těchto případech voláme funkci pro zpracování odpovědi (řádek 20), jejíž definice začíná na 28. řádku (v tomto příkladě se pouze vypíše textová podoba odpovědi). V případě neúspěšné odpovědi (řádek 23) se vykoná nějaký obslužný kód (my zde pouze vypisujeme hlášení). Také by na tomto místě mohlo být ještě další větvení na základě chybových kódů, například pro případ nenalezení stránky (kód 404). Pokud se chcete dozvědět o objektu XMLHttpRequest více, navštivte stránky konsorcia W3C[34].

Nyní si určitě říkáte, že práce spojená s asynchronní komunikací je zbytečně složitá a zdlouhavá. Pokud se používá pouze samotný JavaScript, tak tomu tak opravdu je. Naštěstí existují zdařilé frameworky, které ji úspěšně zjednodušují. Jak jsem se zde již dříve zmiňoval, já jsem pro naši kommponentu mezi všemi vybral populární framework JQuery. Podívejme se tedy na přepis výše uvedeného kódu.

```
1   $.ajax({
2       type: 'GET',
3       url: 'getPollResult.php',
4       data: { id: 1987423 },
5       dataType: 'text',
6       success: function (response) {
7           alert(response);
8       }
9   });
```

Kód 9: Přepis Kódu 8 pomocí frameworku JQuery

34 W3C. *XMLHttpRequest* [online]. [cit. 2010-12-27]. URL: <http://www.w3.org/TR/XMLHttpRequest/>.

Zdá-li se Vám to stále složité, existuje ještě zkrácená verze.

```
1   $.get(
2       'getPollResult.php',
3       { id: 1987423 },
4       function (response) {
5           alert(response);
6       }
7   );
```

Kód 10: Zkrácený zápis Kódu 9

Zde je jasně vidět, jak moc se vyplácí používat JavaScriptové frameworky. Všechny metody JQuery pro práci s AJAX jsou přehledně popsány v dokumentaci na oficiálních stránkách[35].

1.2.4 PHP

Shrňme si, co už vše umíme. Naučili jsme se psát klientské skripty využívající Google API. Z klientských skriptů umíme zasílat požadavky serveru. Další věcí, kterou se ještě musíme naučit je psaní serverových skriptů. U jazyku pro serverové skripty jsem dlouho váhal. Velice mě lákalo prostředí ASP.NET s výhodami využití .NET Frameworku, dokonce jsem v něm i začal tuto komponentu vyvíjet. Setkal jsem se ale záhy s řadou problémů. Nejzásadnějšími byly obtíže s hledáním potřebných materiálů a ukázkových kódů. V zásadě většina materiálů se zabývala vytvářením celých stránek a ne komponent. Z těchto důvodů jsem nakonec přešel k možnosti využít PHP. U nás se stále těší veliké oblibě, má zde delší tradici, a tak vychází mnoho užitečných článků i v češtině.

O PHP toho již bylo napsáno opravdu mnoho. Většina lidí pohybujících se v oboru výpočetní techniky se s ním nejspíš už někdy setkala. Tak si tedy pouze pro úplnost povíme základní vlastnosti a předvedeme si malou ukázku kódu.

PHP je skriptovací jazyk s "céčkovskou" syntaxí (není v jistých ohledech ani příliš vzdálený JavaScriptu). Proměnné jsou dynamicky typované. Od páté verze je podpora pro objektově orientované programování srovnatelná například s Javou. To můžete sami posoudit v následujícím výpisu. Použil jsem stejný příklad jako v kódu 1 pro vytváření objektů v JavaScriptu.

35 The jQuery Project. *Ajax - jQuery API* [online]. [cit. 2010-12-27]. URL: <http://api.jquery.com/category/ajax/>.

```php
<?php
 class auto{
   private $text;
   public $nazev;
   function __construct($nazev, $text){     //konstruktor
       $this->text = $text;
       $this->nazev = $nazev;
   }
   private function getText(){
       return $this->text;
   }
   public function nastartovat(){
       echo $this->nazev.$this->getText();
   }
 }

 $bmw = new auto("BMW X5", " nastartovalo!");
 $fiat = new auto("Fiat Punto", " nenastartovalo!");
 $bmw->nastartovat();
 echo $fiat->nazev;              //příkaz echo slouží pro tisk
řetězců
?>
```

Kód 11: Ukázka objektu v PHP

Jedná se o klasický objektový přístup, jen si musíte dávat pozor, že se pro přístup k metodám nepoužívá obvyklá tečková konvence, nýbrž šipková. Dále pro konstruktor existuje klíčové slovo __construct (i když funguje možnost definování konstruktoru prostřednictvím jména třídy, kvůli polymorfismu se to nedoporučuje). Poslední věcí, na kterou bych chtěl upozornit, je nutnost psaní znaku dolaru před každou proměnnou (často se na něj zapomíná). Pokud jste ale někdy programovali ve kterémkoli "céčkovském" programovacím jazyce, neměl by vám přechod do PHP činit větší obtíže. Pro bližší seznámení s PHP doporučuji jakoukoli příručku pro verzi 5 a vyšší[36].

Tím jsme se naučili už vše potřebné k implementaci a nic nám již nebrání v přistoupení k praktické části diplomové práce.

36 např.:
BRÁZA, Jiří. *PHP 5 : Začínáme programovat* [online]. [s.l.] : [s.n.], 2005 [cit. 2010-12-27]. URL: <http://books.google.cz/books?id=9RHzUTsTxqUC&printsec=frontcover&dq=php+5&hl=cs&ei=DfTmTL2mMI2fOsH1wMEK&sa=X&oi=book_result&ct=result&resnum=1&sqi=2&ved=0CCwQ6AEwAA#v=onepage&q&f=false>.

2 Implementace

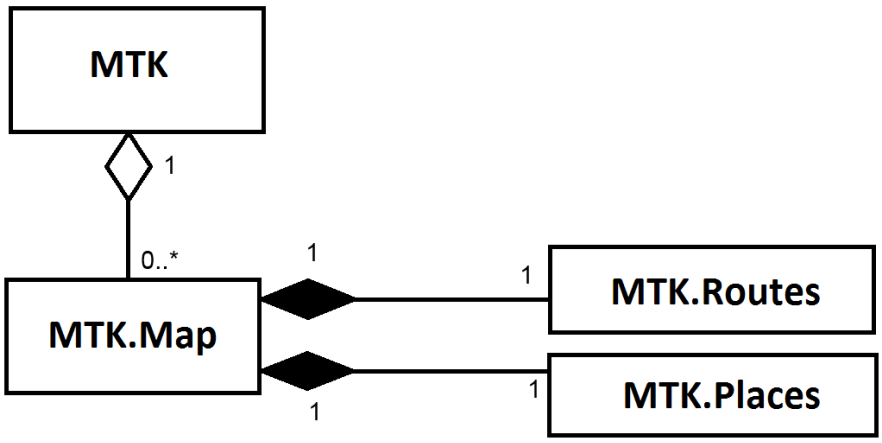
Když už jsme si probrali celý návrh a naučili se pracovat s potřebnou technologií, tak nám nic nebrání v přistoupení k samotné implementaci. V této kapitole se podíváme, jak jsem realizoval návrh popsaný na začátku předchozí kapitoly. Probereme si podrobněji jednotlivé vrstvy. Ukážeme si jejich objektový návrh a vysvětlíme si některé důležité části kódu. Také se zmíním o dalších možnostech vylepšení a o některých slepých uličkách, kterými jsem se při práci vydal. Po přečtení této kapitoly byste měli získat celkem dobrou představu o tom, co vše vývoj takovéto webové komponenty obnáší, případně na co si máte dát pozor a čeho se vyvarovat.

Jak jsme si již řekli v minulé kapitole, komponenta se skládá ze čtyř vrstev. Postupně si popíšeme všechny vrstvy, začneme seshora, stejně jako jsem postupoval při realizaci.

2.1 Prezentační vrstva s částí vrstvy aplikační logiky

Jedná se o klientskou část komponenty, která se vykonává na straně klienta v jeho webovém prohlížeči. Jednotlivé akce se provádějí prostřednictvím klientských skriptů. Pro větší přehlednost programového kódu a pro jeho lepší znovupoužitelnost je využit objektový přístup. Jednotlivé logické bloky kódu jsou rozděleny do tříd.

Vrstva se skláda z jedné zaváděcí funkce a ze čtyř tříd, jejich hierarchii znázorňuje následující obrázek.



Obr. 2:Hierarchie tříd tvořících prezentační vrstvu (zdroj: autor)

- Třída `MTK` je něco jako statická třída, obsahuje různé pomocné statické metody. Mimojiné metody pro transformaci získaných dat od nižších vrstev do HTML elementů, aby se poté mohla zobrazit pomocí webového prohlížeče. Dále obsahuje pole instancí třídy `Map`.
- Třída `Map` se stará o vytvoření mapy v prohlížeči podle atributů, které jsou nastaveny v elementu **mtk:map**. Dále se zde definují reakce na události mapy (jako např. kliknutí) a obsahuje instanci tříd `Routes` a `Places`.
- Třída `Routes` slouží k vytváření, úpravě a zobrazování tras. Nastavení se provádí prostřednictvím elementu **mtk:route**.
- Třída `Places` slouží k vytváření, úpravě a zobrazování míst. Nastavení se provádí prostřednictvím elementu **mtk:place**.
- Zaváděcí funkce `loadScript` se spustí ihned po kompletním načtení stránky. Jejím úkolem je přilinkování potřebných knihoven (knihovna JQuery a Google Maps API). Po načtení knihovny Google Maps API se volá statická funkce `Start` třídy `MTK`. Zdrojový kód vypadá následně:

```
1   window.onload = loadScript;   //spuštění zaváděcí funkce po
  načtení stránky
2
3   function loadScript() {
4       var script = document.createElement("script");
5       script.type = "text/javascript";
6       script.src = "http://code.jquery.com/jquery-1.4.2.js";
7       document.body.appendChild(script);
8
9       script = document.createElement("script");
10      script.type = "text/javascript";
11      script.src = "http://maps.google.com/maps/api/js?
  sensor=false&callback=MTK.Start";
12      document.body.appendChild(script);
13  };
```

Kód 12: Zaváděcí funkce

Na prvním řádku vidíme registraci funkce na událost "kompletní načtení stránky". Od třetího řádku začíná definice zaváděcí funkce. Pro přilinkování knihoven se využívá DOM[37], pomocí něhož vkládáme do HTML-dokumentu elementy typu

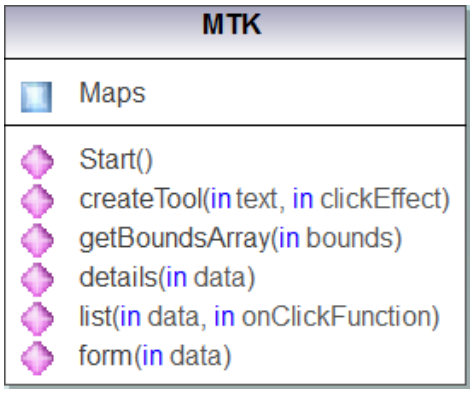
37 Document Object Model

script sloužící pro načtení skriptů. Na řádku 11 si všimněme parametru `callback`. Ten určuje funkci, která se má vykonat po úplném načtení Google API. V tomto případě volíme statickou metodu `Start` třídy `MTK`.

V následující části se podíváme na jednotlivé třídy trochu víc zblízka.

2.1.1 MTK

V řeči objektově orientovaného programování bychom mohli mluvit o třídě MTK jako o statické třídě či singletonu (i když se ve skutečnosti jedná pouze o instanci nepojmenovaného objektu, protože JavaScript nic jiného nepodporuje). Jaké metody obsahuje, vidíme na následujícím diagramu.



Obr. 3: Třída MTK (zdroj: autor; vytvořeno pomocí Altova Umodel)

Jak vidíme, obsahuje metody pro zobrazování dat. Funkce `details` převádí data na výpis dvojic klíč a hodnota. Funkce `form` má podobný účel akorát s tím rozdílem, že hodnota se zobrazí v textovém editovatelném poli. Funkce `list` datovou hodnotu reprezentuje v podobě odkazu, po kliknutí na něj se provede metoda předaná jako druhý parametr. Metoda `createTool` má podobný význam, pouze prvním parametrem není skupina dat ale jen řetězec. Metoda `getBoundsArray` převádí hodnotu `bounds` z Google API na čtveřici hodnot (minimální a maximální hodnoty zeměpisné délky a šířky). Kromě těchto metod ještě obsahuje třída pole Maps pro ukládání instancí Třídy `Map` (na jedné stránce tak může být vloženo více map). Toto pole se plní po zavolání metody `Start`, která se automaticky spouští po načtení Google Maps API. Podivejme se na jeji předpis.

```
1   MTK = {
2     Maps: new Array(),
3
4     Start: function () {
5       var maps = document.getElementsByTagName("mtk:map");
6
7       for (var i =  maps.length - 1; i>= 0; i--)
8         if (maps[i].hasAttribute("id"))
9           MTK.Maps[maps[i].getAttribute("id")] = new
  MTK.Map(maps[i]);
10    },
11    ...
12  };
```

Kód 13: Třída MTK a její metoda Start

Po spuštění metody se vyhledají v HTML-dokumentu všechny výskyty elementu **mtk:map** (řádek 5) a následně se pro každý vytvoří objekt MTK.Map, který si uloží do asociativního pole ve vlastnosti Maps (řádek 9). Klíčem v tomto poli jsou identifikátory udané v elementu **mtk:map**. Tím jsme probrali celou třídu MTK.

2.1.2 MTK.Map

Pro vytvoření instance této třídy se volá konstruktor s jedním parametrem. Tím je element **mtk:map** dokumentu (připomínám, že konstruktor se volá v metodě `Start` třídy `MTK`, viz Kód 13: řádek 9). V konstruktoru se vytváří HTML struktura mapy podle nastavení v předaném elementu. Tato struktura se skládá z canvasu a menu (pokud není nastaveno jinak). Canvas je element, ve kterém se zobrazuje samotná mapa, v menu zase možnosti pro práci s nimi. Dále se v konstruktoru vytváří instance tříd `Routes` a `Places`, odkazy na ně se ukládají do vlastností Routes a Places třídy Map pro budoucí použití. Dalšími v konstruktoru inicializovanými vlastnostmi jsou Canvas (odkaz na element s canvasem mapy) a Menu (odkaz na element s menu mapy). Jako poslední se definují reakce na události vyvolané v mapě (kliknutí do mapy, zoomování nebo posunutí mapy). Před ukončením konstruktoru se v dokumentu původní element s nastavením (element **mtk:map**) nahradí nově vytvořenou HTML strukturou. Třída MTK.Map již žádné jiné metody kromě konstruktoru neobsahuje.

Mapa se může nacházet v jednom ze tři režimů. Prvním je prohlížecí, v tomto módu se zobrazují v menu trasy a místa, která se nacházejí ve zobrazeném výseku mapy. V budoucnu by zde měly být i zobrazeny příslušné body v mapě, ale bohužel v době odevzdání práce ještě nebyl dokončen potřebný klastrovací nástroj. Klastrovací

nástroj slouží pro zobrazování velkého množství bodů pomocí zástupných klastrů. Bez tohoto nástroje by byla naše komponenta příliš pomalá. Dále se v menu v tomto módu nachází odkaz pro vytvoření nových tras a míst a také vyhledávací nástroj.

Kliknutím na bod nebo trasu ve výpisu v menu se mapa přepne do módu prohlížení detailů. V menu se objeví výpis všech informací uložených v databázi a v mapě se zobrazí daný objekt. Z tohoto módu můžeme buď přejít zpět do prohlížecího módu, nebo do editačního.

V editační módu můžeme upravovat nebo mazat záznamy, přemisťovat objekty atd. Po uložení změn se opět přechází zpět do prohlížecího módu.

Pomalu se dostáváme k nejpracnější části. Třídu Route jsem nesčetněkrát předělával, ale výsledkem je vydařený modul pro práci s trasami, který může, co se vlastností týče, konkurovat produktu MapToolkit.

2.1.3 MTK.Route

Tato třída slouží k vytváření, úpravě, mazání a zobrazování tras. Trasy můžeme vytvářet pomocí dvou nástrojů. Na výběr máme buď úsečkový nástroj, který mezi zadanými body vede úsečku (nebo ortodromou, záleží na nastavení). Nebo můžeme použít magnetický nástroj. Ten tyto body propojí křivkou vedoucí po silnicích a cestách.

Na magnetickém nástroji jsem pracoval velice dlouho, než se dostal do nynější podoby. Hlavním stavebním kamenem je služba DirectionsService v Google API, které zašleme požadavek (DirectionsRequest), tato služba nám vrátí odpověď (DirectionsResult). Základní možnosti nastavení požadavku shrnuje následující tabulka.

Parametry	Datový typ	Popis
`avoidHighways`	`boolean`	Pokud má hodnotu true, služba vyhledá trasy mimo dálnice. Implicitně má hodnotu false. Parametr není povinný.
`avoidTolls`	`boolean`	Pokud má hodnotu true, služba vyhledá cesty bez placených úseků. Implicitně false. Je nepovinný.
`destination`	`LatLng|string`	Pozice cílového místa. Definováno buď geografickou pozicí nebo řetězcem obsahujícím adresu. Parametr je povinný.

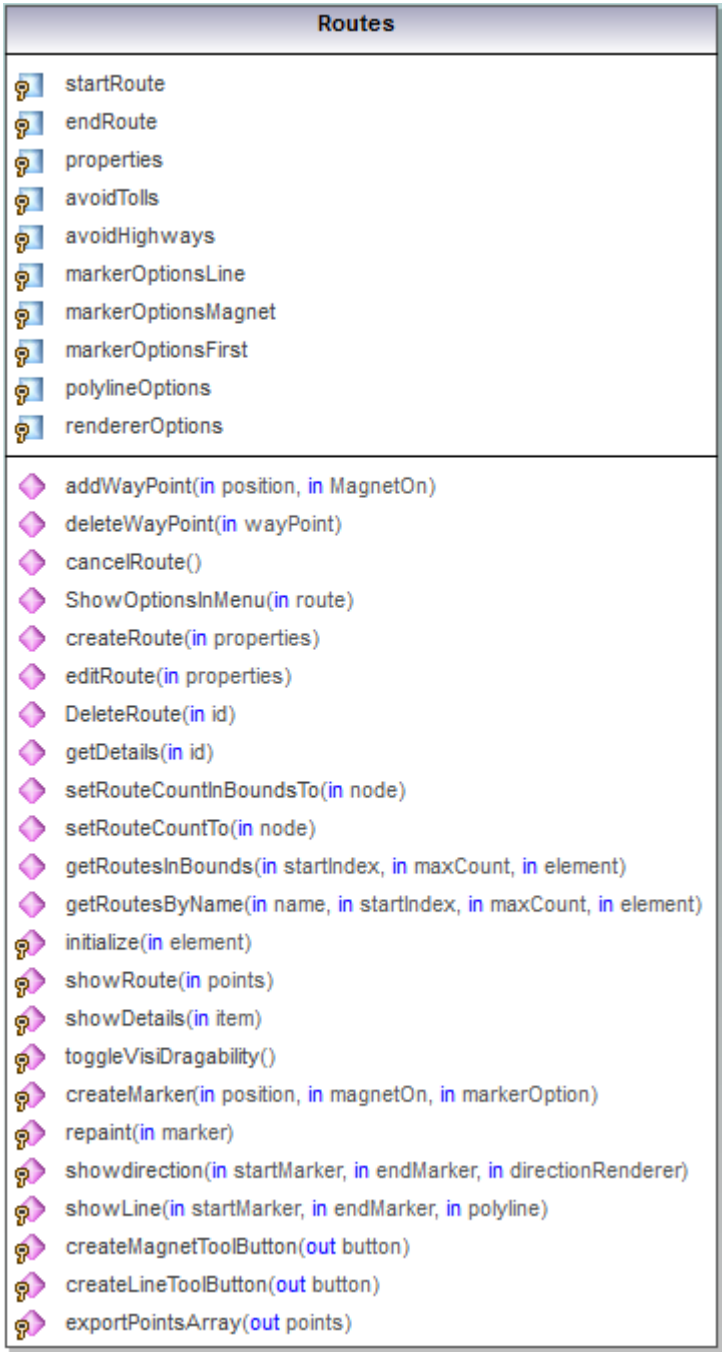
optimizeWaypoints	boolean	Když je nastaveno true, DirectionService uspořádá pořadí průjezdních míst tak, aby cesta jimi procházející byla co nejkratší.
origin	LatLng|string	Pozice počátečního místa. Definováno buď geografickou pozicí nebo řetězcem obsahujícím adresu. Parametr je povinný.
travelMode	DirectionsTravelMode	Typ požadovaného cestovního módu (hodnoty BICYCLING, DRIVING nebo WALKING). Povinný parametr.
waypoints	Array.<DirectionsWaypoint>	Pole průjezdních míst. Trasa se hledá z počátečního místa do cílového přes všechna průjezdní místa v tomto poli (maximálně 8 hodnot). Nepovinný parametr.

Tab 1: Přehled základních vlastností objektu DirectionsRequest (převzato z [4])

Z důvodu omezení počtu stanovišť na trase (waypoints) hledáme cestu po částech (vždy pouze mezi dvěmi sousedícími body), čímž se zvyšuje počet odeslaných požadavků. Původně jsem výsledky ukládal jako pole Geo-bodů (LatLng), ty jsem následně zobrazoval jako křivky (objekt PolyLine z Google API) a ukládal do databáze bod po bodu. To ale příliš zatěžovalo server, protože se cesta skládala z velkého počtu bodů a neše datová vrstva nebyla zrovna moc optimalizovaná pro velké datové toky (neprovádíme cachování apod.). Další nevýhodou bylo, že když se vytvářel dlouhý úsek, po přiblížení již cesta nevedla přímo po cestě, protože se úsek skládal průměrně pouze z 200 bodů. Také bylo nutné doprogramovávat zobrazení copyrightu, které plyne z podmínek použití služby DirectionsService. Všechny tyto neduhy mohl vyřešit vykreslovač (objekt DirectionsRenderer z Google API), ten při každé změně měřítka mapy automaticky přepočítává cestu, takže je hladká i při maximálním přiblížení. Aby vše nebylo tak jednoduché, vykreslovač má svá omezení. Zásadním je, že neumí vykreslit cestu složenou z více jak osmi úseků. Toto se ale dá obejít použitím vlastního vykreslovače pro každý úsek cesty. V tom případě ale musíme změnit datovou reprezentaci cesty (objekt DirectionsRoute z Google API nám stačit nebude) a doprogramovat editaci cesty, kterou už automaticky provádět nemůžeme (úseky by na sebe nenavazovaly).

Nejlepší reprezentací cesty je podle mého soudu obousměrný spojový seznam

stanovišť (waypoints), které jsem použil ve třídě MTK.Routes. Tím můžeme jednoduše manipulovat s úseky cesty (mazat, měnit průběh atd.). Podívejme se tedy na ni blíže.



Obr. 4: Diagram třídy Routes (zdroj: autor; vytvořeno pomocí Altova Umodel)

Třída Routes dovoluje v jednom okamžiku zobrazovat pouze jednu cestu, tuto cestu reprezentujeme obousměrným spojovým seznamem. Prvky seznamu jsou odvozené od objektu Marker z Google API, který slouží pro vykreslování bodů. Ten

jsem rozšířil o odkaz na předchozí a následující prvek a o nastavení úseku. Toto rozšíření ukládám do nové vlastnosti `_state` objektu Marker. Pro uložení používám pomocnou třídu InnerStateMarker.

```
1   function InnerStateMarker(_next, _prev, _legDisp, _magOn) {
2     this.next = _next;
3     this.prev = _prev;
4     this.legDisplay = _legDisp;
5     this.magnetOn = _magOn;
6   };
```

Kód 14: Pomocná třída InnerStateMarker

Vlastnosti `prev` a `next` osahují zmíněné odkazy na předchozí a následující body. `LegDisplay` je odkazem na vykreslovač úseku mezi tímto a předchozím bodem, vykreslovačem může být buď objekt DirectionsRenderer (magnetický úsek), nebo Polyline (úsečkový úsek). Jestli je úsek magnetický nebo úsečkový poznáme podle nastavení poslední vlastnosti `magnetOn`.

Odkaz na začátek spojového seznamu se nachází ve vlastnosti `startRoute` a odkaz na konec v `endRoute`. Pro zobrazení trasy, tvorbu, úpravu a mazání spojového seznamu jsou ve třídě Routes speciální metody.

- `addWayPoint`: slouží k vytvoření nového úseku trasy. Volá se po kliknutí do mapy v módu editace nebo vytváření trasy. První parametr značí geografické souřadnice místa kliknutí, druhý parametr určuje, zda se má pro vytvoření úseku použít magnetický či úsečkový nástroj. Podívejme se na ni trochu více zblízka.

```
1   this.addWayPoint = function (latLng, magOn) {
2     if (endRoute == null) {
3       startRoute = createMarker(latLng, magOn, markerOptionsFirst);
4       endRoute = startRoute;
5       return;
6     }
7
8     var prevEnd = endRoute;
9     if (magOn)
10      endRoute = createMarker(latLng, magOn, markerOptionsMagnet);
11    else
12      endRoute = createMarker(latLng, magOn, markerOptionsLine);
13    prevEnd._state.next = endRoute;
14    endRoute._state.prev = prevEnd;
15    if (magOn)
16      showDirection(prevEnd, endRoute, endRoute._state.legDisplay);
```

```
17    else
18      showLine(prevEnd, endRoute, endRoute._state.legDisplay);
19  };
```

Kód 15: veřejná metoda addwayPoint

Tato metoda má za úkol přidat průjezdní bod a vykreslit nový úsek v mapě. Pokud trasa dosud neobsahuje žádný bod (obousměrný seznam je prázdný), vytvoříme první bod (řádky 2 až 6). V opačném případě přidáme nový bod na konec (řádky 9 až 12), propojíme příslušné prvky seznamu (řádky 13 a 14) a zobrazíme úsek v mapě (řádky 15 až 18). Jak můžeme vidět, metoda využívá další funkce.

- `showLine:` zobrazí v mapě nový úsek v úsečkovém režimu
- `showDirection:` zobrazí v mapě nový úsek v magnetickém režimu
- `createMarker:` vytváří nový průjezdní bod, na tuto funkci se také ještě podíváme.

```
1   function createMarker(latLng, magOn, markOpt) {
2     var marker = new google.maps.Marker(markOpt);
3     marker.setAnimation(google.maps.Animation.DROP);
4     marker.setPosition(latLng);
5     marker.setMap(canvas);
6     if (magOn)
7       marker._state = new InnerStateMarker(null, null,
          new google.maps.DirectionsRenderer(rendererOptions), magOn);
8     else
9       marker._state = new InnerStateMarker(null, null,
          new google.maps.Polyline(polyLineOptions), magOn);
10
11    google.maps.event.addListener(marker, "dragend", function () {
12      repaint(this);
13    });
14
15    google.maps.event.addListener(marker, "dblclick", function () {
16      if (this == startRoute) return;
17      if (this._state.legDisplay != null) {
18        this._state.legDisplay.setMap(null);
19        this._state.legDisplay = null;
20      }
21      this._state.magnetOn = !this._state.magnetOn;
22      if (this._state.magnetOn) {
23        this._state.legDisplay =
            new google.maps.DirectionsRenderer(rendererOptions);
24        this.setOptions(markerOptionsMagnet);
25      }
26      else {
27        this._state.legDisplay = new
```

```
  google.maps.Polyline(polyLineOptions);
28        this.setOptions(markerOptionsLine);
29      }
30      repaint(this);
31    });
32
33    return marker;
34  };
```

Kód 16: Soukromá metoda createMarker

Tato funkce přijímá tři parametry. Prvním je pozice bodu, druhým informace, zda se jedná o magnetický či úsečkový úsek a posledním nastavení vlastností bodu. Nejprve se zobrazí bod na mapě (řádky 2 až 5), poté Marker rozšíříme o naší třídu `InnerStateMarker` (řádky 6 až 9). Nakonec se definují reakce na událost přesunu bodu (řádky 11 až 13) a na událost dvojklik na bodu (řádky 15 až 31). Po dvojkliku se změní nástroj pro vykreslení úseku z magnetického na úsečkový a naopak. Následně se cesta překreslí. Po přetažení bodu se překreslí také. O překreslování se stará funkce `repaint`.

- `repaint`: slouží k aktualizaci úseků obsahující bod předaný v parametru

Abychom probrali všechny zásadní funkce z prezentační vrstvy, které se nachází ve třídě Routes, nesmíme vynechat ještě následující.

- `deleteWayPoint`: slouží k vymazání libovolného bodu trasy. Pokud se maže nějaký bod zprostředku, spojí se dva úseky obsahující mazaný bod v jediný. Zda bude pro vykreslení úseku vzniklého spojením použit magnetický nebo úsečkový nástroj, určuje koncový bod druhého úseku.
- `cancelRoute`: skryje zobrazenou cestu v mapě a smaže obousměrný spojový seznam.
- toggleVisiDragability: přepíná možnost přesouvání průjezdních bodů a jejich viditelnost (kromě prvního bodu); využívá se toho při přepínání komponenty mezi editačním a prohlížecím režimem

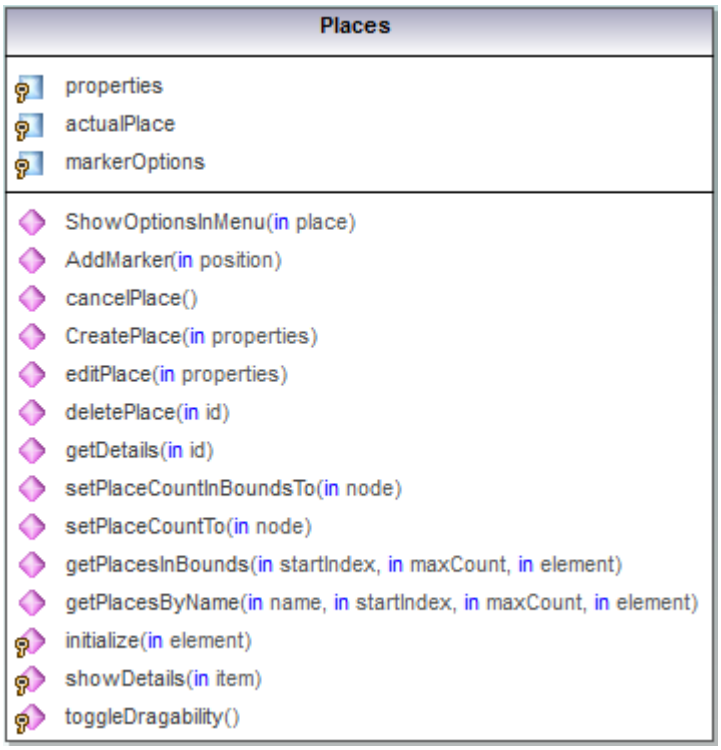
Další velkou skupinou metod třídy Routes jsou metody aplikační logiky. Tyto metody využívají technologii AJAX pro komunikaci se serverem. Zasílají mu asynchronní požadavky pro vytváření, mazání, prohlížení a editaci dat uložených v datovém úložišti.

- `createRoute`: zasílá požadavek pro uložení trasy z našeho spojového seznamu jako nové trasy v datovém úložišti
- `editRoute`: zasílá požadavek pro úpravu existující trasy v datovém úložišti podle trasy ze spojového seznamu
- `deleteRoute`: zasílá požadavek pro smazání trasy z datového úložiště
- `getDetails`: zasílá požadavek o získání všech dat o trase s daným id z datového úložiště
- `setRouteCountTo`: zasílá požadavek pro získání počtu všech tras v datovém úložišti, tento údaj poté zobrazí v elementu, který jsme ji předali jako parametr
- `setRouteCountInBoundsTo`: zasílá požadavek pro získání počtu všech tras nacházejících se v daném výřezu mapy, tento údaj poté zobrazí v elementu, který jsme předali jako parametr
- `getRoutesInBounds`: zasílá požadavek pro získání všech tras z daného výřezu mapy
- `getRoutesByName`: zasílá požadavek pro získání všech tras obsahujících daný řetzec ve jménu

Tím jsme probrali celou třídu Routes. Nyní se podívejme na třídu Place.

2.1.4 MTK.Place

Tato třída slouží ke zobrazování, ukládání, mazání a editaci míst. Třída Place toho má opravdu mnoho společného s tou předchozí. Dalo by se říci, že třída Routes je její složitější sestra. Proto se jí nebudeme zaobírat příliš do podrobna. Pouze si ukážeme její diagram a povíme si k čemu jednotlivé metody slouží.



Obr. 5: Diagram třídy Places (zdroj: autor; vytvořeno pomocí Altova Umodel)

Co se týče vlastností, je jich zde jen pár. Ve vlastnosti `properties` se uchovává seznam položek, které se u místa budou ukládat, tento seznam se tvoří v inicializační funkci podle údajů ze souboru *"PlaceProperties.txt"*. Vlastnost `markerOptions` obsahuje nastavení ikony zobrazované v mapě. Vlastnost actualPlace v sobě skrývá objekt Marker z Google API, který reprezentuje místo, se kterým se právě pracuje.

Stejně jako v případě třídy Routes se metody dělí do dvou skupin. První je skupina metod prezentační vrstvy. Do ní patří:

- `ShowOptionsInMenu`:
- `AddMarker`:
- `cancelPlace`:

Druhá je skupina metod aplikační logiky, které zasílají požadavky na server.

- `createPlace`: zasílá požadavek pro uložení aktuálního místa v datovém úložišti

- `editPlace`: zasílá požadavek pro úpravu existujícího místa v datovém úložišti podle aktuálního místa
- `deletePlace`: zasílá požadavek pro smazání místa z datového úložiště
- `getDetails`: zasílá požadavek o získání všech dat o místu s daným id z datového úložiště
- `setPlaceCountTo`: zasílá požadavek pro získání počtu všech míst v datovém úložišti, tento údaj poté zobrazí v elementu, který jsme předali jako parametr
- `setPlaceCountInBoundsTo`: zasílá požadavek pro získání počtu všech míst nacházejících se v daném výřezu mapy, tento údaj poté zobrazí v elementu, který jsme ji předali jako parametr
- `getPlacesInBounds`: zasílá požadavek pro získání všech míst z daného výřezu mapy
- `getPlacesByName`: zasílá požadavek pro získání všech míst obsahujících daný řetzec ve jménu

2.2 Vrstva aplikační logiky (serverová část)

Když se odešle nějaký požadavek od klienta, zpracuje se na serveru. Kód pro jeho zpracování se nachází v souboru *"AppLogic.php"*.

```
1   require ("DBpripojeni.php");
2   require ("MySQL_DAL.php");
3
4   $DAL_Route = new MySqlRouteProvider(SQL_HOST,SQL_USERNAME,
  SQL_PASSWORD, SQL_DATABASE);
5   $DAL_Place = new MySqlPlaceProvider(SQL_HOST,SQL_USERNAME,
  SQL_PASSWORD, SQL_DATABASE);
6
7
8   function fail($message) {
9       return array("status" => "fail", "message" => $message);
10  }
11
12  function success($data) {
13      return array("status" => "success", "data" => $data);
14  }
15
16  function returnResult($result){
17      die (json_encode($result));
18  }
19  switch ($_POST["action"]){
20  case "CreatePlace":
21      $res = $DAL_Place->InsertPlace($_POST["details"]);
22
23      returnResult($res);
24      break;
25
26  case "EditPlace":
27      $res = $DAL_Place->UpdatePlace($_POST["details"]);
28
29      returnResult($res);
30      break;
31  ...
32  case "InsertRoute":
33      $res = $DAL_Route->InsertRoute($_POST["details"],
  $_POST["points"]);
34
35      returnResult($res);
36      break;
37
38
39  default:
40      returnResult (fail("Spatny parametr action!!!"));
41  }
```

Kód 17: Kód pro zpracování požeadavků od klienta

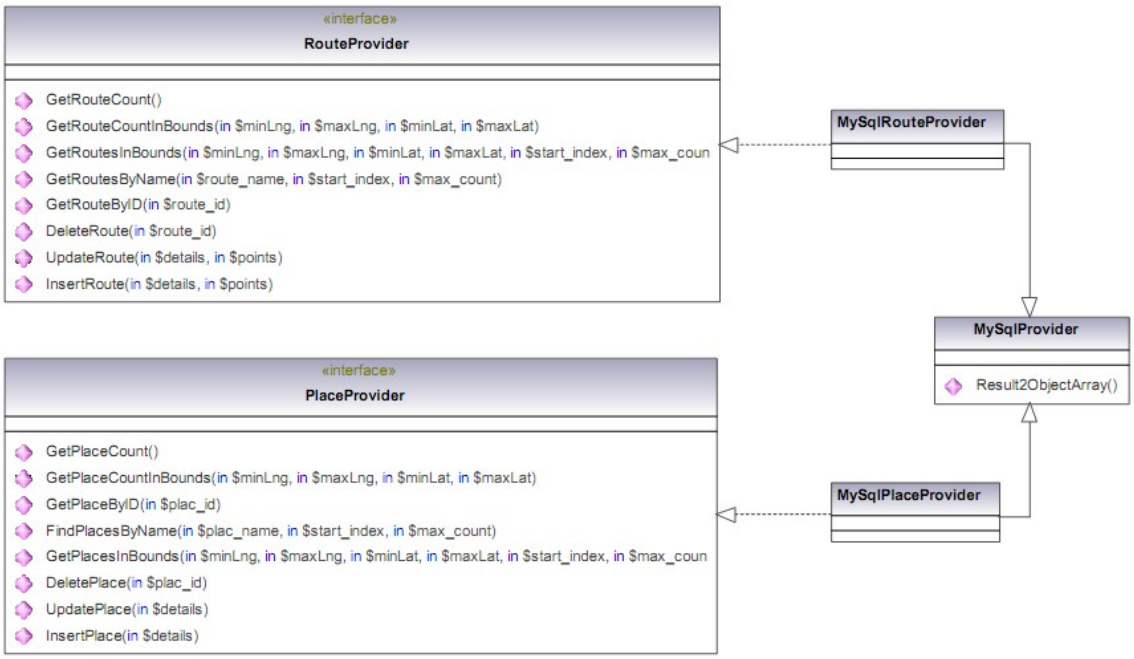
Na začátku souboru se přilinkovávají soubory. První z nich obsahuje údaje potřebné pro přípojení k databázi a ten druhý obsahuje kód vrstvy přístupu k datům. Na 3. a 4. řádku vytváříme objekty z vrstvy přístupu k datům, inicializují se hodnotami nutnými pro připojení k databázi. Dále se definují tři pomocné funkce pro vytváření a posílání odpovědí na požadavek klienta, odpovědi se skládají ze dvou částí: stavu odpovědi (success nebo fail) a části obsahující data nebo zprávu. Když vše proběhne vpořádku, vygeneruje se úspěšná odpověď pomocí funkce `success`, do stavu se vloží hodnota "success" a do datové části vrácená hodnota. V případě výskytu nějakého problému se vygeneruje odpověď pomocí funkce `fail`, do stavové části se vloží hodnota "fail" a do datové části zpráva o vzniklém problému. Pro poslání odpovědi klientovi se musí ukončit skript, k tomuto účelu slouží funkce `returnResult`, ta navracené hodnoty zakóduje do formátu JSON a ukončí skript.

Požadavky se zpracovávají na základě parametru požadavku ***action***. Podle jeho hodnoty se ve větvení nalezne příhodná větev a zavolá se příslušná metoda vrstvy přístupu k datům. Pokud parametru neodpovídá žádná z větví, vrátí se chybová odpověď se zprávou.

2.3 Vrstva přístupu k datům

Mým původním úmyslem bylo pro ukládání dat používat standardní aplikační rozhraní pro přístup k datům (ODBC: Open DataBase Connectivity), čímž by se naše komponenta mohla použít se kteroukoli databází podporující ODBC (což je v dnešní době většina). Od tohoto záměru jsem ovšem upustil kvůli nedostatečné praxi s tímto formátem. Z důvodu úspory programátorského času jsem se přiklonil k ukládání dat používat databázový systém MySQL, s nímž už jsem nějakou praxi měl. Ovšem využití standardu ODBC jsem nezavrhl, ale pouze odsunul do budoucna.

Vrstva přístupu k datům se musí skládat ze dvou tříd implementujících předepsané rozhraní. Jedna třída iplementující rozhraní PlaceProvider a jedna RouteProvider. Tyto třídy se starají o ukládání, úpravu a získávání dat z datového úložiště. Protože jsem se rozhodl ukládat data do databázového systému MySQL, vytvořil jse třídy MySqlRouteProvider a MySqlPlaceProvider. Ty jsem odvodil od společné třídy MySqlProvider, aby nevznikala duplicita kódu. V této rodičovské třídě se nachází společný kód obou tříd.



Obr. 6: Objektový model vrstvy přístupu k datům (zdroj: autor; vytvořeno pomocí Altova Umodel)

Pokud byste chtěli používat naší komponentu s jiným datovým úložištěm, museli byste si nadefinovat vlastní třídy odvezené od výše zmíněných rozhraní. Dále byste museli ještě v souboru "AppLogic.php" přepsat názvy použitých tříd a přilinkovat soubor s novými třídami. V nových třídách byste museli implementovat předepsané rozhraní. Implementace metody GetPlaceCount třídy MySqlPlaceProvider vypadá takto.

```
1   function GetPlaceCount(){
2     @$result = mysql_query("SELECT count(*) FROM mtk_places",
                                                    $this->conn);
3     if ($result == false){
4       return fail("Chyba v prikazu!!!");
5     }
6
7     $count = mysql_result($result,0);
8       if ($count == false)
9         return success(array("count" => 0));
10      return success(array("count" => $count));
11  }
```

Kód 18: Implementace metody GetPlaceCount třídy MySqlPlaceProvider

V této funkci se zadá dotaz pro datové úložiště a pokud vše proběhne správně vrátíme získanou hodnotu. Pro vracení výsledků používáme pomocné funkce `fail` a

`success`, se kterými jsme se již setkali v minulé kapitole. Tímto způsobem se definují všechny zbylé metody.

2.4 Datové úložiště

Jak jsem se zmínil v předešlé kapitole, naše komponenta podporuje zatím pouze databázový systém MySQL, což by se ale s implementací podpory ODBC ve vrstvě přístupu k datům mělo změnit.

Nejdůležitějším úkolem v této vrstvě byl návrh databázových tabulek. Mým požadavkem bylo umožnit přidávání položek ukládaných do databázových tabulek podle požadavků uživatele. Na výběr jsem měl buď přidání tabulky s uživatelskými položkami, nebo přidání uživatelských položek jako nových sloupců přímo do stávajících databázových tabulek. Rozhodl jsem se pro druhé řešení, protože nám tak zůstane větší kontrola nad přidávanými položkami a do budoucna nám umožňuje více možností pro další vylepšeni, jako jsou například určování datových typů a dalších vlastností nových položek (v současnosti všechny nové položky jsou typu `Varchar(45)` a nejsou povinné). Na druhou stranu se tím značně komplikuje přidávání nových položek v době, kdy už jsou v databázi uložena nějaká data.

Pro vytváření databázových tabulek jsem vytvořil PHP-skript, který vše provede automaticky. Musíme jen zadat údaje potřebné pro přihlášení do databázového systému a určit názvy pro nové položky. Skript se již o vše postará. Více si o tomto nástroji povíme při vytváření ukázkové webové aplikace.

3 Ukázková webová aplikace

Komponenta je navržena tak, aby ji mohl začlenit do své webové aplikace i programátorský laik. Jedinou nutnou znalostí jsou základy HTML. I toto by se ale mělo změnit, aby ji mohl využívat opravdu každý, v budoucnu plánuji vyvinout nástroj pro správu komponenty. V této chvíli se pro začlenění komponenty musí učinit jen pár následujících kroků.

- Zkopírovat složku *"MTK – Source"* do adresáře, ve kterém se nachází stránka, do níž komponentu chceme vložit. Tato složka obsahuje vše potřebné k úspěšnému zavedení komponenty.
- Nainstalovat databázový systém MySQL a vytvořit v něm dva typy uživatelů (uživatelský a administrátorský). Doporučuji k tomu použít nástroj MySQL Administrator[38], práce s ním je intuitivní a tudíž zde postup nebudu popisovat.
- Přihlašovací údaje vyplnit do souborů *"DBpripojeni.php"* a *"DBpripojeni_root.php"*. V těchto souborech se zadává adresa serveru s databázovým systémem (`SQL_HOST`), přihlašovací jméno (`SQL_USERNAME`), heslo (`SQL_PASSWORD`) a název databázového schématu, v němž se budou ukládat data (`SQL_DATABASE`).
- V souborech *"PlaceProperties.txt"* a *"RouteProperties.txt"* vyplnit na samostatné řádky názvy uživatelských polí (data, která se o místech a cestách budou ukládat do databáze).
- V prohlížeči spustit skript *createDatabase.php*[39], který se nachází také ve zdrojové složce komponenty. Tím se vytvoří databázové tabulky i s novými uživatelskými sloupci. Pokud již v databázovém systému existuje databázové schéma (určené dříve zmíněnou konstantou `SQL_DATABASE`), skript se zeptá, jestli chcete pokračovat. V tomto případě doporučuji nepokračovat, změnit hodnotu `SQL_DATABASE` a spustit skript znovu. V opačném případě totiž ztratíte všechna data v tomto schématu.

38 Ke stažení například na http://dev.mysql.com/downloads/gui-tools/5.0.html

39 Do adresního pole prohlížeče zadejte adresu ve tvaru "adresa/MTK%20-%20Source/createDatabase.php"

- Poslední věcí je vložení komponenty přímo na stránku a nastavení komponenty, to se provádí následujícím způsobem.

```
1   <html xmlns="http://www.w3.org/1999/xhtml" lang="cs">
2     <head>
3       <title>MapTulKid</title>
4       <script type="text/javascript" src="MTK -
  Source/MTK.js"></script>
5     </head>
6     <body>
7       ...
8       <mtk:map id="map1" center_lat="49.816721"
  center_lng="15.1710">
9         <mtk:route stroke_color="#000000" stroke_width="3" />
10        <mtk:place marker_icon="icon40s.png" />
11      </mtk:map>
12      ...
13    </body>
14  </html>
```

Kód 19: Vložení komponenty do stránky

V hlavičce stránky se vloží soubor se skripty (řádek 4) a na příslušném místě v těle stránky se vytvoří element **mtk:map** s nastavením komponenty (viz řádky 8 – 11). Nyní se pojďme podívat na možnosti nastavení komponenty.

3.1 Nastavení komponenty

U komponenty můžeme nastavovat vlastnosti mapy, tras a míst. Vše se provádí prostřednictvím atributů v příslušných elementech v dokumentu. V elementu **mtk:map** se mohou nastavovat následující atributy:

- **id**: je to povinný parametr, určuje jednoznačný identifikátor komponenty (pro případ použití více mapových komponent v jednom dokumentu), identifikátorem může být libovolný řetězec
- **style** - určuje podobu mapové komponenty, zapisují se do něj CSS styly, tento parametr je nepovinný
- **menu_node_id**: zapisuje se do něj id HTML-elementu, ve kterém se mají zobrazovat možnosti pro práci s komponentou, pokud není určen, vytvoří se automaticky vestavěné menu, parametr je nepovinný
- **place_tool_enable**: určuje, zda se má nabízet nástroj pro zobrazování,

vytváření a editaci míst (POIs), zadává se do něj pravdivostní hodnota, jeho výchozí hodnota je `true`, parametr je nepovinný

- **route_tool_enable**: určuje, zda se má nabízet nástroj pro zobrazování, vytváření a editaci cest, zadává se do něj pravdivostní hodnota, jeho výchozí hodnota je `true`, parametr je nepovinný
- **center_lat**: pomocí tohoto parametru se zadává výchozí zeměpisná šířka pro střed mapy, přípustné hodnoty jsou z intervalu <-90, 90>, výchozí hodnota je 0, parametr je nepovinný
- **center_lng**: pomocí tohoto parametru se zadává výchozí zeměpisná délka pro střed mapy, přípustné hodnoty jsou z intervalu <-180, 180>, výchozí hodnota je 0, parametr je nepovinný
- **zoom**: pomocí tohoto parametru se zadává výchozí zoom pro mapu, přípustné hodnoty jsou od 0 (nejméně podrobné) přibližně do 20 (v některých případech může být i o něco vyšší, záleží na podrobnosti map), výchozí hodnota je 1 (celý svět), parametr je nepovinný
- **max_data_count**: určuje počet položek zobrazených na jedné stránce výpisu (např. u výpisu tras), výchozí hodnota je 5, parametr je nepovinný

Co se týče tras, element **mtk:route** není povinný, pokud ho v elementu **mtk:map** definujeme, můžeme v něm nastavovat následující nepovinné atributy:

- **stroke_color**: pomocí tohoto parametru se nastavuje barva křivky pro vykreslování tras, barva se zadává v hexaformátu, začíná znakem '#' a následuje šest číslic z šestnáctkové soustavy, výchozí hodnota je '#0000FF' (modrá)
- **stroke_opacity**: tento parametr určuje průhlednost křivky pro vykreslování tras, zadává se reálné číslo (s desetinnou tečkou) z intervalu <0, 1> (0 – průhledná, 1 – plná), výchozí hodnota je 0.7
- **stroke_weight**: tento parametr udává tloušťku křivky pro vykreslování tras (zadává se v pixelech), výchozí hodnota jsou 3
- **geodesic**: parametr obsahuje pravdivostní hodnotu, pokud je nastaveno na

`true`, vytváří se úsek trasy v úsečkovém módu jako nejkratší cesta z výchozího do cílového místa (ortodroma – nejkratší spojnice na kulové ploše, ve většině případů se jedná o oblouk), v opačném případě se body spojí úsečkou, výchozí hodnota je `true`

- **avoid_highways**: parametr obsahuje pravdivostní hodnotu, určuje, zda v magnetickém módu má zakázat hledání tras vedoucích po dálnicích (hodnota `true`), výchozí hodnota je `false`
- **avoid_tolls**: parametr obsahuje pravdivostní hodnotu, určuje, zda v magnetickém módu má zakázat hledání tras vedoucích po placených úsecích (hodnota `true`), výchozí hodnota je `false`
- **start_icon**, **magnet_icon**, **line_icon**: obsahují řetězec (cestu) k obrázku pro zobrazení průjezdních stanovišť (počáteční stanoviště, stanoviště v úsečkovém a magnetickém módu), výchozí hodnoty jsou:

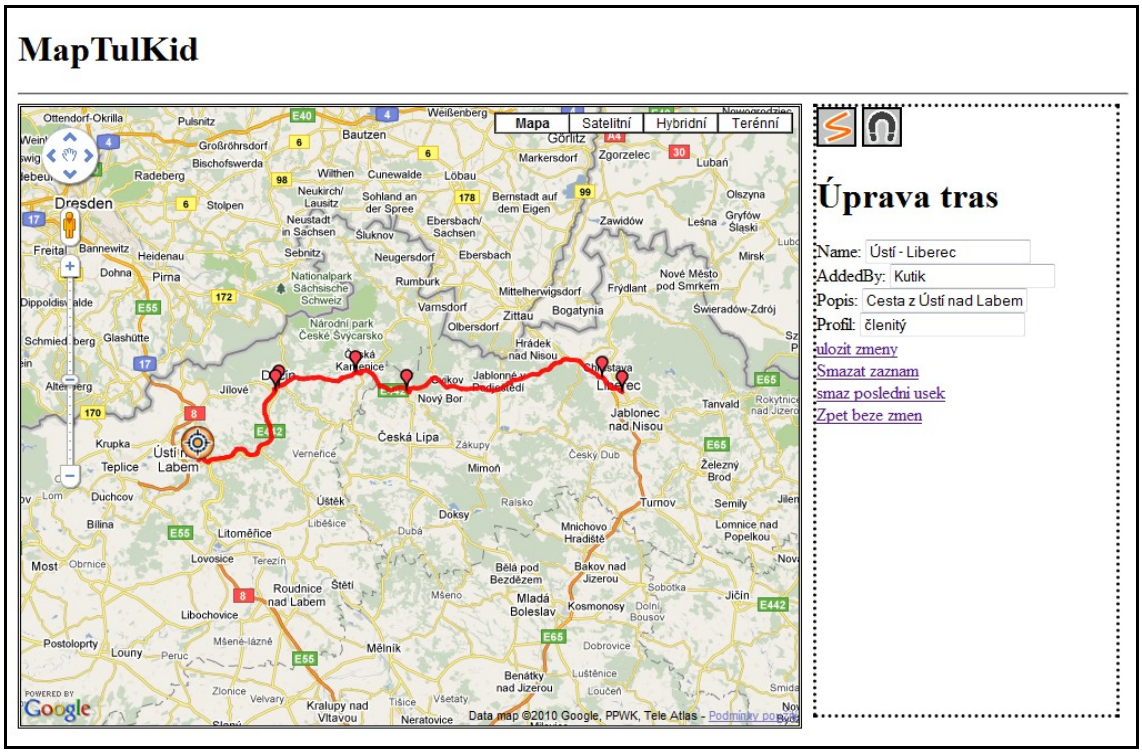
  "http://maps.google.com/mapfiles/kml/pal3/icon32.png" pro **start_icon**,
  "http://labs.google.com/ridefinder/images/mm_20_red.png" pro **magnet_icon** a
  "http://labs.google.com/ridefinder/images/mm_20_blue.png" pro **line_icon**

- **magnet_shadow**, **line_shadow**, **start_shadow**: obsahují řetězec (cestu) k obrázku stínu stanovišť, výchozí hodnotou je prázdný řetězec

Element **mtk:place** je stejně jako předchozí nepovinným potomkem elementu **mtk:map**. U nástroje pro práci s místy můžeme nastavovat pouze dva atributy:

- **marker_icon**: určuje cestu k souboru s obrázkem znázorňujícím místo v mapě, výchozí hodnotou je
  "http://maps.google.com/mapfiles/kml/pal3/icon40.png"
- **marker_shadow**: určuje cestu k souboru s obrázkem znázorňujícím stín místa v mapě, výchozí hodnotou je prázdný řetězec

Výsledný dokument poté v prohlížeči může vypadat například takto.



Obr. 7: Ukázka výsledné aplikace (zdroj: autor)

V budoucnu by se měly možnosti nastavení komponenty ještě rozšířit. Kromě jiného by měla přibýt možnost zabezpečení komponenty. Zabezpečení by například mohlo spočívat v zákazu editačního režimu pro neautorizované uživatele. Dále by se mohly rozšířit možnosti grafické úpravy komponenty pomocí kaskádních stylů atd. S rozšíření možností nastavení bych také chtěl zapracovat na jeho zjednodušení. Toho by se mohlo například docílit vytvořením grafického editoru nastavení.

Závěr

Cílem mé práce bylo vytvořit webovou komponentu pro práci s digitálními mapami, která by mohla nahradit placený produkt Maptoolkit společnosti Toursprung. Pokrýt však všechny jeho nabízené nástroje by vydalo na více diplomových prací. Já jsem ve své práci vytvořil základ, na kterém se dá, a pevně věřím, bude dále stavět.

Vyvinutá komponenta slouží k ukládání, zobrazování a editaci objektů v mapách společnosti Google. Obsahuje editor pro vytváření tras a zájmových míst (POIs), který nabízí téměř totožné možnosti, jako konkurenční produkt (mj. magnetický a úsečkový nástroj pro zakreslování tras do mapy, formuláře pro editaci informací o objektech atd.). Všechny informace se ukládají pro pozdější použití do datového úložiště. Ačkoliv komponenta umí nyní spolupracovat pouze s databázovým systémem MySQL, je navržena takovým způsobem, že nebude v budoucnu problém jednoduše zajistit podporu i pro jiná datová úložiště. Vytvoření databázové struktury v úložišti (tabulek a jejich propojení) nám zjednoduší webový administrátorský nástroj.

Pro zvýšení škálovatelnosti a zlepšení výkonu komponenty by se v budoucnu mohl implementovat klastrovací nástroj[40]. Ten je již v současné době ve stádiu rozpracování. Další vylepšení by si zasloužila vrstva přístupu k datům, kde by se například mohlo zavést cachování dat. Prostoru pro další zdokonalování je ještě mnoho a jsem přesvědčen, že má diplomová je kvalitním výchozím bodem.

40 Nástroj pro shlukování velkého počtu objektů v malé oblasti do jednoho zástupného objektu (klastru)

Zdroje

[1] BELLINASO, Marco. Webové programování v ASP.NET 2.0 : problém, návrh, řešení. vyd. 1. [s.l.] : [s.n.], 2007. 648 s.

[2] YANK, Kevin; ADAMS, Cameron. Začínáme JavaScript : Základy programování, webové formuláře, DOM a Ajax. [s.l.] : [s.n.], 2006. 333 s.

[3] SUEHRING, Steve. JavaScript : Krok za krokem. [s.l.] : [s.n.], 2008. 335 s.

[4] Google Code [online]. c2010, 2010-12-02 [cit. 2010-12-26]. Google Maps JavaScript API V3 Reference - Google Maps JavaScript API V3 - Google Code. Dostupné z WWW: <http://code.google.com/intl/cs-CZ/apis/maps/documentation/javascript/reference.html>.