TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 - Informační technologie

Portál lokálních událostí

Portal of local events

Diplomová práce

Autor: **Bc. Ondřej Nedvídek**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

V Liberci 10. 5. 2011

Prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil autorská práva (ve smyslu zákona č. 121/2000 Sb. O právu autorském a o právech souvisejících s právem autorským).

Souhlasím s umístěním diplomové práce v Univerzitní knihovně TUL.

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č.121/2000 Sb. 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 ode mne právo požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Datum

Podpis

Poděkování

Tuto diplomovou práci bych nenapsal nebýt dvou osob, které mi pomohli ve chvílích, kdy jsem to nejvíce potřeboval, tímto bych jim rád poděkoval za pomoc. Děkuji Antonínu Hýžovi a vedoucímu mé diplomové práce Mgr. Jiřímu Vranému.

Abstrakt

Tato práce popisuje, jakým způsobem se postupuje při vývoji mashup aplikace, co musí aplikace splňovat a co vlastně toto označení znamená.

Podstatná část funkcionality je tvořena za pomocí GoogleMaps API, které má na starost interakci s uživatelem při práci s mapou. Právě na mapu je navázána podstata aplikace, ta spočívá ve spojení informace s lokalitou, které se informace bezprostředně týká. Jelikož funkce a využitelnost aplikace závisí na sdílení informací mezi jednotlivými uživateli, je výsledná webová aplikace vlastně formou sociálního webového systému.

Pro realizaci aplikace byla použita metodika Model View Controller(MVC), která je dnes využívána při vývoji profesionálních webových aplikací. Mezi použité nástroje patří PHP framework CakePHP, MySQL, ale hlavně JavaScript, který je používán jako nástroj pro práci s GoogleMaps API.

Výsledkem je portál lokálních událostí splňující standardy dnešní webové aplikace a zahrnující principy mashup aplikace, který uživateli dává možnost, sdílet s ostatními uživateli informace ve formě nejen textové, ale i vizuální.

Klíčová slova: GoogleMaps API, Mashup, MVC, CakePHP

Abstract

This paper describes how to proceed with the development of mashup applications, what criteria must the application meet and what that expresion means.

A substantial part of the functionality is created by using GoogleMaps API, which is responsible for interaction with the user working with a map. The map is linked body of the application, it is in conjunction with location information, information that directly relates. As the functionality and usability of the application depends on the sharing of information between users, is the final web apps actualy the web based on social system.

To realize the application was a methodology Model View Controller (MVC), which is now used to develop professional Web applications. Among the used tools include PHP framework CakePHP, MySQL, but mainly JavaScript which is used as a tool for working with GoogleMaps API.

The result is a portal of local events fully satisfying standards of today's web applications and containing the principles of mashup application that gives users the ability to share information with other users not only in text form but also visual.

Keywords: GoogleMaps API, Mashup, MVC, CakePHP

Obsah

1. Úvod

Naše okolí je prvním, co nás v životě zajímá a ani po zbytek života se to nezmění, ačkoliv se okruh bude pravděpodobně zvětšovat a potřeba informací o všem co se děje okolo nás bude až exponenciálně narůstat. V mladém věku nás zajímá, co se děje doma, čím jsme starší, okolí naší působnosti zvětšuje a chceme do něj zahrnout ulici, pak město, a nakonec co největší okruh, co jsme schopni obsáhnout.

Nicméně jedno se nemění, nejvíce nás zajímá okolí, v kterém se vyskytujeme většinu času. Takové okolí, pro většinu lidí, znamená například město, ve kterém žijí, jelikož to je celek, v jehož rámci jsme se schopni pohybovat v rozumném čase a ve většině případů určuje množinu možností, kterých můžeme využit, ať už pro zábavu, nebo pro vlastní profit.

Ve chvíli, kdy potřebujeme získat informace o těchto možnostech, tak musíme využít nějakého informačního kanálu, kterých je v dnešním světě spousta a každý musí sám vyzkoušet, jaký mu nejvíce vyhovuje, případně nějaká skupina z nich. Někomu stačí přečíst regionální deník, někdo si na základě těchto informací vyhledává data pomocí jiných zdrojů, aby se dozvěděl více. Jedním z takových zdrojů je fenomén dnešní doby internet.

Shrneme-li poznatky z předchozích odstavců, můžeme říci, že mnoho lidí vyžaduje získáváni informací, vázaných na nějaké svoje okolí. Proč bychom tyto informace nemohli podávat uživatelům ve formě vizuální informace, která jim v mnoha případech řekne nejvíce, a to z pohledu pozice na mapě. Uživatel má hned představu o tom, kde se daná událost stala, nebo se bude konat a naložit s touto informací dle vlastního uvážení.

Touto myšlenkou se zabývá i tato diplomová práce, která popisuje tvorbu mashup aplikace, zprostředkovávající informace uživatelům na základě vybrané lokace. Informace nabývají různého charakteru a je už pouze na uživateli, jakým způsobem bude volit požadavky, aby získal právě požadovanou množinu informací.

V diplomové práci jsou popsány jednotlivé aspekty a důraz je kladen na osvětlení jednotlivých funkcionalit z hlediska práce s GoogleMaps API, které souvisí s tvorbou mashup aplikací, vývojového prostředí s použitím frameworku PHP a metodiky návrhu, používané hlavně u návrhu webových aplikaci, Model View Controller.

2. Rešeršní část

2.1. Fenomén sociální sítě

Sociální sítě jsou zde již mnoho let, nicméně v posledních letech nabývají na síle a počty členů se raketovým tempem zvyšují. Důvodem je snadnost komunikace mezi členy těchto sítí. Asi mnoho lidí napadne pod pojmem „Sociální síť“ nějaká dnešní webová aplikace, která sdružuje lidi. Avšak tento termín tady byl již dávno předtím a označoval skupinu lidí, která měla podobné zájmy, nebo problémy a v rámci této skupiny byla tendence komunikace mezi jednotlivými členy, sdílení svých informací a zjišťování informací o jiných členech.

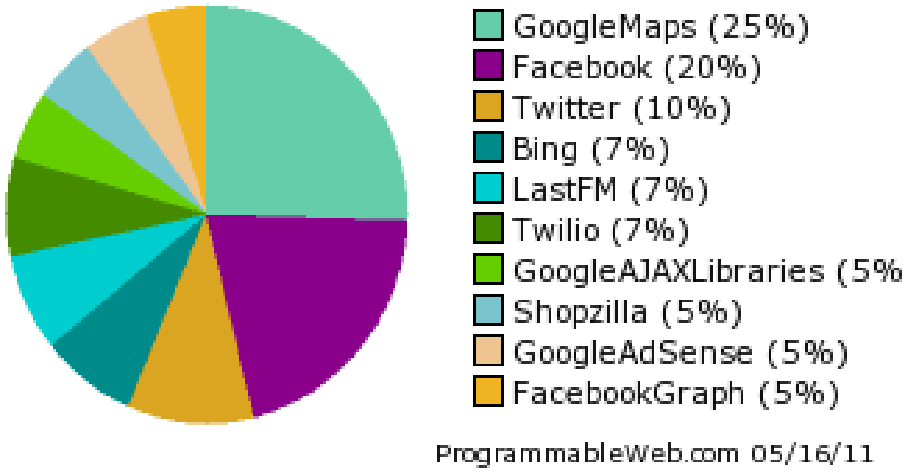
V dnešní době internetu je vytvoření a vedení takovéto sítě mnohem snadnější a proto jsme svědky obrovského boomu těchto sítí. Králem mezi těmito sítěmi je jistě Facebook s neskutečnými 600 miliony aktivních uživatelů (Leden 2011). Nicméně není jediným a z velkých hráčů stojí za zmínku např. sociální síť pro spíše profesionálnější prezentaci a sdílení informací LinkedIn. Z ryze českých můžeme uvézt Lide.cz, libimseti.cz.

Všechny tyto sítě pracují na principu sdružování uživatelů, kteří mají nějakou společnou potřebu získávání informací a zároveň jsou ve většině případů ochotní i přispívat informacemi vlastními.

2.2. Mashup aplikace

Mashup vychází z anglického slova, které by bylo možné do češtiny přeložit jako „mixáž“. Název se tedy co nejvýstižněji snaží vyjádřit princip tohoto typu aplikací, kdy můžeme jako Mashup aplikaci označit každou aplikaci, která pracuje s jiným zdrojem informací v reálném čase. V praxi to jsou aplikace, které pro svoji prezentaci nebo funkci využívají jiné webové aplikace.

Ať už se jedná o aplikace, které užívají pro přihlášení např. vašeho uživatelského účtu na Facebooku. Přestože se nepohybujete přímo na stránkách Facebooku, ale na stránkách této aplikace, můžete využívat některých funkčností, jako je udávání tzv. „liků“(znamení, že se uživateli něco líbí). Příkladem takové aplikace může být www.lamer.cz, kde můžete hodnotit příspěvky uživatelů. Aplikace je navázána na váš účet na Facebooku a pro ostatní uživatele, kteří si příspěvky také prohlížejí, je zobrazeno, kdo dal danému příspěvku již zmiňovaný „like“.



Obr 1: Graf využití API pro tvorbu Mashup aplikací zdroj: [1]

Dalším příkladem mashup aplikace je i www.idnes.cz, kde můžeme pozorovat v mnoha příspěvcích použití plně funkčních map pro zobrazování pozice, kde se stala událost o které se v článku píše.

Na obrázku „Obr 1: Graf využití API pro tvorbu Mashup aplikací zdroj: [1]“ vidíme, které z aplikací jsou v dnešní době nejvíce využívány k tvorbě „mashupů“. Jak již napovídala předchozí kapitola, trendem dnešní doby jsou sociální sítě a v prostředí tvorby mashup aplikací tomu není jinak. Druhou a třetí pozici tedy zaujímají Facebook a Twitter.

Nicméně na prvním místě vidíme GoogleMaps API, které je pro tuto práci nejdůležitější. Podle statistik programmableweb.com [1] je to dnes nejvyužívanější API pro tvorbu mashup aplikací a jeho služeb využívá každý čtvrtý vývojář, který se „mashupem“ zabývá.

Je nutné ještě podotknout, že mashup aplikace může samozřejmě používat více typů aplikací, ať už sociálních, mapovacích a ostatních. Vše již záleží na příčetnosti člověka, který má vývoj mashup aplikace na starost.

2.3. GoogleMaps API

Z předchozí kapitoly víme, GoogleMaps API je bezkonkurenčně nejvyužívanější prostředí pro tvorbu mashup aplikací, tím pádem i nejvyužívanějším prostředkem pro implementaci map do webové aplikace. Vývoj prostředí má na starosti asi největší internetová společnost dnešní doby Google. Ačkoliv GoogleMaps API není stěžejním projektem této společnosti, tak zaujímá podstatnou pozici na jejím portfoliu, což

zajišťuje vývojářům, kteří s tímto API zacházejí jistotu a maximální podporu při jeho využívání.

Toto prostředí je momentální chvíli ve dvou verzích a to V2 a V3. Vývoj verze V2 je již oficiálně ukončen a v případě, že již stávající systém neběží na této verzi, není důvod ji používat. Proto byla zvolena verze V3, kde vývoj nových funkcí stále pokračuje a do budoucna znamená větší potenciál pro přidání dalších funkcionalit do aplikace.

GoogleMaps API vývojářům nabizí nepřeberné množství funkcí a možnosti práce s mapou se stávají takřka nevyčerpatelné. Pro práci s tímto prostředím je na stránkách Google vypracována podrobná dokumentace [2]. Ta je členěna do několika částí, kde uživatel získá nejprve základní informace o prvotní inicializaci mapy ve své aplikaci, tato sekce je schována pod pojmem „Basics“ (základy).

Následující sekce jsou plné tutoriálů, jak zacházet s mapou komplexněji a využívat složitější možnosti práce s ní. Jsou zde uvedeny jednotlivé příklady použití, od vkládání značek na mapu, až po práci s listenery, které lze nadefinovat veškerým objektům v rámci zobrazení mapy. Jednotlivé příklady jsou popsány jak v textové podobě, tak vizualizovány přímo pomocí použití daného příkladu na cvičné stránce.

3. Programátorské nástroje a metody návrhu

Nástroje volené pro tuto práci jsou v dnešní době asi nejrozšířenějšími nástroji pro tvorbu webové aplikace. Důvodem této volby je hlavně veliké množství materiálů, které jsou dostupné v internetové podobě z důvodu velké základny vývojářů využívající tyto nástroje. Volba z tohoto pohledu byla jasná a padla na AMP, zkratka označuje spojení technologií Apache server + MySQL databáze + PHP.

Moderní přístup k vývoji webových aplikací jde ruku v ruce používaní MVC architektury a v mnoha případech se používá, k usnadnění a zejména zkvalitnění práce, frameworků.

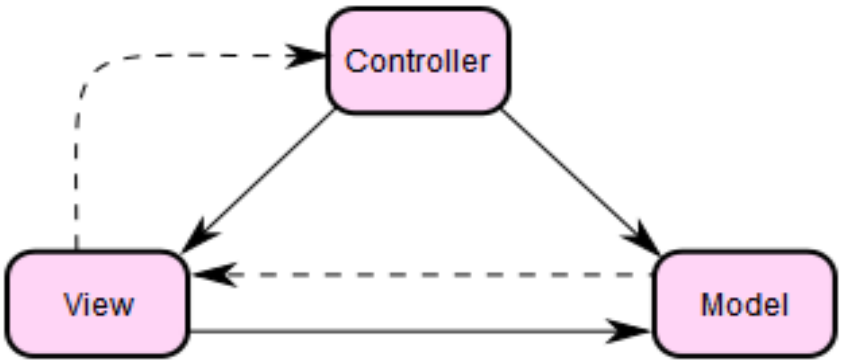
3.1. Nástroje

Funkčně nejpodstatnější část aplikace je realizována v jazyce PHP, jejíž struktura jednotlivých částí je vysvětlena v následujících kapitolách „3.2 Model View Controller“ a „3.3 PHP framework“. Databáze je spravována pomocí MySQL a celý web běží na Apache Server. Pro interaktivní práci uživatele s aplikací, ale hlavně kvůli práci s GoogleMaps API aplikace využívá interpretovaného programovacího jazyka JavaScript.

3.2. Model View Controller

Metodika návrhu MVC (Model View Controller) určuje, jakým způsobem by měl návrhář webové aplikace postupovat. Lépe řečeno, dělí funkčnosti aplikace na tři hlavní skupiny. Tímto striktním rozdělením je jednoznačně určena působnost jednotlivých částí, což má za přínos zpřehlednění aplikace a zejména jednodušší ladění jednotlivých částí. Odpadá tedy zažité psaní kódu do jednoho souboru, který řeší jak logiku aplikace, tak její prezentační část a někde, při tom všem, se systém ještě připojuje k databázi a požaduje potřebná data.

Další výhodou je snadnější změna jednotlivých částí, která nemá ovlivnit části ostatní. Toho můžeme využít např. ve chvíli, kdy se rozhodneme pro změnu prezentační(View) části, ať už z důvodu modernizace vzhledu, nebo pouze poskytnutí uživateli, možnosti vybrat si vyhovující vzhled prezentační části v nastavení aplikace. V této chvíli zůstávají požadavky na funkci systému z hlediska správy dat stále stejné a není tedy nutné měnit přístup k datům(Model), ani logiku s jakou jsou data zpracovávána(Controller).



Obr 2: Model-View-Controller zdroj: [12]

Na obrázku „Obr 2“ je znázorněna funkce metodiky, kdy uživatel podává požadavky na systém. Požadavky jsou ve většině případů dvojího typu, buď se jedná o přechod na jinou stránku nebo o odesílání formulářů. Při přechodu na další stránku aplikace je vždy v první řade volán kontrolér který v případě potřeby zažádá o potřebná data, části *model* a ten je vrátí. Přijatá data je v mnoha případech nutné nejprve zpracovat a následně je kontrolér může poslat na *view*.

Část *view* již data nijak neupravuje po významové formě, ale má na starosti pouze jejich prezentační část. Z tohoto popisu práce metodiky MVC lze tedy vycházet při určení práce jednotlivých prvků.

- Model – část určená pro práci s daty. Ve většině případů se jedná o práci s databází, ale data může model získávat i z jiných zdrojů, jako je např. XML atp. Získaná data od *kontroléru* je v některých případech nutné validovat a upravit do formátu nutného pro uložení např. do databáze. V případě že *kontrolér* o data zažádá, *model* vyřídí potřebnou práci pro získání dat a následně je předá.
- View – Tato část je určena pouze pro prezentaci dat. Je to jediná část se kterou uživatel pracuje. Funguje jako rozhraní se zbytkem aplikace. Uživatel přes tuto část ovládá aplikaci a ta mu zároveň vrací požadované výsledky. Tato část s daty nijak nepracuje, ale pouze je vhodnou formou prezentuje(zobrazuje) uživateli.
- Controller – Tato část má na starost veškerou logiku aplikace. Jejím úkolem je reakce na požadavky uživatele. Ve chvíli, kdy uživatel pošle požadavek, je předán této vrstvě. Vrstva požadavek vyhodnotí a v případě potřeby vyřeší nutné

výpočty a zachová se podle implementované logiky. Následně vhodný soubor dat posílá na *view* pro prezentaci výsledků.

3.3. PHP framework

Framework slouží, jako softwarová podpora pro programování. Hlavním cílem při používání frameworku je zjednodušení vývoje softwaru.

Co musí obsahovat není přesně definované, zpravidla to jsou podpůrné programy, knihovny metod a tříd, případně vlastní strukturu, která vývojáře vede k používání nějakého postupu při vývoji aplikací. Cílem vedení vývoje, bývá zahrnutí návrhových vzorů a jiných metodik při návrhu aplikace.

PHP frameworky jsou tedy podpůrnými prostředky pro práci s PHP. Oba frameworky, které jsou dále v kapitole zmíněny jsou volně stažitelné přímo na oficiálních stránkách a jejich instalace probíhá formou zkopírování příslušných knihoven na určené místo, společně s adresářovou strukturou pro vývoj dané aplikace.

Tyto dva frameworky jsou vyvíjeny s otevřeným zdrojovým kódem, což umožňuje vývojářům si modifikovat i kód v knihovnách frameworku, tento způsob by měl být v jejich vlastním zájmu upřednostněn až v posledním a velice dobře odůvodněném případě, vzhledem k možné nestabilitě frameworku.

Vlastnosti, kvůli kterým byly oba tyto pracovní nástroje zvoleny, jsou implementace MVC architektury a velká uživatelská základna, která zajišťuje velký počet informačních zdrojů.

3.3.1 ZendFramework

U nás velice známý framework pro vývoj v PHP, vyvíjený americkou společností Zend Technologies USA, Inc.. Z důvodu rozšíření u nás lze čerpat i z velkého množství informací v češtině.

Charakteristika:

- Opravdu rozsáhlá knihovna metod a tříd – velice robustní (ve verzi 1.9 - 26 MB)
- Přehledná adresářová struktura aplikace
- V začátcích strmější křivka učení
- Nepříliš uživatelsky přívětivá instalace ovladače databáze, pomocí „.ini“ souboru
- Složité přidávání vlastních component (podpůrný kód kontroléru)
- Méně přehledná oficiální dokumentace

- Nepříjemně dlouhá indexace knihovních souborů při spouštění vývojového prostředí eclipse a celkově pomalejší práce na méně výkonném PC

3.3.2 CakePHP

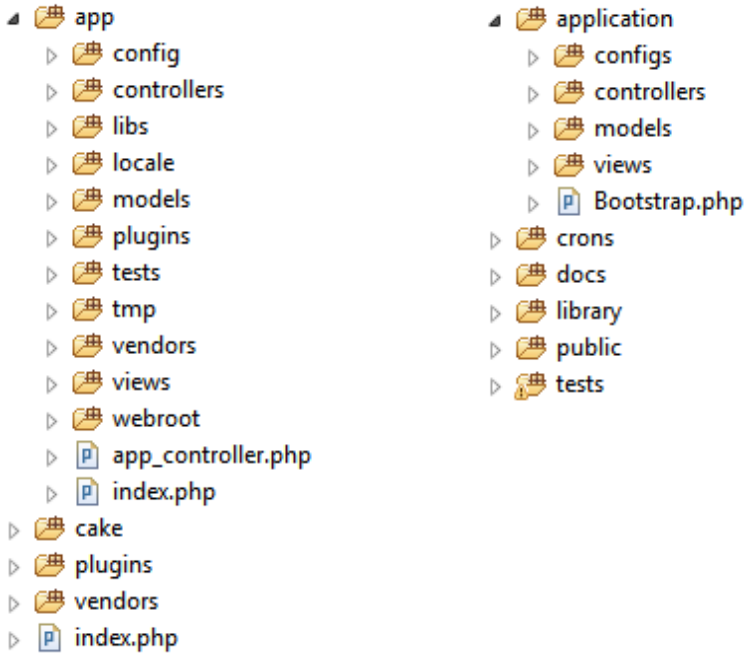
Na českém trhu nepříliš známý framework, který si ale v posledních letech získává na oblibě. Disponuje velkou uživatelskou základnou, hlavně ve světě, což vývojáře v CakePHP odkazuje zejména na anglicky psané zdroje informací.

- Velká knihovna metod a tříd (verze 1.3.6 – 6 MB)
- Ze začátku méně přehledná struktura aplikace
- Pozvolnější křivka učení
- Instalace ovladače databáze pracuje s polem v souboru database.php, její užívání je přehledné a uživatelsky přívětivé
- Jednoduché a intuitivní přidávání component, helperů i behaviour (podpůrné kódy kontroléru, view a modelu). Spočívá v jednoduchém nadefinování názvu tříd prvků do polí $components, $helpers, $behaviours.
- Velice přehledná a dobře členěná oficiální dokumentace
- Rychlejší indexace i práce ve vývojovém prostředí eclipse

3.3.3 Zvolil jsem CakePHP

Přes předchozí zkušenosti se ZendFramework, bylo rozhodnuto pro CakePHP. Důvodů bylo hned několik a jsou patrné z výše uvedených charakteristik. Největší výhodou byla dobře zpracovaná oficiální dokumentace, která je opravdu výborně dělená a hledání potřebných informací znamená otázku několika vteřin.

Dalším důvodem je velice dobře členěná adresářová struktura, která obsahuje všechny potřebné prvky pro snadnou a intuitivní práci s framworkem. Jak struktura vypadá je vidět na obrázku „Obr 3: CakePHP a ZendFramework adresářová struktura“, kde je vidět, ze ZendFramework působí na první pohled přehledněji a jednodušeji. Po několikadenním užívání, však CakePHP působí daleko přímočařejším a ve výsledku přehlednějším dojmem. Z důvodu práce hlavně na menších projektech, nebyl zaznamenán případ, kdy by měly knihovny ZendFramework navrch, naopak jednoduchost použití některých částí CakePHP, která je na domovských stránkách nazývána magií, kterou v některých případech bezpochyby je, předčila veškerá očekávání.



Obr 3: CakePHP a ZendFramework adresářová struktura

Další velkou výhodou je práce s podpůrnými třídami, které nejprve dědí vlastnosti příslušné třídy(*AppHelper*, *AppComponent*, *AppBehaviour*) a následně je stačí vložit do určené adresářové struktury, čímž jsou připraveny k použití. Podpůrné třídy jsou členěné do výše zmíněných skupin, behaviours, helpers a components, jejichž kód je použit pro podporu částem aplikace odpovídajícím MVC architektuře. Behaviours se používají k přidání vlastnosti modelu, helpers k view a v neposlední řadě components u kontrolérů.

Posledním důležitým aspektem vývoje v CakePHP, je rychlost a to nejen vývoje aplikace, kdy inicializace základních parametrů, jako je přístup do databáze, určení šifrovacího klíče pro hash a zobrazení první stránky, je otázka pár kliknutí myši a napsání několika málo údajů. Tak následná práce s knihovnami CakePHP, které umožňují kompromis mezi dobře vybaveným souborem knihovních tříd a velikostí, kterou výsledná aplikace bude zabírat na serveru, což má i vliv na výkon práce při vývoji aplikace, kdy odpadá stále se opakující a velice otravná indexace nepřeberného množství souborů, jak tomu nastávalo při práci s ZendFramework.

4. Uživatelské možnosti aplikace

V této kapitole jsou popsány principy na kterých aplikace funguje. Je zde nastíněn účel jednotlivých funkcí aplikace a myšlenka s kterou bylo k aplikaci při jejím vývoji přistupováno.

4.1. Popis funkce aplikace

Aplikace pracuje z pohledu dnešní doby standardním přístupem. V první řadě je nutné, aby se uživatel zaregistroval a následně i přihlásil do samotné aplikace. Po přihlášení se uživateli zobrazí úvodní stránka „Home“ (viz.Obr 4: Home).

Na této stránce je vhodné uvézt informace ohledně webu, které mohou být pro uživatele zajímavé. Pro příklad by zde provozovatel stránek mohl uvézt novinky, připravované inovace webu, varovné zprávy a podobně.

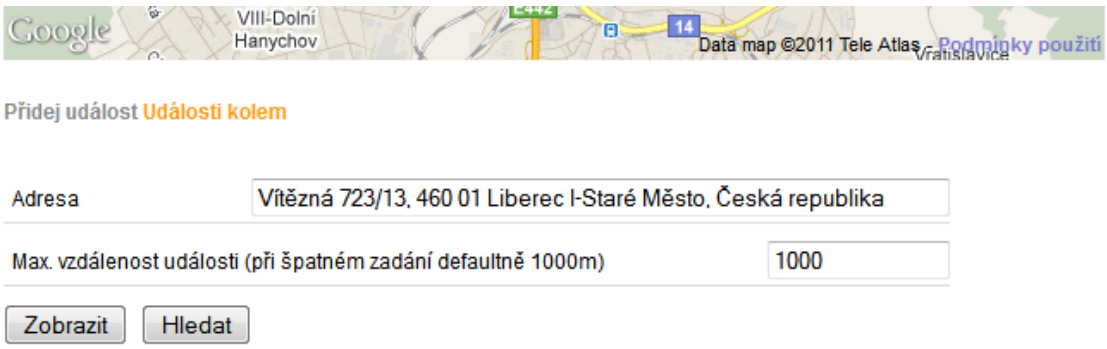
Obr 4: Home

Následně uživatel pokračuje na „Výběr města“ (viz.Obr 5: Výběr města), ve které má zájem zjišťovat jednotlivé události (o událostech viz. následující kap.).

Obr 5: Výběr města

Po výběru daného města již nic nebrání v práci s danou lokalitou. Možnosti uživatele pracovat s danou lokalitou jsou následující.

- Zobrazení na základě typu události
- Řazení jednotlivých událostí - abecedně, dle doby začátku události/konce události, uživatelů, titulků
- Zobrazování událostí na základě pozice uživatele – uživatel může zvolit zobrazení na základě vlastní pozice, čehož docílí volbou položky „Události kolem“. Následně si nastaví svůj akční rádius a pomocí tlačítka „Zobrazit“ si nechá události spadající do tohoto rádiusu vykreslit(ukázka formuláře viz. „Obr 6: Události kolem“).
- Přidávání a mazání vlastních událostí – každý uživatel se musí před používáním aplikace přihlásit, to mu ale zároveň dává možnost vázat události na svůj účet. Jiní uživatelé pak mají možnost zjistit, kdo s uživatelů událost přidal a na tomto základě se rozhodovat o věrohodnosti.



Obr 6: Události kolem

4.2. Typy události

Události, se kterými může uživatel v aplikaci pracovat jsou rozděleny do tří skupin.

- Novinky – v této skupině se nacházejí všechny události zejména informativního charakteru. Novinky jsou standardní zprávy vázané na danou lokalitu. Lze mezi ně zahrnovat např.: problémy s dopravou v určitých ulicích, omezení funkčnosti úřadů, nové budovy ve městě a další.
- Kulturní akce – tento typ událostí se vztahuje ke všem typům kulturních akcí ve městě. Tento typ akce uvítají zejména ti co provozují kina, divadla, festivaly

a jiné větší kulturní akce. Nicméně ho mohou zvolit i samotní umělci pro zviditelnění vlastní osoby, kdy určí pozici na mapě s titulkem např. „Dnes večer jenom pro Vás hraje na kytaru Jan Novák“.

- Reklamní akce – poslední typ události je důležitý hlavně pro firmy, které mohou touto formou prezentovat svoje výrobky, nebo služby v jednotlivých lokalitách. Do tohoto typu akce se mohou schovat výprodeje, akční nabídky, novinky na trhu apod. Pro příklad může automobilka pro každý den určovat zajímavé akce pro své zákazníky nebo uvézt nový automobil, který je možné již objednat na její pobočce.

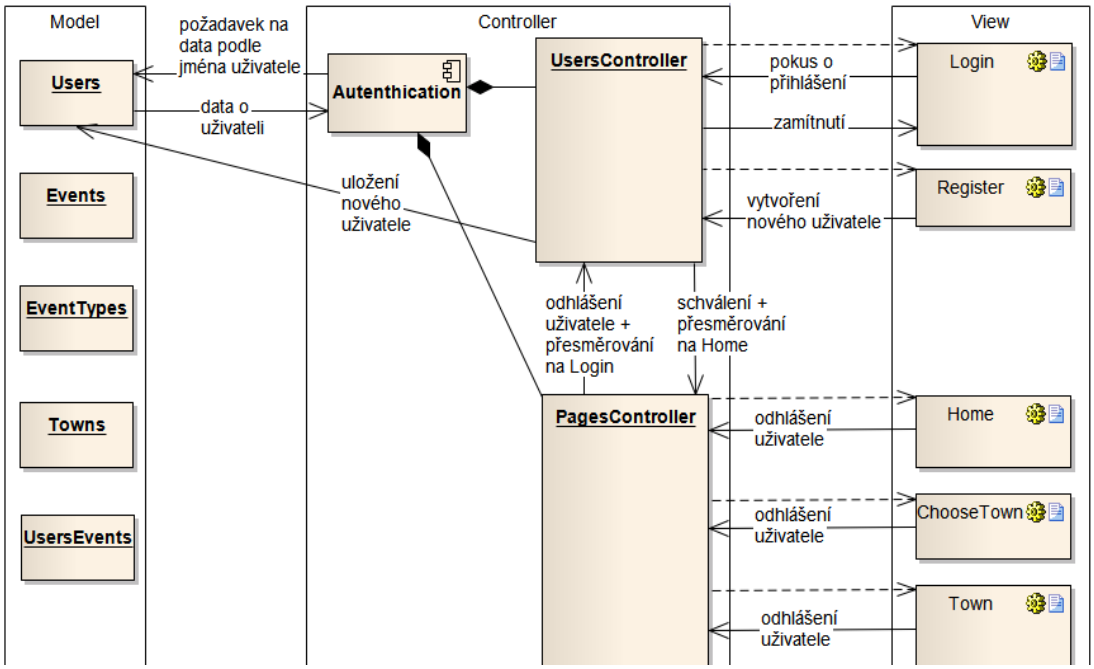
5. Příprava pro implementaci

V kapitolách „5.1.“ a „5.2.“ je znázorněna reakce systému na požadavky uživatele. Na obrázcích je systém rozdělen do částí podle MVC metodiky a požadavky uživatele jsou znázorněny směrem od view ke kontroléru části. Reakce na systému jsou popsány textovou formou a korespondují s vizuální reprezentací systému na obrázku.

V dalších dvou kapitolách je nejprve znázorněna struktura databáze a princip přepočtu souřadnicového systému GoogleMaps API na metrový systém, za účelem určení událostí v okruhu uživatele.

5.1. Správa uživatelského účtu

Na obrázku „Obr 7: Správa uživatelského účtu“ lze pozorovat, že logiku autentizace uživatelů řeší objekt *Authentification*, který vyřizuje veškerou komunikaci s databází na této úrovní. Objekt je volán vždy před každým vyřízením požadavku uživatele po přihlášení do aplikace. V případě, že z nějakého důvodu je vyhodnoceno, že se jedná o požadavek od neautentizovaného uživatele, tak není vykonán a uživatel je přesměrován na *Login*.



Obr 7: Správa uživatelského účtu

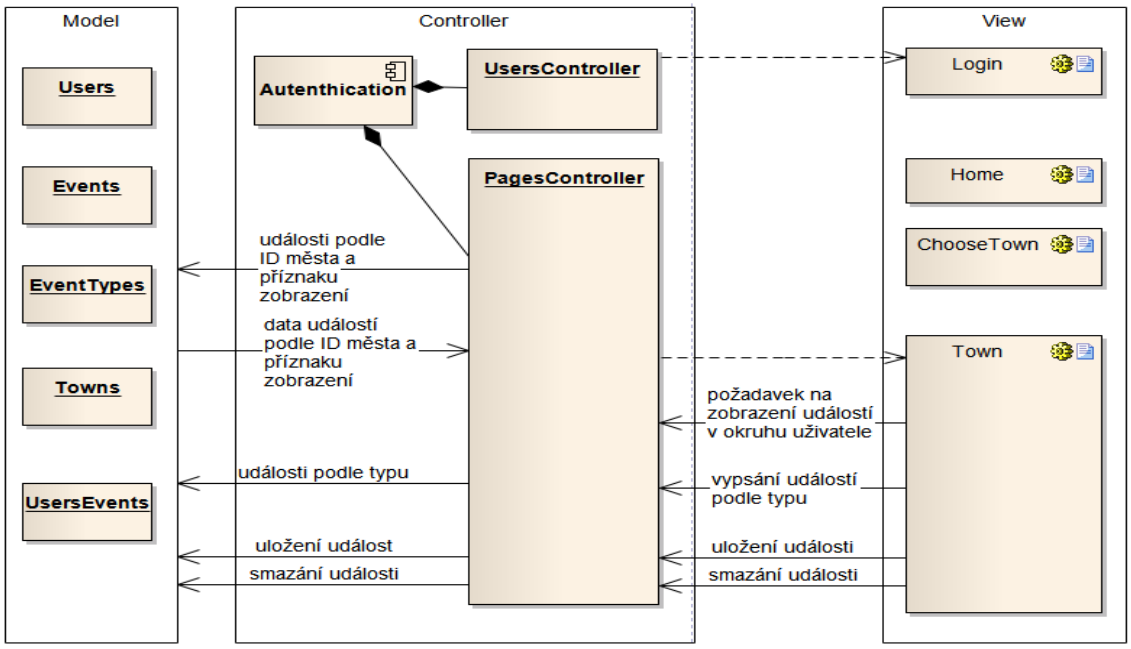
Kontrolér *UsersController* má na starost práci s daty při přihlášení uživatele a případně jeho odhlášení. Ve chvíli, kdy je z nějakého view zažádáno o odhlášení aktuálního uživatele, tak *UsersController* tak vykoná pomocí objektu *Authenticate* a přesměruje na Login. Registrace uživatele probíhá formou odeslání formuláře z view

Register, *UsersController* data zpracuje a pošle je na příslušný model (*Users*), který je uloží do databáze.

5.2. Vizualizace, ukládání a mazání událostí

Ve chvíli, kdy je vybráno město, je zaslán požadavek na Model o data událostí, které se týkají zvoleného města. Model data vrátí a kontrolér je předá na zobrazení view *Town*. *Ve* chvíli, kdy uživatel zvolí možnost vypsání si pouze určitého typu události(např. kulturní akce), tak *PagesController* volá model s parametry města a parametrem typu události. Model vrací požadovaná data, která jsou zpracována a zobrazena stejně jako v předchozím případě.

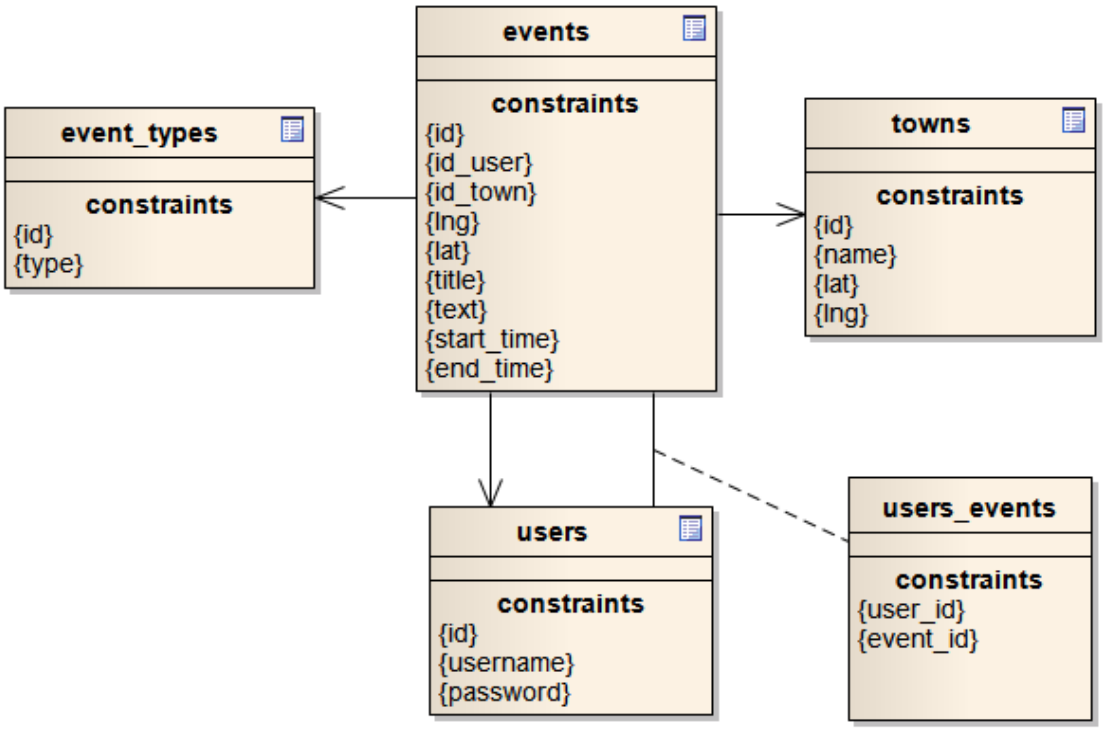
Pokud uživatel požaduje zobrazit události ve svém okolí, předává na *PagesController* požadavek „zobrazení událostí v okruhu uživatele“, ten zažádá o veškeré události z databáze a pomocí uživatelovi pozice a rádiusu vyseparuje odpovídající události a uloží příznak o jejich zobrazení pro další vizualizaci stránky. „Příznak zobrazení“ je položka v tabulce, která řeší vztah M:N mezi událostmi(*Events*) a uživateli(*Users*). Tento vztah znamená, že v jednu chvíli může mít uživatel požadavek na zobrazení více událostí, ale zároveň jedna událost bude požadována k zobrazení pro více uživatelů. Ve chvíli kdy je příznak uložen, je přesměrován požadavek na metodu pro zobrazení *Town*, který v první řadě vyzíská potřebná data v závislosti na příznaku zobrazení, městu a případně na typu události, který si uživatel pro vizualizaci zvolí. Data posílá na *Town*, kde se opět pouze prezentují v požadovaném formátu.



Obr 8: Vizualizace, ukládání a mazání událostí

5.3. Návržená databáze

Předpokladem je, že název města může být v rámci světa stejný na více místech, takže jeho lokalita v lng a lat může nabývat pro dvě města se stejným názvem různých hodnot. Na obrázku je vidět, že mezi tabulkou *events* a *users* jsou hned dva vztahy, kterých může nabývat. Prvním je vztah M:N, který byl vysvětlen již v předchozí kapitole, jako řešení problému při správě příznaku pro zobrazení, které je nutné vyřešit ve chvíli, kdy se jednotliví uživatele rozhodnou pro vizualizaci událostí ve svém okolí(v okruhu svojí pozice). Druhým vztahem je 1:N, který znázorňuje vztah, že každá událost patří nějakému uživateli. Výsledná databáze je ve 3. normální formě.



Obr 9: Schéma databáze

5.4. Vyhledávání událostí na mapě z pozice uživatele

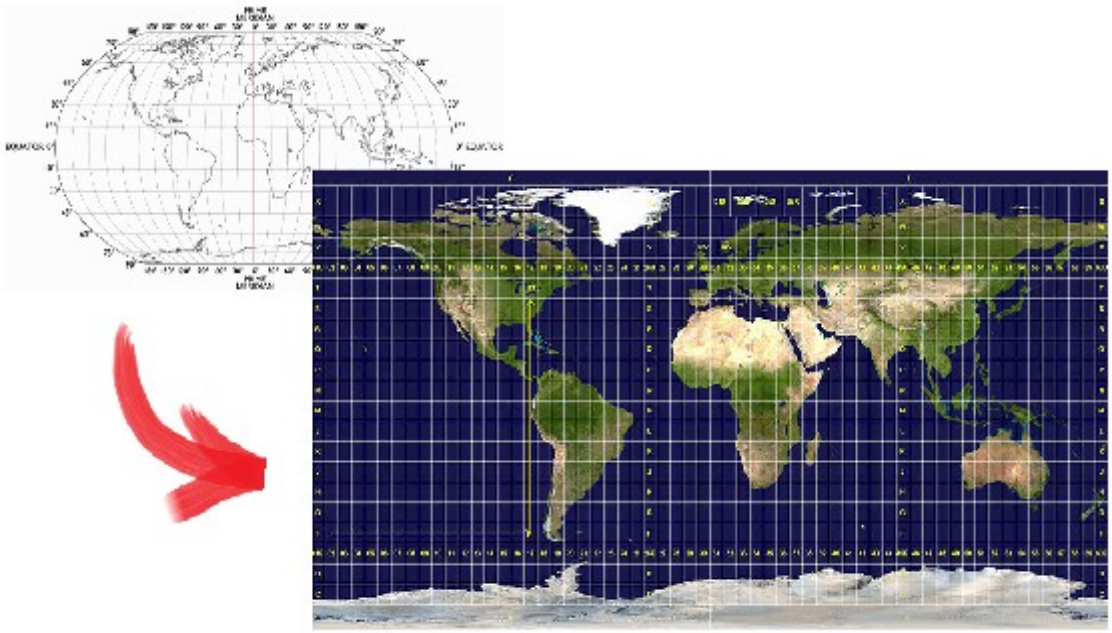
V kapitole „4.1 Popis funkce aplikace“ je popsána funkčnost z pohledu uživatele. V této kapitole si nastíníme princip na jakém pracuje aplikace ve chvíli, kdy uživatel této funkce použije.

V GoogleMaps API se pracuje ve standardním souřadnicovém systému, který používá např. GPS. Tento systém vyvinutý americkou armádou se označuje zkratkou

WGS-84 (World Geodetic System v poslední verzi z roku 1984). Pracuje na principu rozdělení polohy země na stupně, minuty a vteřiny zeměpisné délky a šířky.

Právě z tohoto principu vyplývá problém, když potřebujeme zjistit vzdálenost mezi dvěma body. Pro zjištění této vzdálenosti byl v aplikaci implementován mechanismus, který nejprve převede souřadnice ze systému pracujícího ve stupních do jiného, který pracuje v metrech. WSQ-84 je tedy nejdříve převeden na vhodný metrový systém, a to UTM (Universal Transverse Mercator), který je obdobně jako v prvním případě vyvinutý americkou armádou.

Díky tomu, že oba systémy vyvinula a dále provozuje stejná organizace, bylo v jejich zájmu vytvořit mechanismus pro přepočet mezi nimi. Převod mezi systémy je podrobně vysvětlen viz. [3].



Obr 10: Převod souřadnicových systému - WGS-84 na UTM zdroj: [13],[14]

6. Implementace

V této kapitole je popsána samotná implementace dané aplikace. V jednotlivých částech kapitoly je vysvětlena práce s CakePHP, kde jsou nastíněny požadavky na nastavení databáze a práce s ní. Možnosti a smysl kontroléru a osvětlení důležitých vlastností a metod. Dále je zde přiblížena práce s GoogleMaps API, od inicializace mapy, až po používání prostředků prostředí, jako jsou listenery, značky a vyznačování oblastí na mapě.

6.1. Databáze v CakePHP

Na ukázce kódu vidíme povinné parametry, které je nutné pro správné spojení s databází nastavit. Jsou to standardní parametry, jako typ používané databáze a přístupové údaje společně s cestou k dané databázi. Více informací o nastavení databáze viz. [4].

```
var $default = array(
        'driver' => 'mysql',
        'persistent' => false,
        'host' => 'localhost',
        'login' => 'root',
        'password' => 'heslo',
        'database' => 'community_web'
    );
programovací jazyk: PHP, umístění: app/config/database.php
```

Jak je uvedené v dokumentaci CakePHP, pro používání databáze je nutné vytvořit příslušné třídy pro jednotlivé tabulky databáze, které odpovídají jejímu návrhu z kapitoly „5.3 Návržená databáze“. Výsledné třídy jsou *EventType*, *Event*, *Town*, *User* a *UsersEvent* , jak je vidět názvy korespondují s názvy tabulek.

Podle zásad MVC, je všechen kód, který se týká práce s databází, řešen pouze v rámci části „model“. CakePHP tento způsob programování aplikace podporuje, až na helper *Paginate*, jehož podmínky pro získávání dat se definují v rámci příslušného kontroléru viz. kapitola „6.5.5 Výpis událostí – Paginator“.

Na následující ukázce metody *getAllEventsDependsOnTown($id)*, je vidět, jak lze získat data z tabulky *Event*, za použití zděděné metody *find()*. Metoda vrácí všechny

data, která odpovídají požadavku na *id* města. Další možnosti práce s databází pomoci CakePHP jsou přehledně uvedeny opět v dokumentaci frameworku.

```
function getAllEventsDependsOnTown($id){
    return $this->find('all',
      array('conditions' => array('Event.id_town' => $id)));
}
programovací jazyk: PHP, umístění: app/models/event.php
```

6.2. Přihlašování uživatelů pomocí CakePHP Authenticate

Přihlašování je dnes již standardní potřebou většiny webových aplikací. Proto je v CakePHP, implementována componenta *Authenticate*.

Authenticate je nutné nejdříve nastavit pro používání daným kontrolérem. Vzhledem k tomu, že byl požadavek na aplikaci, aby byla přístupná v celém rozsahu(mimo stránky pro registraci) pouze v případě, že je uživatel přihlášen, tak bylo zvoleno globální nastavení componenty *Authenticate* v kontroléru *AppController*, od kterého jsou odvozeny všechny kontroléry v aplikaci.

```
class AppController extends Controller {
    var $components = array('Auth', 'Session');

programovací jazyk: PHP, umístění: app/app_controller.php
```

Metoda *beforeFilter()* je vždy volána před všemi ostatními metodami kontroléru. V této metodě nastavíme pomocí metody *alow()*, že nepřihlášený uživatel má přístup pouze na stránky pro registraci. Ve chvíli, kdy systém vyhodnotí, že uživatel není z nějakého důvodu oprávněn k práci s aplikací, reaguje zamítnutím přístupu a sdělí uživateli důvody odmítnutí.

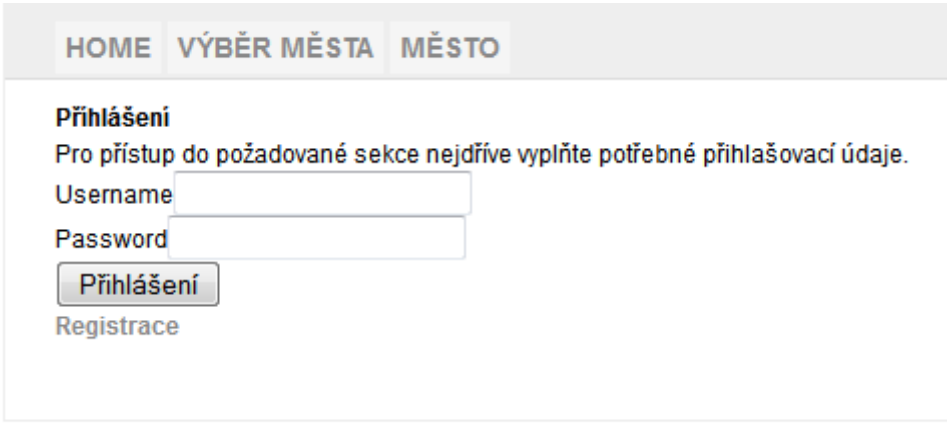
V případě, že chyba nastala při přihlašování uživatele, je zobrazena zpráva definovaná v *$this->Auth->loginError*. Další případ neoprávněného pokusu o přístup do aplikace může být stav, kdy uživatel nevyplnil požadované údaje, nebo jejich trvání vypršelo a z toho důvodu byl uživatel odhlášen. V tuto chvíli systém nepustí uživatele do systému a upozorní ho zprávou nadefinovanou v *this->Auth->authError*.

```
 function beforeFilter(){
    $this->Auth->allow(array('register'));
    $this->Auth->loginError = "Přístup byl odepřen. Vaše
uživatelské jméno nebo heslo je vyplněno špatně.";
    $this->Auth->authError = "Pro přístup do této sekce
nejdříve vyplňte potřebné přihlašovací údaje.";
    }
```

programovací jazyk: PHP, umístění: app/app_controller.php

Na obrázku „Obr 11: Reakce systému v případě neoprávněného přístupu“ je vidět reakce systému v případě, že se uživatel pokouší dostat do jiné než přihlašovací sekce v době, kdy mu již vypršel čas přihlášení z důvodu nečinnosti nebo v případě kdy opomenul vyplnit nejdříve potřebné přihlašovací údaje.

Systém reaguje stejným způsobem, ale za použití zprávy viz. loginError v případě, že uživatel použije špatné přihlašovací údaje.



Obr 11: Reakce systému v případě neoprávněného přístupu

6.3. PagesController

Hlavní řídící a výpočetní třídou celé aplikace je třída *PagesController*. Tato třída přebírá veškerou rozhodovací funkci ve chvíli kdy se uživatel přihlásí do systému. Řeší tedy veškerou logiku aplikace, kromě přihlašování, které má na starost třída *UsersController* zmíněná v předchozí kapitole. Podle MVC zastává třída PagesController funkci kontroléru, je tedy spojnicí mezi modelem a view, v tomto případě řídí veškerý tok dat hlavně mezi zobrazovanou mapou a databází, ve které jsou

uložena data s mapou související. Dalším úkolem této třídy je výpočet a určení dat, která budou zobrazována v případě, že uživatel zvolí nějakou formu požadovaného zobrazení událostí, případně svoje události nějakým způsobem spravuje.

```
class PagesController extends AppController
{
  var $name = 'Pages';
  var $uses =
     array('Event','EventType','User','Town','UsersEvent');
  var $helpers = array('Paginator');
..
programovací jazyk: PHP, umístění: app/app_controller.php
```

V kódu uvedeném výše jsou vidět atributy třídy. Atribut $uses určuje, které tabulky z databáze třída využívá k práci, respektive se jedna o třídy, které zastřešují práci přímo s databází a třída *PagesController* tedy pracuje s daty, které příslušné třídy vrací. Dalším atributem je $helpers, kde je použit prvek přímo z knihovny CakePHP, pro práci se stránkovým výpisem dat, viz. kapitola „6.5.5 Výpis událostí – Paginator“.

```
..
  function town($idTown = null, $idEvent = null){
   if(empty($idTown)){
     $sessionTown = $this->Session->
       read('Auth.User.SelectedTown');
     if (!empty($sessionTown)){
      ..
     }
   }
     $selectedTown = $this->Town->getTownById($idTown);
     $this->Session->write('Auth.User.SelectedTown',
        $selectedTown);
     $this->data['selectedTown'] = $selectedTown['Town'];
  ..
programovací jazyk: PHP, umístění: app/app_controller.php
```

Action *town()* je volána ve chvíli, kdy je požadován výpis stránky *town.ctp*, což je hlavní stránka pro práci s mapou vázaná na nějakou lokalitu(město). Action

se označují všechny metody v rámci CakePHP, které následně odkazují na stejně pojmenované stránky pro zobrazení. V tomto případě metoda *town()* má na starost zobrazení pouze stránky *town.ctp*.

V první části kódu metoda nejprve zjišťuje, jestli bylo zvoleno uživatelem město, které se má zobrazit. Pokud bylo v předchozím kroku zvoleno výběrem ze seznamu možných měst, je informace uložena v proměnné *$idTown,* předané v hlavičce *town()*.

V případě, že nebylo zvoleno, tak action *town()* zjišťuje jestli už uživatel někdy během svého přihlášení zvolil město, v jehož lokalitě se chce pohybovat a následně jen přešel z nějakého důvodu do jiných částí aplikace a chce se vrátit, aniž by měnil požadovanou lokaci. V tomto případě je po každém výběru města uložena do session informace o výběru a je uchovávána až do chvíle, kdy se uživatel odhlásí ze systému.

Další metody a funkční části třídy *PagesController*, jsou uvedeny v kapitolách, které jsou rozdělené v závislosti na jejich funkci.

6.4. GoogleMaps API

Veškerý kód, který se týka práce s prostředím GoogleMaps API V3 se nachází v souboru app\webroot\js\map.js.

6.4.1 Inicializace prostředí GoogleMaps API V3

Aby bylo možné používat samotné prostředí GoogleMaps API V3, je nutné ho v první řadě inicializovat. Toho docílíme přidáním skriptu do hlavičky stránky a veškeré nástroje pro práci s prostředím jsou připraveny k použití. Script přidáme níže uvedeným způsobem.

```
<?php echo $this->Html->
script('http://maps.google.com/maps/API/js?
sensor=false&language=cz'); ?>
```

```
programovací jazyk: HTML,PHP,
umístění: app/views/layout/default.ctp
```

Použitý způsob vyjadřuje přidání pomoci Html Helperu ve frameworku CakePHP. Můžeme si povšimnout, že už při inicializaci scriptu lze nastavit parametry, pro práci s mapou. V tomto případě to je parametr *sensor* a *language.* Parametr *sensor*

určuje, jestli zařízení, na kterém uživatel mapu používá, má zároveň možnost určit svojí polohu. Nejčastěji se jedná o mobilní zařízení obsahující GPS modul. V případě této aplikace bylo zvoleno nastavení *sensor=false*, tedy zamezení používání sensoru.

Language nám určuje, jaký jazyk je zvolen při práci s mapou, tedy v jakém jazyce bude prostředí GoogleMaps API s námi komunikovat.

6.4.2 Inicializace mapy

Samotnou mapu zobrazíme pomocí konstruktoru objektu *google.maps.Map.* Pro správné volání konstruktoru je nutné správně nadefinovat jednotlivé parametry. Ukázka definice klíčových parametrů aplikace je vidět níže.

```
function initialize() {
  var mapDiv = document.getElementById('map-canvas');
  map = new google.maps.Map(mapDiv, {
    center: new google.maps.LatLng(selectedLat,
selectedLng),
    zoom: 13,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  });
..
$(function () {
   initialize();
..
programovací jazyk: JavaScript, umístění: app/webroot/js/map.js
```

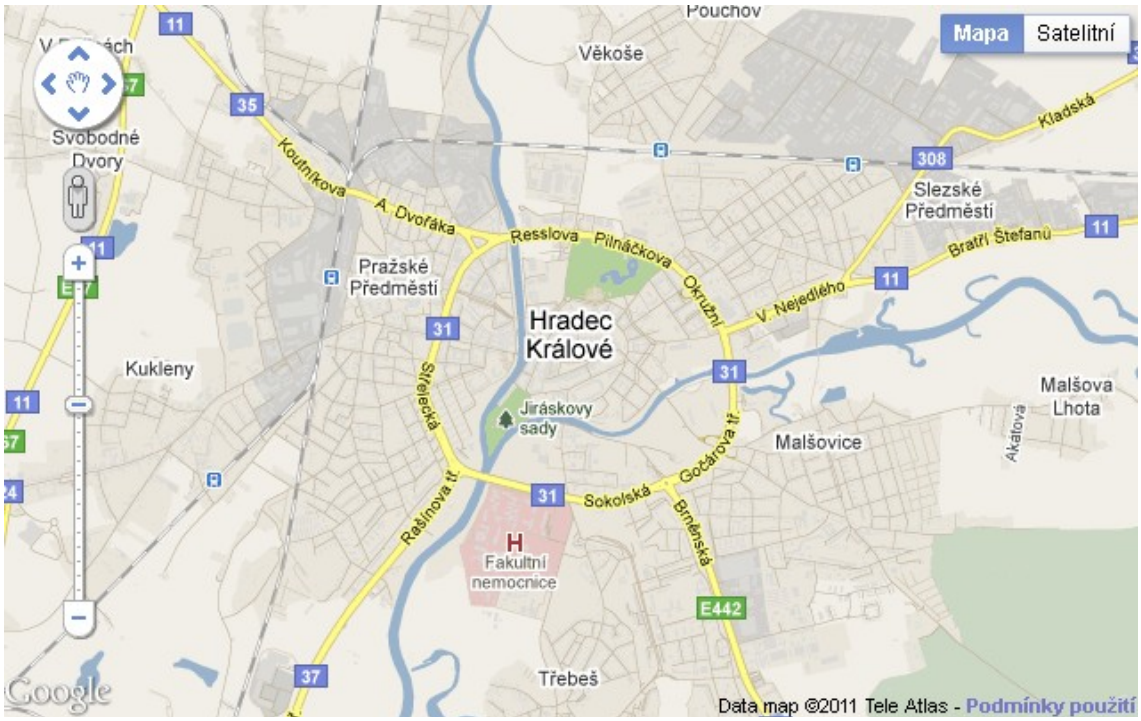
Metoda initialize() je volána ve chvíli, kdy je DOM připraven, což je zajištěno použitím knihovny jQuery a jeji metody .ready(), respektive jejího zkráceného ekvivalentu $(handler).

V ukázce kódu vidíme výše zmiňovaný konstruktor google.maps.Map, kde hned prvním parametrem, který mu je předán, je ukazatel na HTML prvek stránky, kde požadujeme mapu zobrazit. Tomuto prvku je nutné nadefinovat jeho velikost, jinak se mapa nezobrazí.

Dalším parametrem je specifikace pozice středu mapy, kterou určí objekt třídy *google.maps.LatLng*. Konstruktoru třídy předáváme číselné údaje o zeměpisné šířce a zeměpisné délce.

Parametr zoom určuje, jak již název napovídá, úroveň přiblížení mapy. Poslední parametr, který je v aplikaci nastaven, je *mamTypeId*. Možnosti nastavení tohoto parametru odpovídají konstantam:

- *MapTypeId.ROADMAP* – klasické zobrazení mapy
- *MapTypeId.SATELLITE* – satelitní zobrazení (zobrazení složené ze satelitních snímků)
- *MapTypeId.HYBRID* – kombinace výše zmíněných zobrazení
- *MapTypeId.TERRAIN* – mapa s teréním informacemi



Obr 12: Základní zobrazení mapy

6.4.3 Listenery

Pro interaktivní práci s mapou je důležité používat listenery, což je mechanismus, který jednotlivým prvkům přiřadí události, na které mají reagovat společně s metodou určující jak by daná reakce měla vypadat. Práce s listenery je stejná jako v jiných programovacích jazycích při tvorbě formulářů (Delphi, Java, C++ a další).

```
google.maps.event.addListenerOnce(map, 'tilesloaded',
addMarkers);
```

```
programovací jazyk: JavaScript, umístění: app/webroot/js/map.js
```

Z výše uvedeného kódu je patrné, že listener je připojen k objektu *map* a jeho reakce je vázána na událost načtení tohoto prvku. Tento druh listeneru *addListenerOnce* pracuje tak, že je jednou obsloužena událost, ke které je vázán a následně jeho funkčnost končí. Nebo-li po načtení objektu *map* je vykonána metoda *addMarker()*.

Následující kód ukazuje další dva typy událostí, které jsou v aplikaci obsluhovány. Tyto listenery jsou použity pro práci se značkami (přidávání událostí a zobrazování událostí v závislosti na pozici uživatele).

```
google.maps.event.addListener(newEventMarker, 'mouseup',
function(){ ..
google.maps.event.addListener(newEventMarker, 'click',
function(){ ..
```

```
programovací jazyk: JavaScript, umístění: app/webroot/js/map.js
```

První událost *mouseup* reaguje na stav, kdy uživatel používá možnosti přenášení značky po mapě pomocí myši. Při dokončení přesunu značky uživatel pustí tlačítko myši a v našem případě aplikace reaguje na tuto událost zjištěním adresy na základě pozice a upravením adresy v poli *Adresa*.

Událost *click* má za úkol řešit případ, kdy uživatel chce zobrazit informace ke značce na kterou klikne myší.

6.5. Realizace správy událostí

Správu událostí je nutné rozdělit na několik částí. Jednotlivé části vystihují, jak jsou řešeny podmnožiny problémů při práci s mapou. V první podkapitole je realizována vizualizace na mapě z hlediska práce pouze s GoogleMaps API. Ve druhé podkapitole, je vyřešena problematika přidávání listenerů při práci se značkami, se kterými uživatel interaktivně pracuje ve chvíli přidávání nové události nebo v případě, že se rozhodne vizualizovat události ve svém okruhu a potřebuje zadat svoji stávající pozici na mapě.

Třetí podkapitola řeší problém přidávání událostí do systému jak z pohledu práce s GoogleMaps API, tak z pohledu práce kontroléru *PagesController* při ukládání dat do databáze. Ve čtvrté podkapitole je vyřešeno odebírání událostí přidaných uživatelem, které je už řešeno pouze z pohledu kontroléru *PagesController*. Poslední kapitola má na starost výpis událostí, pomocí helperu CakePHP Paginate, což je opět součást logiky kontroléru *PagesController.*

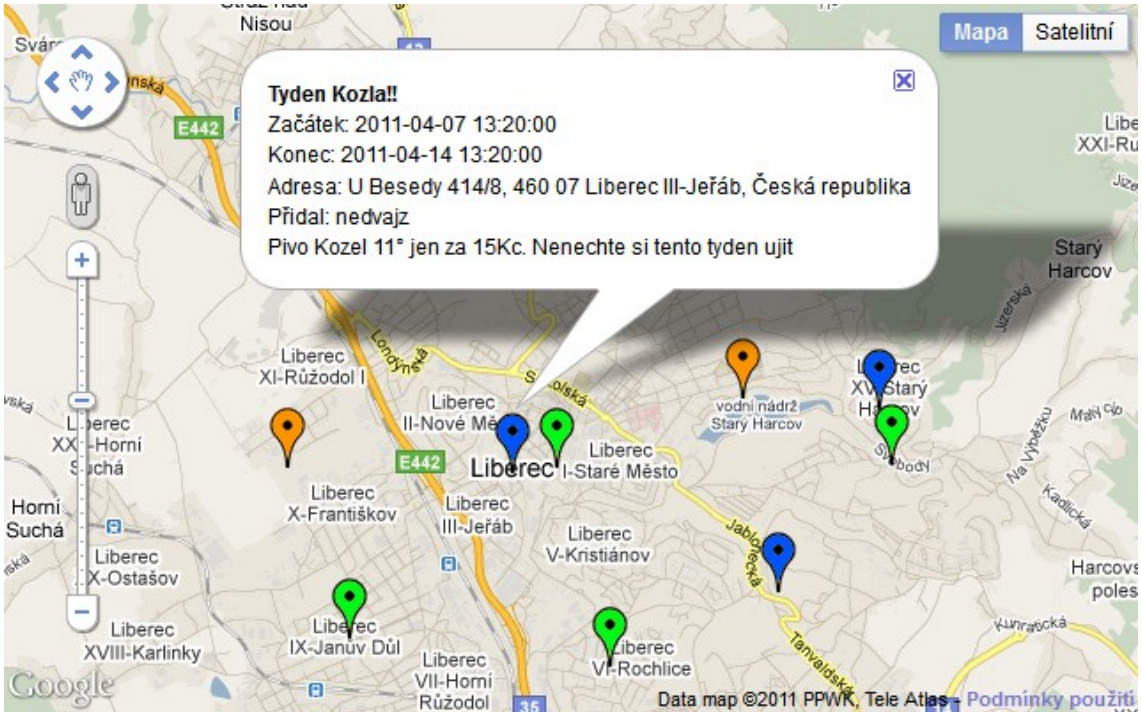
6.5.1 Vizualizace událostí na mapě

Události, které jednotliví uživatelé přidávají na mapu se ukládají do databáze. Ve chvíli kdy uživatel zvolí město, ve kterém chce události s ním svázané zobrazit, je nutné tyto události vizualizovat na mapě města (dané lokality).

Data získaná z databáze jsou předána pro zobrazení. V rámci skriptové části zobrazení je hodnotami událostí naplněno pole. S polem dále pracuje metoda *addMarkers()*, která je volána při načtení objektu *map* viz kap. „6.4.3 Listenery“.

```
function addMarkers() {
    function createMarker(map, position, number) {
        var marker = new google.maps.Marker({
          position: position,
          map: map,
          icon: eventMarkers[events[number].id_type-1]
        });
..
programovací jazyk: JavaScript, umístění: app/webroot/js/map.js
```

Metoda *addMarkers()* v sobě ukrývá další metodu *createMarker()*, která je následně volána ve for-cyklu. Metoda *createMarker()* vytváří jednotlivé objekty značek třídy *google.maps.Marker*. V konstruktoru třídy jsou předávána data pro jednotlivé parametry. Povinnými parametry jsou *position* a *map*. První ze dvou parametrů určuje pozici, na které se daná značka zobrazí a je mu předáván objekt třídy *google.maps.LatLng*.



Obr 13: Zobrazení značek na mapě

Druhým povinným parametrem je určení mapy, ke které bude značka připnuta. To je v našem případě určeno předáním reference na objekt *map*.

Z tohoto parametru také vychází princip, jakým danou značku můžeme zrušit v rámci mapy. Pokud je nutné značku vymazat z mapy, například z důvodu vymazání dané události, ke které příslušela, tak postupujeme viz. níže uvedený kód.

```
marker.setMap(null);
marker = null;
programovací jazyk: JavaScript
```

Poslední z uvedených parametrů je *icon, který* určuje vzhled dané značky na mapě. Tento parametr lze spojit ještě s vlastností *shape*, což pomůže vytvořt realističtější trojrozměrný vzhled značky. Parametr *shape* určuje stín, jaký značka bude vrhat v pozadí. Oba parametry přijímají jako vstupní data cestu, kde se obrázek nachází. U obrázku pro parametr *shape* je nutné, aby byl formátu 24-bit PNG.

```
..
google.maps.event.addListener(marker, 'click', function() {
  geocoder.geocode({'latLng':marker.getPosition()},
   function(results, status){
    if (status == google.maps.GeocoderStatus.OK){
```

```
      var myHtml =
       '<p><strong>'+events[number].title+'</strong><br/>'+
       'Začátek:'+events[number].start_time+'<br/>' +
       'Konec:'+ events[number].end_time +'<br/>' +
       'Adresa:'+ results[0].formatted_address +'<br/>' +
       'Přidal:'+ users[number].username +'<br/>' +
       '<p>' + events[number].text + '</p>';
       infoWindow.setContent(myHtml);
       infoWindow.open(map, marker);
     }else{
       alert(posDoesntExist);
  }});});
  } //konec metody createMarker

    for (var i = 0; i < events.length; i++) {
      var latLng =
       new google.maps.LatLng(events[i].lat, events[i].lng);
      createMarker(map, latLng, i);
    }
  }
programovací jazyk: JavaScript, umístění: app/webroot/js/map.js
```

Při každém přidání značky metodou createMarker(), je značce zároveň zaregistrován listener reagující na kliknutí na značku, to vyvolá okno s informacemi, které přímo souvisí s událostí vizualizovanou v rámci mapy. Ukázka reakce na kliknutí na značku je na obrázku „Obr 13: Zobrazení značek na mapě“. Vzhled vyvolaného informačního okna lze formátovat pomocí jazyka HTML a CSS. Funkce třídy *google.maps.Geocoder* je osvětlena v následující kapitole „ 6.5.3 “ na příhodnějším příkladu použití.

6.5.2 Listenery pro nové události a zobrazení pozice uživatele

Ve chvíli, kdy si zvolíme přidávání nové události (nebo volbu zobrazení dat na základě uživatelovi pozice) formou kliknutí myší na místo na mapě, kde se má příslušný ukazatel zobrazit, musí mít mapa zaregistrovaný listener. Ten reaguje na kliknutí a zavolá příslušné metody, které vytvoří požadované značky.

```
if($('#events-add-link').hasClass('active')){
 google.maps.event.addListenerOnce(map, 'click',
                                   addClickedEvntMarker);
```

```
programovací jazyk: JavaScript+jQuery,
umístění: app/webroot/js/map.js
```

V uvedeném kódu je vidět, že registrace listeneru je závislá na tom, zda používáme možnost přidávání nové události nebo zobrazování události v okolí uživatele. V tomto případě se jedná o přidávání nové události. Aplikace to zjistí tak, že se za pomocí jQuery příkazu ptá, jestli HTML prvek event-add-link, který odpovídá odkazu „Přidej událost“ na „Obr 17: Události v okruhu uživatele“, je aktivní(má nastavenou třídu active). V případě že tomu tak je, je zaregistrován listener, který reaguje na kliknutí na mapu vyvoláním metody *addClickedEvntMarker*(více o metodě viz. kap. 6.5.3). V opačném případě, kdy je aktivní druhý HTML prvek, je listener zaregistrován s metodou *addClickedPosMarker*(viz. kap. 6.6.2).

Pro přidávání listeneru určených přímo pro práci se značkami slouží v aplikaci metody a*ddNewEventMarkerListeners* a *addUserPosMarkerListeners.* Metody tedy mají na starost přidávání listenerů v případech, kdy se uživatel rozhodne přidat událost a pro správné umístění si nejprve vizualizuje pozici přidávané události (obr. 14). Nebo ve chvíli, kdy uživatel použije funkčnost řešící vykreslení událostí, které jsou v nějakém zvoleném rádiusu kolem jeho pozice (obr. 16 a obr.17).

Listenery mají tedy na starost práci s těmito dvěma značkami po přidání na mapu. Jejich kód je podobný a závisí již na dříve uvedených informacích v kapitolách „ 6.4.3 Listenery“ a „ 6.5.1 Vizualizace událostí na mapě“. Jejich funkcionalita spočívá v přidání dvou listenerů pro každou ze dvou typů značek. První listener má na starost změnu adresy ve chvíli, kdy se se značkou pohybuje v rámci mapy pomocí přenášení. Adresa se mění jak v bublině, která je zobrazena právě za pomocí druhého listeneru, který nastaví reakci na kliknutí na značku, tak v pozici pro zadávání adresy. Stejná reakce je implementována u přidávání nové události tak při vyhledávání událostí kolem pozice uživatele.

```
$('#EventAddress').val(results[0].formatted_address);
$('#EventLat').val(newEventMarker.getPosition().lat());
$('#EventLng').val(newEventMarker.getPosition().lng());
```

```
programovací jazyk: JavaScript+jQuery,
umístění: app/webroot/js/map.js
```

Jak se postupuje při zadáváni dat pro zobrazení v bublině, popisuje předchozí kapitola. Adresový řádek je vyplněn pomocí prvního řádku ukázky kódu. Další dva řádky mají na starost vyplnění skrytých položek formuláře, které jsou klíčové pro uložení, zobrazení značky, ale i pro určení událostí v okolí uživatele. Tyto dva řádky vyplňují do formuláře hodnoty zeměpisné šířky a délky.

6.5.3 Přidávání událostí

Jak bylo zmíněno v úvodu kapitoly „6.5 Realizace správy událostí“, tak je přidávání událostí řešeno z dvou pohledu. Prvním pohledem je přidávání události na mapu města, což má na starost skript obsažený v souboru *map.js*.

```
function addClickedEvntMarker(event){
    if(newEventMarker == null){
          newEventMarker = new google.maps.Marker({
                position: event.latLng,
                 map: map,
                 draggable: true,
                 icon: markerYellow
          });
          addNewEventMarkerListeners();
    }

geocoder.geocode({'latLng':newEventMarker.getPosition()},
    function(results, status){
      if (status == google.maps.GeocoderStatus.OK){
       $('#EventAddress').val(results[0].formatted_address);
       $('#EventLat').val(
                newEventMarker.getPosition().lat());
       $('#EventLng').val(
                newEventMarker.getPosition().lng());
      }else{ alert(posDoesntExist); }
    });
programovací jazyk: JavaScript+jQuery,
umístění: app/webroot/js/map.js
```

Metoda *addClickedEvntMarker()*, je volána ve chvíli, kdy je vybrána možnost přidávání značky na mapu města a zároveň uživatel klikne na mapu, kam chce událost přidat. V této chvíli se v případě, že se tak již nestalo předtím, vytvoří značka a této značce jsou přiděleny příslušné listenery. Zaregistrování listenerů ke značce má na starosti metoda *addNewEventMarkerListeners()*. V parametrech pro vytvoření příslušné značky si můžeme všimnout parametru *draggable*. *Draggable* má na starost zpřístupnění možnosti pohybu značkou za pomocí metody drag&drop, což v tomto případě uživatelsky zpříjemňuje práci při přidávání nové události. Uživatel nejdříve pouze klikne na přibližnou pozici, kde se má daná událost konat a tím vytvoří značku na mapě. Pozici značky pak může společně s přiblížením mapy upřesnit jejím jednoduchým přesunutím na výslednou pozici.

Obr 14: Zobrazení přidávané události na mapě

Pro vyhledávání pozice v závislosti na souřadnicích přidané značky slouží objekt *geocode* třídy *google.maps.Geocoder*. Metoda *geocode.geocode()* po předání hledané pozice vrací příznak o úspěšnosti hledání v proměnné *status* a v případě, že byla úspěšná, tak i výsledek(výsledky) v poli *results*. Ve chvíli, kdy metoda vrátí požadovanou adresu(výsledek v poli *results*), je zapsána do řádku pro zadávání adresy. Dále jsou vyplněna skrytá pole formuláře hodnotami příslušné zeměpisné šířky a délky.

```
function showEvntAdressOnMap(aAddress){
  geocoder = new google.maps.Geocoder();
  geocoder.geocode({address:aAddress},
   function(results,status){
    if (status == google.maps.GeocoderStatus.OK){
      if(newEventMarker == null){
        newEventMarker = new google.maps.Marker( .. );
        addNewEventMarkerListeners();
      }
      $('#EventAddress').val(results[0].formatted_address);
      newEventMarker.setPosition(
                  results[0].geometry.location);
    }else{  alert(posDoesntExist); }
  });
}
```

programovací jazyk: JavaScript, umístění: app/webroot/js/map.js

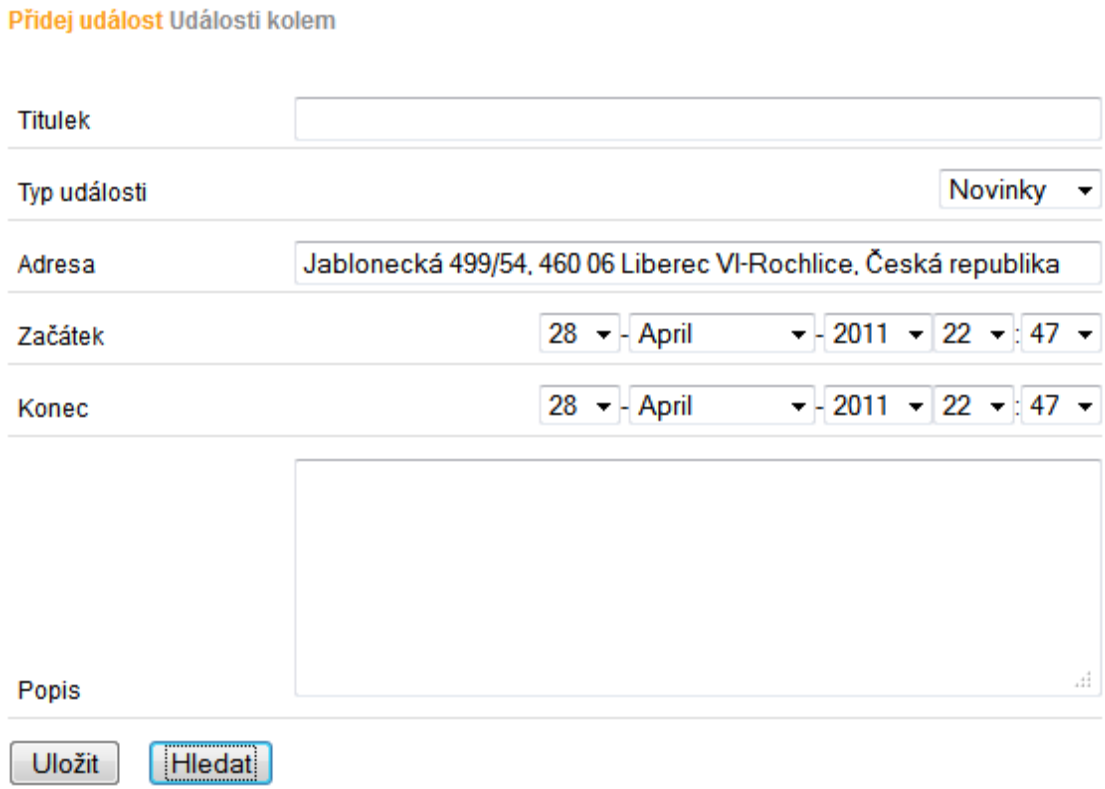
Pokud se uživatel rozhodne přidávat události pomocí vyhledání textově zadané adresy, je ve chvíli požadavku na hledání zadané adresy volána metoda *showEvntAdressOnMap()*. Stejně, jako v předchozím případě, je použita metoda *geocode.geocode()*, která pracuje i inverzně, za pomocí předání zadané adresy, na jejímž základě se pokusí vyhledat skutečnou adresu a výsledek vrací ve stejném formátu, jako v předchozím případě.

V případě, že není značka přidávání události stále vytvořena, metoda jí v tuto chvíli vytvoří a zadá souřadnice nalezené adresy. Následně metoda ještě doplní adresu ve správném formátu opět do řádku pro zadávání adresy a končí.

```
    function addEvent(){
         $this->Event->save($this->data['Event']);
         $this->redirect(array('controller' => 'Pages',
                                  'action' => 'town'));
    }
```

programovací jazyk: PHP,
umístění: app/controllers/pages_controller.php

Jelikož formulář splňuje konvenci zadáváných názvu shodnou s názvy položek pro ukládání do databáze frameworku CakePHP, tak stačí jednoduše předat data formuláře *Event* metodě *$this->Event->save()* a o víc se nemusíme starat. Mechanismus modelu *Event*, zděděny od třídy *AppModel* se postará o veškerou práci při ukládání do databáze. Následně je nutné metodu přesměrovat na action *town()*, která následně vykreslí znovu mapu města s již přidanou událostí.



Obr 15: Formulář pro přidávání události

6.5.4 Odebírání událostí

Při odebírání události je volána metoda *deleteEvent()* v kontroléru PagesController a podobně jako v předchozím případě se jedná o jednoduchou obsluhu modelu *Event* pomocí metody děděné od třídy *AppModel*, kterou je v tomto případě *$this->Post->deleteAll()*. Metoda maže položky v databázi podle parametru *id* události, které je unikátní pro každou událost a je pro určení položky dostatečný. Parametr *id_user* je zde zahrnut z důvodu kontroly, jestli se nesnaží o smazání neoprávněný uživatel, který volá metodu přímo pomocí url adresy a příslušného *id* události.

```
function deleteEvent($id){
 $this->Event->deleteAll(
   array(
     'Event.id' => $id,
     'Event.id_user' => $this->Session->read('Auth.User.id')
   ),
   false);
 $this->redirect(
   array('controller' => 'Pages', 'action' => 'town');
  }
programovací jazyk: PHP,
umístění: app/controllers/pages_controller.php
```

6.5.5 Výpis událostí – Paginator

Výpis událostí do tabulky pod mapou je realizován pomocí helperu *Paginator* knihovny CakePHP. Chování helperu společně s určením dat, která bude vypisovat, je nastaveno v controlleru *PagesController*, který následně posílá data určená k „paginate“(stránkování, stránkovému výpisu) na požadované view, v našem případě se jedná o *town.ctp.* Nastavení helperu *Paginator* je variabilní, ale pro náš případ nás zajímají zejmén čtyři hlavní prvky.

- conditions – určuje podmínky, za kterých budou z tabulky *Event* vybrány položky. V našem případě je v proměnných *$idTown* a *$idEvent* hodnota, odpovídající městu, kde uživatel zvolil výpis a hodnotu typu události, kterou chce uživatel vypsat. Logika zadávání parametrů odpovídá standardním dotazovacím jazykům na databázi (vice viz. [4]).
- limit – určuje počet položek, který bude v jednu chvíli zobrazen ve view
- fields – všechna data, která jsou požadována po databázi, v případě aplikace požadujeme veškerá data z tabulek (*EventType, User, Event*)
- joins – určuje spojení tabulky *Event* s tabulkami *EventType* a *User*

6.6. Zobrazeni událostí na základě pozice uživatele na mapě

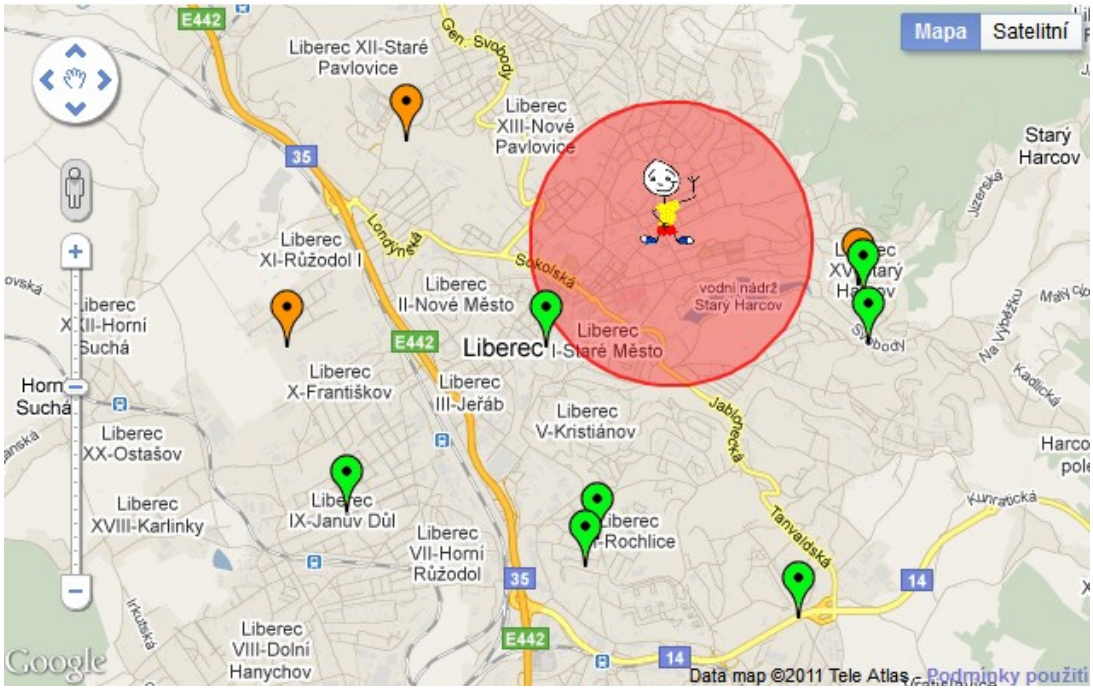
Ve chvíli, kdy uživatel zvolí tento způsob zobrazení požadovaných událostí, zpravidla volí posloupnost dvou kroků. Prvním krokem je zvolení pozice, ze které chce vycházet a vykreslení kruhu, který vyznačuje maximální rádius od uživatele a symbolizuje plochu, která určí jaké události mají být vypsány/vykresleny.

Druhým krokem je samotný požadavek na systém, aby tyto požadavky zobrazil. Tento krok je splněn odesláním formuláře na příslušnou metodu kontroléru.

6.6.1 Zobrazení okruhu pomocí GoogleMaps API

V případě že chceme vyznačit nějakou oblast na mapě, musíme použít k tomu určené nástroje. GoogleMaps API zprostředkovává hned tři varianty pro značení oblastí na mapě.

- Polygon – jakýkoliv polygon (mnohoúhelník). Je definován za pomocí pole vrcholů. Pozice vrcholů je určena jako u všech případů v rámci GoogleMaps API, a to objektem třídy google.maps.LatLng.
- Circle – kružnice kolem bodu určeného pomocí google.maps.LatLng. Dalším důležitým parametrem je rádius kružnice, který se určuje v metrech.
- Rectangle – na rozdíl od předchozích dvou, rectangle nepoužívá pro určení svých rozměrů třídy *google.maps.LatLng*, nýbrž třídy google.maps.LatLngBounds , která definuje jeho hranice.



Obr 16: Zvýraznění oblasti pomocí objektu třídy maps.google.Circle

Pro vyznačení oblasti zvolené uživatelem bylo v úloze použito asi nejlogičtějšího způsobu a tedy objektu třídy *google.maps.Circle*. Toto řešení bylo

zvoleno z důvodu, že kružnice odpovídá požadavku, kdy uživatele zajímá vzdálenost, kterou urazí na všechny strany stejně.

```
function getCircleArea(position, radius, newCircle){
    var defRad = 1000;
    if(radius == null){
          radius = defRad;
    }else{
          radius = parseInt(radius);
    }
    if(isNaN(radius)){
          alert(wrongNumber);
          radius = defRad;
    }
    ..
```

programovací jazyk: JavaScript, umístění: app/webroot/js/map.js

V první části kódu je vidět chování (v případě) špatně zvoleného, nebo nezvoleného rádiusu, kde aplikace automaticky vyplní výchozí hodnotu na 1000m.

```
    ..
    if(markerUserArea == null){
          var areaOptions = {
                strokeColor: "#FF0000",
                 strokeOpacity: 0.8,
                 strokeWeight: 2,
                 fillColor: "#FF0000",
                 fillOpacity: 0.35,
                 map: map,
                 center: position,
                 radius: radius
           };
          markerUserArea = new google.maps.Circle(areaOptions);
     }else{
          markerUserArea.setCenter(position);
          markerUserArea.setRadius(radius);
     }
}
```

```
programovací jazyk: JavaScript, umístění: app/webroot/js/map.js
```

Pokud vykreslujeme oblast kolem ukazatele pozice uživatele poprvé, je nutné oblast vytvořit a nadefinovat ji potřebné vlastnosti.

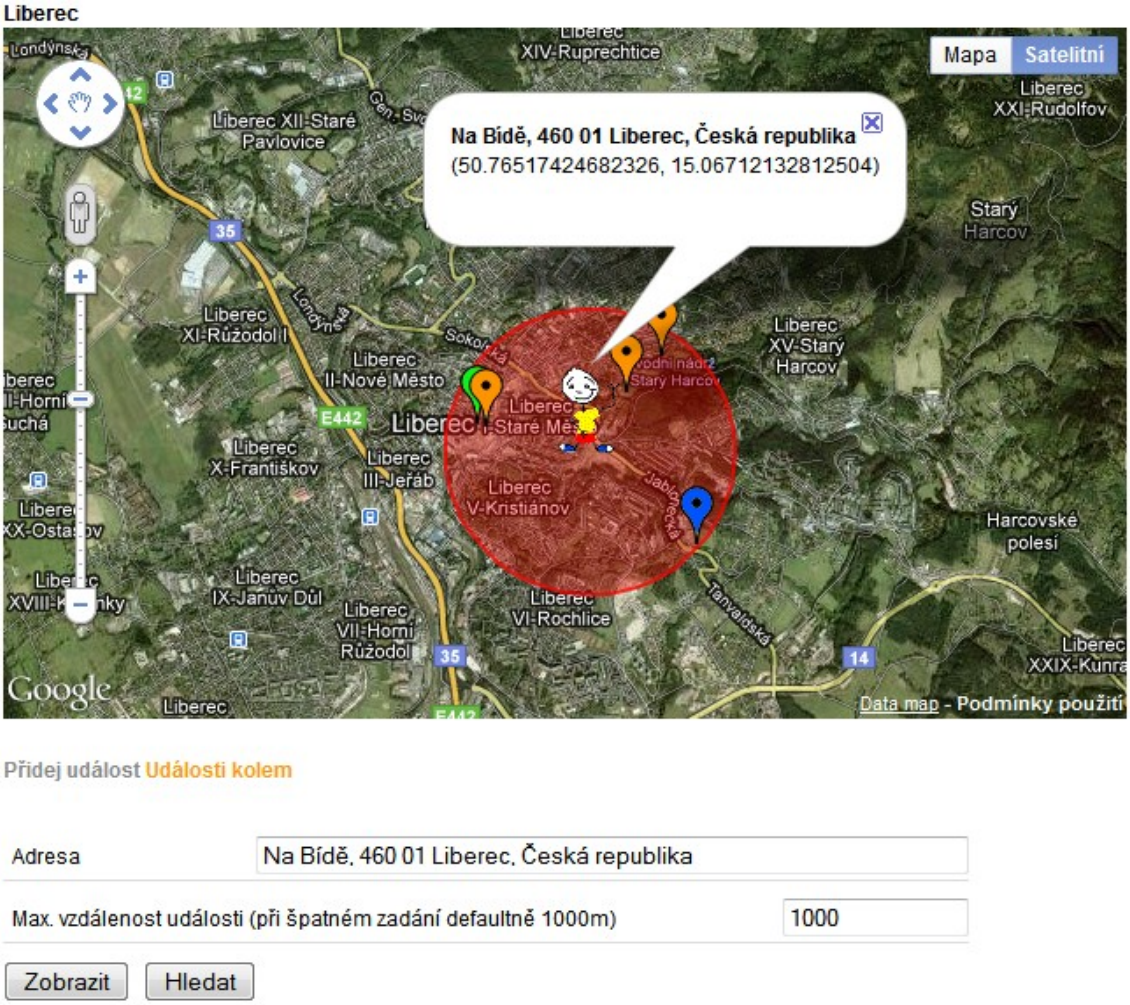
- strokeColor, strokeOpacity, strokeWeight – se týkají hranice kruhu a postupně nastavují jeho barvu, průhlednost a šířku
- fillColor, fillOpacity – určuje podobu výplně kruhu, obdobně jako výše se určuje barva výplně a její průhlednost, logicky pak není třeba určit šířku, protože je po cele ploše výplně
- map, position – zde plní stejnou funkci, jako při definování značky (marker)
- radius – parametr určující radius kružnice v jednotkách metrů.

  Výsledné vyznačení oblasti je znázorněno na „Obr 16: Zvýraznění oblasti pomocí objektu třídy maps.google.Circle“.

6.6.2 Určení událostí v okolí uživatelovi pozice

V první řadě je nutné určit pozici uživatele, což je řešeno podobným způsobem, jako v kapitole „6.5.3 Přidávání událostí“. Je zde použit princip obsluhy mapy vizualizace značky na mapě pomocí kliknutí na požadovanou pozici, což vyvolá metodu *addClickedPosMarker* (obdoba metody *addClickedEvntMarker*). Nebo vyhledáním podle adresy, což obsluhuje metoda *showPosAdressOnMap* (obdoba metody *showEvntAdressOnMap*). Jelikož se příliš neliší od metod řešených viz. uvedena kapitola výše, tak se jejich realizací v této kapitole nebudeme dále zabývat.

Jak bylo popsáno v kapitole „5.4 Vyhledávání událostí na mapě z pozice uživatele“, pro určení událostí je nutné nejdříve přepočíst souřadnicový systém WSG-84 na UTM, což řeší metoda *__convertLntAndLtdToUTM($latOrig, $lngOrig)*. Předanými parametry jsou zeměpisná šířka a délka, následně po provedení přepočtu na Easting a Northing souřadnice systému UTM metoda vrací pole s těmito hodnotami.



Obr 17: Události v okruhu uživatele

Služeb *__convertLntAndLtdToUTM($latOrig, $lngOrig)* využívá metoda *findEventsAround()*, která je volána odesláním formuláře při požadavku na zobrazení události v okolí uživatele.

```
function findEventsAround() {
   $refPositionInUTM =
     $this->__convertLngAndLtdToUTM($this->data['Position']
                  ['lat'], $this->data['Position']['lng']);
     ..
programovací jazyk: PHP,
umístění: app/controllers/pages_controller.php
```

V první řadě je zjištěna referenční pozice, kterou je pozice uživatele. Pozici formulář předává ve standardním formátu GoogleMaps API, tedy jako hodnoty

zeměpisné šířky a délky. Tyto hodnoty je tedy nutné v první řadě převést na UTM a uchovat si je jako proměnnou referenčního bodu.

```
..
$allTownEvents = $this->Event->find('all',
    array('conditions' => array('Event.id_town' =>
          $this->data['Position']['id_town'])));
..
programovací jazyk: PHP,
umístění: app/controllers/pages_controller.php
```

V dalším kroku je nutné vybrat veškerá data, která by mohla odpovídat uživatelem vybranému prostoru. Data která by mohla odpovídat vybranému prostoru jsou zároveň podmnožinou dat vázaných na danou lokalitu, proto jako soubor dat, ve kterých budeme vyhledávat, jsou zvolena veškerá data dané lokality(města).

```
..
$allPassedEvents = array();
foreach($allTownEvents as $townEvent){
    $eventPositionUTM =
       $this->__convertLngAndLtdToUTM($townEvent['Event']
                  ['lat'], $townEvent['Event']['lng']);
    $posDepOnRefPos =
       array('northing' => $refPositionInUTM['northing'] -
                                $eventPositionUTM['northing'],
             'easting' => $refPositionInUTM['easting'] -
                                 $eventPositionUTM['easting']
      );
..
programovací jazyk: PHP,
umístění: app/controllers/pages_controller.php
```

Postupně jsou všechny souřadnice převedeny na systém UTM a následně normovány podle středu (souřadnice událostí jsou odečteny od souřadnic středu)

```
..
$distance = sqrt(pow($posDepOnRefPos['northing'],2) +
                          pow($posDepOnRefPos['easting'],2));
if ($distance < $this->data['Position']['radius']) {
    $usersDataArr = array('UsersEvent' => array(
```

```
            'user_id' => $this->Session->read('Auth.User.id'),
            'event_id' => $townEvent['Event']['id']
        ));
        $this->UsersEvent->save($usersDataArr);
    }
    ..
    programovací jazyk: PHP,
    umístění: app/controllers/pages_controller.php
```

U takto upravených souřadnic již stačí pouze použít Pythagorovu větu a vypočítat tím jejich vzdálenost od středu. Tuto vzdálenost porovnáme s rádiusem, který uživatel zadal a v případě, že prvek odpovídá vymezené oblasti (rádius je větší, než jeho vzdálenost od středu), je událost zapsána tabulky *UsersEvent*. Ve chvíli, kdy jsou připravována v kontroléru *PagesContoller* data pro zobrazení města, tak se kontroluje, jestli v tabulce *UsersEvent* nejsou položky vázané na identifikační číslo uživatele, který je aktuálně připojený do systému. Pokud tomu tak je a takové položky se v tabulce *UsersEvent* nacházejí, kontrolér pošle na zobrazení události, které odpovídají identifikačnímu číslu města, ale zároveň jejich „id“ odpovídá hodnotě *event_id* z tabulky *UsersEvent*.

7. Závěr

Cílem diplomové práce bylo seznámení se s problematikou vývoje mashup aplikací. Vývoj tohoto typu aplikací se přímo vztahuje k použití cizího zdroje informací (viz. 2.2.), pomocí kterého je naplněna podstata termínu „mashup“. Jako cizí zdroj mělo posloužit API GoogleMaps, jehož informační přínos do aplikace měl spočívat v poskytnutí prostředí, jež dovoluje uživateli vizuálně pracovat s mapou.

Výsledkem práce měla být aplikace, která spadá do výše zmíněné kategorie a vhodným způsobem, jako portál lokálních událostí, využívá možností GoogleMaps API. Využitelnost takového portálu spočívá v možnosti uživatelů sdílet informace s ostatními uživateli. Informace navíc měly být vhodným způsobem navázány na danou lokalitu, k čemuž je vhodným nástrojem právě prostředí GoogleMaps API.

Dále bylo nutné zvolit vhodné prostředky, pomocí kterých mela být mashup aplikace, společně s implementací funkcionality GoogleMaps API, realizována.

Výsledkem je webová aplikace, která splňuje veškeré výše zmíněné požadavky. Aplikace je navrhnuta pomocí metodiky Model View Controller, což ji rozděluje na tři části, které vzájemně spolupracují a jejichž logika je řízena z velké části třídou PagesController, která zároveň odpovídá jedné z nich a to části „Controller“. Podle metodiky je aplikace členěna na 5 tříd spadajících do části „Model“, 3 třídy spadající do části „Controller“ a poslední část „View“ tvoří 5 souborů. Návrh aplikace pomocí této metodiky vhodným způsobem implementuje prostředí GoogleMaps API do vrstvy „View“, která je zároveň vrstvou, se kterou pracuje uživatel.

Použití GoogleMaps API zprostředkovává vizualizaci mapy na požadovaných lokalitách, které byly zvoleny jako města České Republiky. Další práce s API, která uživateli dává možnost snadné orientace v rámci svého prostředí, je realizovaná pomocí listenerů registrovaných na jednotlivé prvky mapy, značek, vyhledávání adres podle pozic značek nebo vyhledávání pozic na základě zadávání adres. Veškerý kód, který pracuje s GoogleMaps API je přehledně členěný v souboru map.js a je napsán v jazyce JavaScript s použitím knihovny jQuery.

Možnosti uživatele vycházející z výše uvedených informací přehledně řeší obousměrnou komunikaci s ostatními uživateli. Uživatel má možnost vizualizovat, do tří skupin událostí dělené informace, na mapě podle požadavku na typ, čas začátku, konce nebo jiných parametrů tykajících se jednotlivých událostí. Případně jednoduchým

způsobem, spočívajícím zejména ve využití listeneru mapy, přidávat jednotlivé události kliknutím na požadovanou pozici.

Ačkoliv aplikace splňuje svými parametry požadavky uvedené v zadání v plném rozsahu, lze ji ještě z hlediska uživatelské přívětivosti doplnit několika funkcemi. Jednou z nich by mohla být automatická detekce uživatelovi pozice na základě jeho IP adresy, případně pomocí GPS modulu. Další možností je implementace mechanismu pro zasílání zpráv mezi uživateli nebo možnosti vytváření si uživatelských skupin. Poslední dvě funkcionality by mohly být vyřešeny implementací služeb sociální sítě Facebook.

Seznam ilustrací

Literatura

[1] ©ProgrammableWeb.com, *Top Mashup Tags*, [online], [cit. 21.4.2011], URL:<http://www.programmableweb.com/mashups>

[2] Google, *Google maps API family* [online], [cit. 14.10.2010], URL:<http://code.google.com/intl/cs-CZ/APIs/maps/index.html>

[3] Prof. Dutch S., *Converting UTM to Latitude and Longitude (Or Vice Versa)* [online], [cit. 20.4.2010] URL:< http://www.uwgb.edu/dutchs/usefuldata/utmformulas.htm >

[4] CakePHP, *Cookbook* [online], [cit. 23.4.2011], URL:<http://book.CakePHP.org>

[5] Andrew P., *10 jQuery Plugins for Easier Google Map Installation* [online], [3.2.2010], URL:<http://speckyboy.com/2010/02/03/10-jquery-plugins-for-easier-google-map-installation/>

[6] CakePHP, *Understanding Model-View-Controller* [online], [cit. 23.4.2011], URL:<http://book.CakePHP.org/view/890/Understanding-Model-View-Controller>

[7] Fox P., *Using PHP/MySQL with Google Maps* [online], [cit. 14.10.2010], URL:<http://code.google.com/intl/cs-CZ/APIs/maps/articles/phpsqlajax.html>

[8] doc. PLÍVA Zdeněk, Ing. DRÁBKOVÁ Jindra, *Metodika zpracování diplomových, bakalářských a vědeckých prací na FM TUL*, Technická univerzita v Liberci, Liberec 2009, 38 stran.

[9] Zend, *ZendFramework* [online], [cit. 25.4.2011], URL:<http://framework.zend.com/>

[10] Google, *Code Playground* [online], [cit. 14.10.2010], URL:< http://code.google.com/APIs/ajax/playground/ >

[11] Mayzes S., *Google Maps jQuery Plugin V1.01* [online], [cit. 14.10.2010], URL:< http://googlemaps.mayzes.org/ >

[12] Hall J.., *Simple Explanation of Model View Controller (MVC)* [online], [cit. 6.4.2010], URL:<http://jordanhall.co.uk/programming/simple-explanation-of-model-view-controller-mvc-2206990/>

[13] Vincent R., *Greenwich ano - anebo Greenwich ne : Proč nultý poledník systému GPS leží o 100 m východněji vzhledem k poledníku observatoře Greenwich* , [online], 2009, URL:<http://www.vugtk.cz/nzk/c5-09/vincent.htm>

[14] *Nieger M. A., The Universal Transverse Mercator (UTM) coordinate system*, [online], 2010, URL:<http://therucksack.tripod.com/MiBSAR/LandNav/UTM/UTM.htm>