VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SADA WEBOVÝCH NÁSTROJŮ PRO VIZUALIZACI VÝSLEDKŮ GEOLOKALIZAČNÍHO SYSTÉMU

A SET OF WEB TOOLS FOR VISUALIZATION GEO-LOCATION SYSTEM PERFORMANCE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PAVEL HODOVAL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN BREJCHA**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Hodoval Pavel**
Obor: Informační technologie
Téma: **Sada weboých nástrojů pro vizualizaci výsledků geolokalizačního systému**
A Set of Web Tools for Visualization Geo-Location System Performance
Kategorie: Web

Pokyny:

1. Nastudujte různé způsoby prezentace dat v geografickém prostředí, např. heatmapa, Choroplethova mapa, kartogram, apod.
2. Proveďte rešerši existujících řešení pro vizualizaci geografických dat.
3. Seznamte se s nástroji pro vývoj geografických aplikací ve webovém prostředí (např. LeafLet, OpenStreetMap, Google Map apod.)
4. Zvolte několik vhodných způsobů prezentace výsledků geolokalizačního systému a implementujte je jako sadu webových nástrojů.
5. Demonstrujte funkčnost sady webových nástrojů na vhodném příkladu.
6. Diskutujte funkčnost dosažené implementace a možnosti budoucího vývoje.

Literatura:

- Dodá vedoucí.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Brejcha Jan, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L. S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je vytvořit sadu flexibilních webových nástrojů a webové rozhraní pro vizualizaci výsledků geolokalizačního systému. Práce zahrnuje úvod do problematiky vizualizace dat v kartografii a základů zpracování obrazu. Dále obsahuje rešerši existujících řešení v podobě různých webových aplikací či nástrojů a shrnuje dostupné technologie. Byla navržena a implementována knihovna pro zpracování a vizualizaci geografických dat a její výsledky jsou zobrazovány v podobě webové mapy prezentující geografická a statistická data. Kvalita implementace byla otestována jak uživatelským tak strojovým testem, jejichž výsledky byly z větší části pozitivní.

Abstract

Aim of this thesis is to create a set of web tools and web user interface to present the results of geo-location system. Thesis contains introduction to theory of data visualization in cartography and basic image processing algorithms. Also there's a research of existing solutions and tools as well as available web technologies. As a result, a library for geographic data processing and visualization was implemented and its results are shown as web map presenting geographical and statistical data. Quality of the implementation was verified and tested by user test and automated machine test with mostly positive results.

Klíčová slova

vizualizační nástroj, geografie, geolokalizace, zpracování dat online, online mapa, heat mapa, body na mapě, výšková mapa, mapa sklonů svahů, webová aplikace, php knihovna

Keywords

visualization tool, geography, geo-localization, online data processing, online map, heatmap, map markers, heightmap, slope map, web application, php library

Citace

HODOVAL, Pavel. *Sada webových nástrojů pro vizualizaci výsledků geolokalizačního systému*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Brejcha

Sada webových nástrojů pro vizualizaci výsledků geolokalizačního systému

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Brejchy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Hodoval
22. května 2017

Poděkování

Tímto bych chtěl velice poděkovat vedoucímu mé práce Ing. Janu Brejchovi za poskytnutí odborné pomoci při vypracování této bakalářské práce.

Obsah

1 Úvod **3**
- 1.1 Způsoby prezentace dat v kartografii . . . 3
  - 1.1.1 Kartogram . . . 4
  - 1.1.2 Heat mapa . . . 4
  - 1.1.3 Anamorfovaná mapa . . . 5
  - 1.1.4 Výšková mapa . . . 5
- 1.2 Základy zpracování obrazu . . . 6
  - 1.2.1 Konvoluce . . . 6
  - 1.2.2 Barvy v počítačové grafice . . . 7

2 Existující řešení **9**
- 2.1 Geolokalizační systémy . . . 9
  - 2.1.1 Google Maps . . . 9
  - 2.1.2 Google Earth Engine . . . 9
  - 2.1.3 Bing Maps . . . 10
  - 2.1.4 OpenStreetMap . . . 10
  - 2.1.5 GeoImgr . . . 11
  - 2.1.6 ImageLocationEstimation . . . 11
  - 2.1.7 QGIS . . . 12
- 2.2 Shrnutí kapitoly . . . 12

3 Zvolené technologie a nástroje **14**
- 3.1 Mapové podklady . . . 14
  - 3.1.1 Google Maps, OpenStreetMap . . . 14
  - 3.1.2 QGIS rozšíření QMetaTiles . . . 14
- 3.2 Webové technologie . . . 15
  - 3.2.1 HTML . . . 15
  - 3.2.2 CSS . . . 15
  - 3.2.3 JavaScript . . . 16
  - 3.2.4 jQuery . . . 16
  - 3.2.5 PHP . . . 16
  - 3.2.6 Leaflet.js . . . 17
  - 3.2.7 Heatmap.js . . . 17

4 Návrh a implementace **18**
- 4.1 Návrh řešení . . . 18
  - 4.1.1 Cílové skupiny uživatelů . . . 18
  - 4.1.2 Požadavky na projekt . . . 18

4.2 Knihovna VizGeo . . . 19
4.2.1 Generování bodů na mapě . . . 19
4.2.2 Generování heat mapy . . . 21
4.2.3 Pomocná třída ColorHelper . . . 22
4.2.4 Generování mapy sklonů . . . 23
4.2.5 Dynamický generátor dlaždic . . . 24
4.2.6 Knihovna pro JavaScript . . . 26
4.3 Jednoduchý MVC framework . . . 27
4.4 Webové rozhraní . . . 28
4.4.1 Mapa s dynamickými vrstvami . . . 29
4.4.2 Administrační rozhraní . . . 29

5 Testování **31**
5.1 Uživatelský test . . . 31
5.1.1 Zadání úkolů . . . 31
5.1.2 Provedená zjištění . . . 32
5.1.3 Návrh řešení . . . 33
5.2 Strojový test výkonu . . . 33

6 Závěr **34**
6.1 Dosažené výsledky . . . 34
6.2 Budoucí vývoj . . . 34

Literatura **35**

A Obsah přiloženého CD **36**

B Manuál **37**

Kapitola 1

Úvod

Nástroje pro vizualizaci geografických dat se dnes těší velké oblibě, ať už v podobě online mapových služeb jako Google Maps, různých navigačních systémů (Waze, Sygic, Garmin aj.), turistických map od Mapy.cz, geocachingu nebo při prezentaci statistických dat, která využívají zejména aplikace na předpověď počasí, třeba český Meteor od ČHMI.

Existují geolokalizační systémy, které pomocí algoritmů počítačového vidění dokáží z fotografie identifikovat třeba nadmořskou výšku [1], přibližnou polohu místa, kde byla fotografie pořízena [10][4], pod jakým úhlem kamery byla pořízena a podobně [6].

Výsledky geolokalizačního systému je potřeba nějakým způsobem prezentovat, analyzovat a dále zpracovávat. Protože však neexistuje žádné řešení, které by poskytovalo všechny potřebné funkce a nebo není dostatečně přizpůsobitelné, tato práce má za cíl vytvořit takové webové nástroje, které budou dostatečně flexibilní a umožní vizualizovat libovolné výsledky jakéhokoliv geolokalizačního systému, za předpokladu dodržení jistého standardu vstupních dat.

V této kapitole se zaměřím na teoretický úvod do problematiky kartografie, geolokalizace a možnosti vizualizace geografických dat a výsledků geolokalizačního systému, nechybí ani úvod do zpracování obrazu. V kapitole 2 shrnu rešerši existujících nástrojů a systémů pro vizualizaci geografických dat a zhodnotím jejich přínos a nedostatky. Kapitola 3 je určena informacím o vizualizačních, mapových a serverových technologiích pro webové aplikace. V kapitole 4 popisuji konkrétní postupy a metodiky, které jsem využil k návrhu a implementaci konkrétních aplikací a nástrojů. Rozbor a výsledky mnou provedených testů hotového řešení jsou obsahem kapitoly 5. V kapitole 6 diskutuji dosažené výsledky podpořené uživatelským experimentem, přínos mé práce a možnosti budoucího rozvoje.

Pro pochopení problematiky, které se tato práce věnuje a kterou do značné míry využívá, je důležité, abychom nastínili teoretický základ, ze kterého pak budeme vycházet a na který se budeme v průběhu práce neustále odkazovat.

Probereme důležité vizualizační metody, na kterých tato práce stojí, podíváme se na různé barevné modely v počítačové grafice a v neposlední řadě prozkoumáme základní metody a algoritmy zpracování obrazu, na kterých práce staví některé výpočetní úkony.

1.1 Způsoby prezentace dat v kartografii

Existuje mnoho způsobů, jak v kartografii vizualizovat statistická data, například kartogram, teplotní mapa, anamorfovaná mapa či mapa výšková [12]. Vzhledem k tomu, že práce se zabývá vizualizací geografických dat, si tyto metody představíme a uvedeme jejich vyu-

žití – především pak teplotní mapu, kterou používám k vizualizaci četnosti nějaké veličiny na mapě a výškové mapy, na kterých provádím různé výpočty a následné zobrazuji jejich výstup.

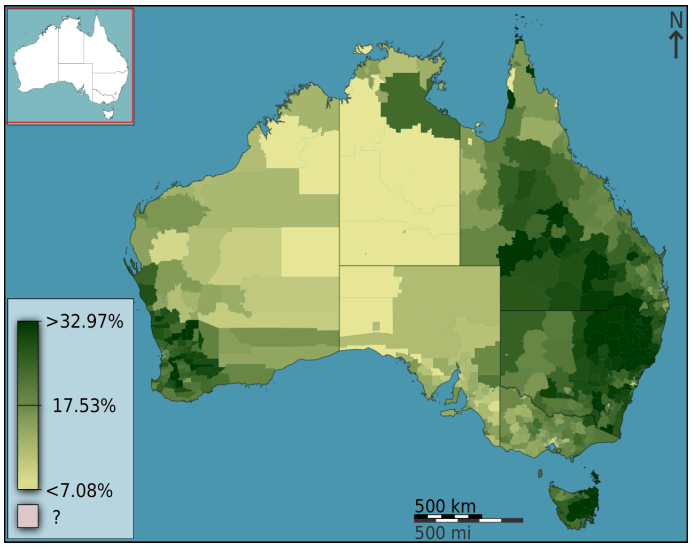
1.1.1 Kartogram

Kartogram je typ mapy, kde jsou jednotlivé oblasti odlišeny různým odstínem/zabarvením v závislosti na poměr k určité statistické veličině, jak znázorňuje obrázek 1.1. V angličtině se označuje jako *choropleth map*.

Kartogramy dělíme na 2 základní druhy – **pravé**, kde se veličina vztahuje na plochu území (například hustota zalidnění) a **nepravé**, u těch se veličina nevztahuje na plochu daného území, kupříkladu počet obyvatel.

Je nutné si uvědomit, že český výraz kartogram není ekvivalentem anglického *cartogram*. Cartogram se v češtině označuje jako anamorfovaná mapa, které se budu věnovat v kapitole 1.1.3.

Kartogram tedy nabízí snadný způsob, jak graficky znázornit rozdíly v konkrétní oblasti a zobrazit jejich poměr.

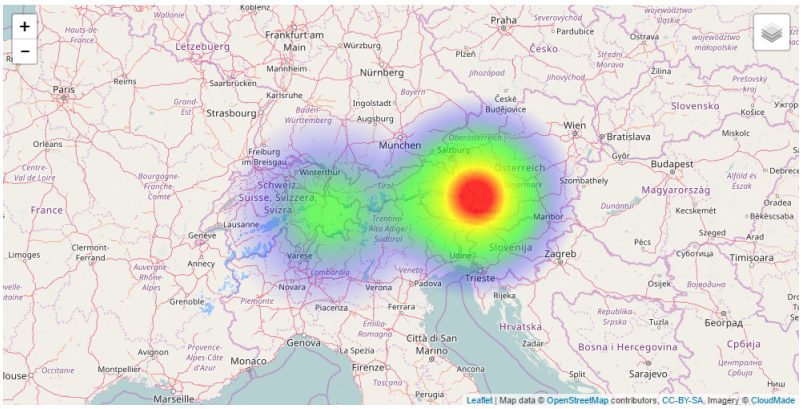


Obrázek 1.1: Ukázka kartogramu znázorňující, kolik Australanů se považuje za Angličany, zdroj: Wikipedie

1.1.2 Heat mapa

Heat mapa (česky teplotní mapa), je znázornění dat většinou uspořádaných v matici, kde je každá hodnota reprezentována barvou z barevného spektra – to je často tvořeno barvami rozkladu bílého světla nebo stupni šedi, možno je však využít jakoukoliv barevnou škálu.

Existuje více druhů heat map, například webové heat mapy, které znázorňují, jaké části webové stránky uživatelé nejčastěji prohlížejí, nebo biologické heat mapy, které se nejčastěji používají v molekulární biologii, kde heat mapa zobrazuje například různá stádia buněk nebo srovnání vzorků od různých pacientů.

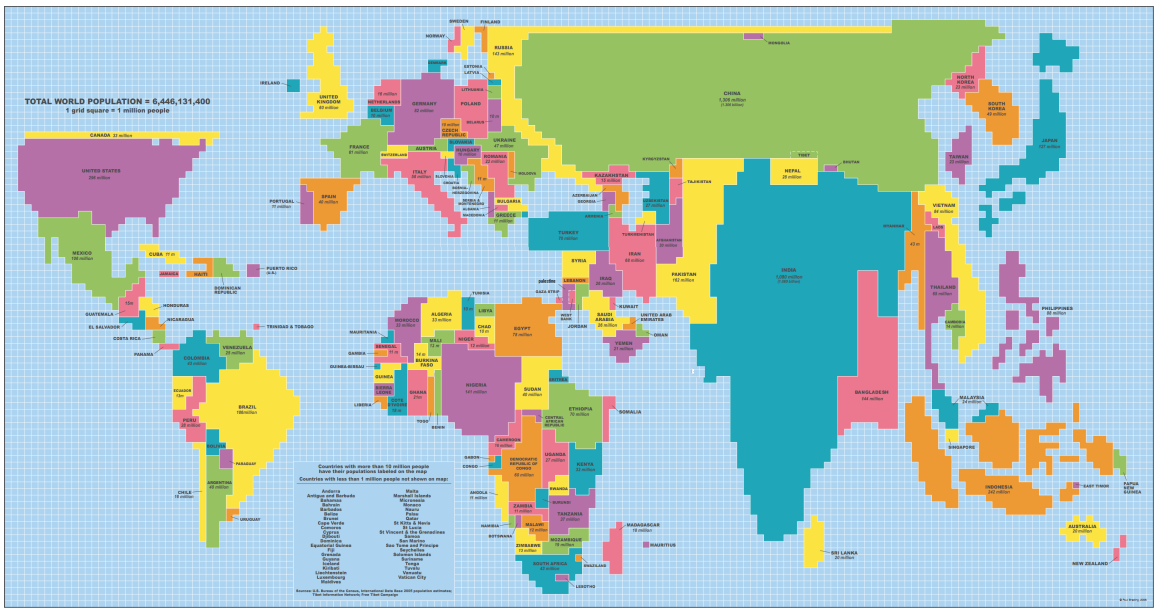


Obrázek 1.2: Ukázka heat mapy

1.1.3 Anamorfovaná mapa

Anamorfovaná mapa (*cartogram*) je typ mapy v kartografii, kde dochází k přeměně rozložení mapy za účelem dosažení lepší čitelnosti, možnosti na mapu vměstnat více zobrazovaných dat nebo bylo zlepšeno celkové vizuální působení mapy.

Anamorfované mapy dělíme zpravidla na 2 základní typy: **plošné** – transformace proběhla rovnoměrně na celém povrchu mapy a **radiální**, kde transformace vychází z konkrétního centra.



Obrázek 1.3: Anamorfovaná mapa světa transformovaná podle počtu obyvatel jednotlivých států

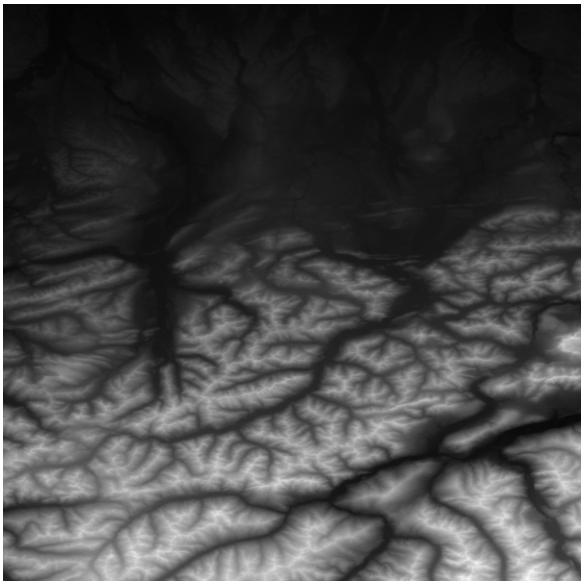
1.1.4 Výšková mapa

Výšková mapa (anglicky *heightmap*) je v počítačové grafice rastrový obrázek, který reprezentuje výšková data určité oblasti, kde každý pixel představuje výšku od základny terénu

(např. od 0 m n. m.). Je situována v stupních šedi, přičemž černá představuje nejmenší vzdálenost od základny a bílá největší vzdálenost.

Výšková mapa může být uložena v libovolném obrazovém formátu (např. PNG), existují ale také speciální formáty. jež umožňují spolu s obrázkem uchovávat i další metadata, například formát GeoTIFF, který uchovává také veškeré údaje o zeměpisných souřadnicích daných částí mapy a usnadňuje tak práci s různými mapovými nástroji – např. těmi na vytváření mapových podkladů, importování do GIS aplikací apod. Je důležité si uvědomit, že při použití formátů jako je PNG, bitová hloubka obrázku určuje, kolik hodnot je možno reprezentovat pomocí barvy – v případě 8bitového PNG (8 bitů na barevný kanál) lze reprezentovat pouze 256 barev, tzn. 256 různých výšek. V takovém případě může především v oblastech s velkými výškovými rozdíly dojít k nepřesnostem.

Výškové mapy mají v kartografii a počítačové grafice mnoho uplatnění, ať už jako vstupní data pro různé výpočetní algoritmy, reprezentaci dat na mapě, nebo především pro generování 3D terénu [2], které je velmi rozšířené v segmentu vývoje počítačových her.



Obrázek 1.4: Ukázka výškové mapy části Alp, zdroj: USGS

1.2 Základy zpracování obrazu

V mé práci využívám algoritmy pro převod mezi různými barevnými prostory jako RGB a HSV a také techniky zpracování obrazu – zejména konvoluci pro aproximaci derivace obrazu, díky které spočítám sklony svahů terénu, ty pak pomocí operací s barvami zakóduji jako pixely na výsledné mapě. Je proto nezbytné, abychom tyto barevné prostory a techniky zpracování představili a vysvětili, jakým způsobem fungují.

1.2.1 Konvoluce

Z matematického hlediska je konvoluce operátor zpracovávající dvě funkce. V počítačové grafice se využívá diskrétní konvoluce a je definována následujícím vzorcem:

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x - i, y - j) \cdot h(i, j) \tag{1.1}$$

Funkce $h(i, j)$ představuje tzv. konvoluční jádro nebo také konvoluční masku, a v případě zpracování dvourozměrného diskrétního obrazu se jedná o matici, kterou posouváme po obraze a aplikujeme na všechny jeho pixely kromě těch krajních, protože pro ně nedokážeme určit hodnoty celého okolí. Hodnoty sousedních pixelů vynásobíme s jednotlivými prvky matice (určené konvolučním jádrem) a sečteme je dohromady, tím získáme novou hodnotu pixelu pro danou souřadnici [8].

Příklad: Pro demonstraci si spočítáme jednoduchou konvoluci. Mějme níže uvedený výřez z obrázku (1. matice) a konvoluční jádro $[-1, 1]$ (2. matice), dosazením do vzorce pak dostaneme následující:

$$(f * h)(x, y) = \begin{pmatrix} 3 & 5 & 1 \\ 14 & 8 & 21 \\ 7 & 0 & 35 \end{pmatrix} * \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

Středový prvek matice, v tomto případě číslo 8, bude pixel, pro který chceme provést konvoluci. Matici obrázku tedy překryjeme konvoluční maskou a spočítáme sumu:

$$3 \cdot -1 + 14 \cdot -1 + 7 \cdot -1 + 1 + 21 + 35 = 33$$

Výsledkem této konvoluce je tedy hodnota 33, což bude nová hodnota prostředního pixelu. Metodu následně opakujeme pro zbytek nekrajových pixelů a postupně všechny spočítáme.

Konvoluce je základem mnoha operací s diskrétním obrazem, např. odstranění šumu, detekce hran aj. O konkrétním využití konvoluce v mé práci se budu zmiňovat v kapitole 4.2.4.

1.2.2 Barvy v počítačové grafice

Barvy jsou základním stavebním kamenem pro vytvoření obrazu. Na pohled to sice není zcela zjevné, ale všechny barevné prostory fungují na principu skládání více barev z barevného spektra dohromady. Skládání (míchání) barev dělíme na aditivní a subtraktivní.

U aditivního míchání se jednotlivé složky barev sčítají, a tím vzniká světlo vyšší intenzity. Základní barevné složky jsou červená, modrá a zelená. Smícháním všech tří barev vzniká bílá barva.

Pro subtraktivní míchání platí, že přidáváním barevných složek se naopak světlo zeslabuje a každá přidaná vrstva je pohlcuje. Základní barvy tvoří azurová, purpurová a žlutá. Smícháním všech těchto barev vzniká černá. Na tomto principu je založen například barevný model CMYK (cyan, magenta, yellow, key), který se hojně využívá kupříkladu v tiskárnách.

V mé práci využívám hned několik barevných prostorů a převody mezi nimi, proto nyní představíme některé z nich.

RGB

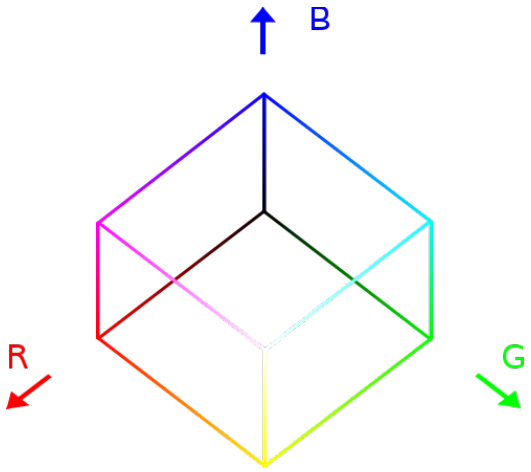
RGB (R – red, G – green, B – blue) je nejzákladnější a nejrozšířenější barevný model, jež využívá většina dnešních zobrazovacích zařízen (monitory, displeje mobilních zařízení atp.). RGB využívá aditivního míchání a výsledkem je tedy složení červené, zelené a modré

barvy – ty jsou reprezentovány tříslożkovým vektorem, kde každá hodnota nabývá hodnot z intervalu $\langle 0; 1 \rangle$. Nejčastěji bývá reprezentován třemi osmibitovými hodnotami $\langle 0; 255 \rangle$, kde čím vyšší hodnota, tím vyšší intenzita dané barevné složky. Maximální počet zobrazitelných barevných odstínů je tedy $256^3 = 16777216$. Obrázek 1.5 znázorňuje barevný prostor RGB pomocí krychle.

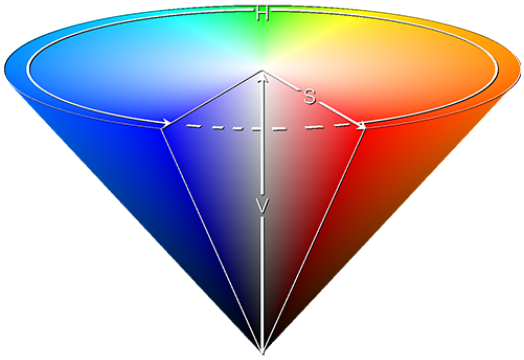
HSV

U modelu HSV na rozdíl od většiny ostatních barevných prostorů složky nereprezentují barvy, ale spíše barevné vlastnosti, a to následujícím způsobem: H (hue) je barevný odstín a určuje převládající barvu, S (saturation neboli sytost), určuje příměs ostatních barev a V (value či jas), udává množství bílého světla.

Barevný prostor se znázorňuje pomocí šestibokého jehlanu, jak je vidno na obrázku 1.6.



Obrázek 1.5: Krychle znázorňující RGB prostor, zdroj: Wikipedie



Obrázek 1.6: Zobrazení barevného prostoru HSV, zdroj: Wikipedie

Paleta JET

Jedná se o paletu implicitně dodávanou s výpočetní aplikací MATLAB. Stala se nepsaným standardem pro vizualizaci nejrůznějších grafových a statistických dat. Paleta funguje na principu převodu (mapování) vstupní hodnoty na určenou barvu.

Kapitola 2

Existující řešení

Tuto kapitolu věnuji studiu existujících geolokalizačních nástrojů a srovnám jejich funkcionalitu. Podíváme se na známé mapové služby jako Google Maps, Bing Maps nebo OpenStreetMap, ale také na nové služby, které zatím nejsou dostupné pro širokou veřejnost – např. Google Earth Engine.

Prozkoumáme ale i méně známé aplikace nebo technologie, například GeoImgr či ImageLocationEstimation. Na závěr (vizte 2.2) shrnu rozdíly či nedostatky jednotlivých řešení a porovnám je s požadavky kladenými na mou práci.

2.1 Geolokalizační systémy

2.1.1 Google Maps

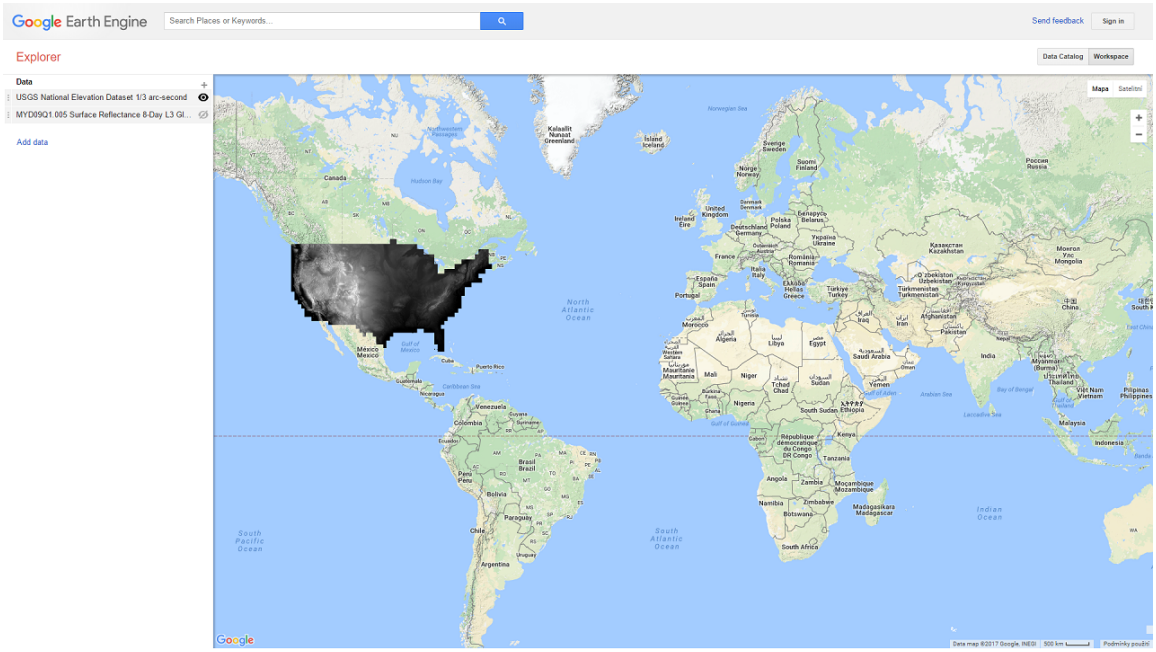
Pravděpodobně nejznámější a nejpopulárnější mapová aplikace, která je kromě webové verze dostupná také na nespočtu platforem, například Google Android, Apple iOS, atp. [1]Google Maps nabízí širokou škálu funkcí – hledání tras (v podobě mobilní aplikace pak i pěší/auto navigaci, turistickou navigaci apod.), zobrazování fotografií z vybraného místa na mapě (uživatelé mohou tyto fotografie přidávat, jejich poloha je určena podle GPS v mobilním zařízení), informace o restauracích a dalších podnicích v okolí (u některých nabízí dokonce virtuální prohlídku) a v neposlední řadě funkci Street View – na dostupných místech světa je možné virtuálně procházet nejrůznější města a lokality tak, jak byla nafocena speciálními vozy Google vybavenými 360° kamerou. Google také nabízí nástroje pro vývojáře, takže je možno jejich mapové podklady využít ve vlastních webových aplikacích, vizte 3.1.1.

2.1.2 Google Earth Engine

[2]Earth Engine je mladá webová služba společnosti Google, která si dává za cíl poskytnout pohodlné prostředí pro vědce a analytiky. Nabízí možnost nad klasickými mapovými podklady z Google Maps zobrazovat a analyzovat různá data. Data jsou dostupná ve formě tzv. Datasets Catalog, kde si uživatel zvolí, jaká data chce na mapu přidat a s těmito daty potom pracuje jako s jednotlivými vrstvami, které je možné skrývat. Některé vrstvy mají také interaktivní funkce, jako například možnost přetáčení v satelitních snímcích podle data pořízení a porovnávat tak například změny na zemském povrchu.

[1] https://maps.google.com/
[2] https://earthengine.google.com/

Součástí této služby je také rozhraní pro vývojáře, kde je možné načíst libovolný mapový podklad a různé datové vrstvy a dále s nimi pracovat, vše je tvořeno v jazyce JavaScript a využívá se také Google Maps API. Řešení je nicméně prozatím velmi uzavřené s nemožností snadného exportu dat na vlastní stránky. Služba je zatím dostupná pouze pro zájemce, je nutné si tedy zaslat žádost o přístup.



Obrázek 2.1: Ukázka Earth Engine Explorer a různých datových vrstev

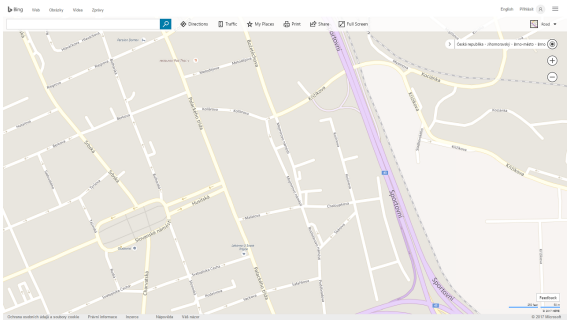
2.1.3 Bing Maps

[3]Bing Maps je mapový nástroj Microsoftu a kromě webové aplikace je také dostupný na Microsoft Store pro zařízení s Windows a Windows Phone. Stejně jako Google Maps nabízí možnost vyhledání trasy pro auto, městskou hromadnou dopravu nebo cestování pěšky. Bing Maps také umožňuje zobrazení aktuální dopravní situace na mapě v podobě barevného znázornění aktuálního vytížení jednotlivých tras, zobrazení záběrů z dopravních kamer a varování na právě probíhající práce na silnicích.

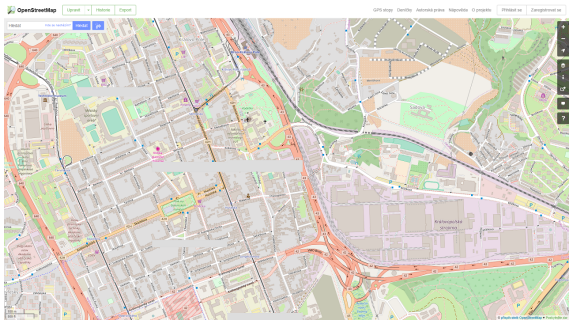
2.1.4 OpenStreetMap

[4]OpenSteetMap je webový nástroj prezentující stejnojmenné mapové podklady, které jsou, jak z názvu vypovídá, dostupné zdarma pod otevřenou licencí a je možné jejich technologii libovolně využívat. Na rozdíl od Google Maps nenabízí takovou nabídku funkcí, jedná se o službu spíše zaměřenou na vývojáře (nejen) webových aplikací. Služba nicméně nabízí hledání trasy mezi dvěma body.

[3] https://www.bing.com/maps
[4] https://www.openstreetmap.org/



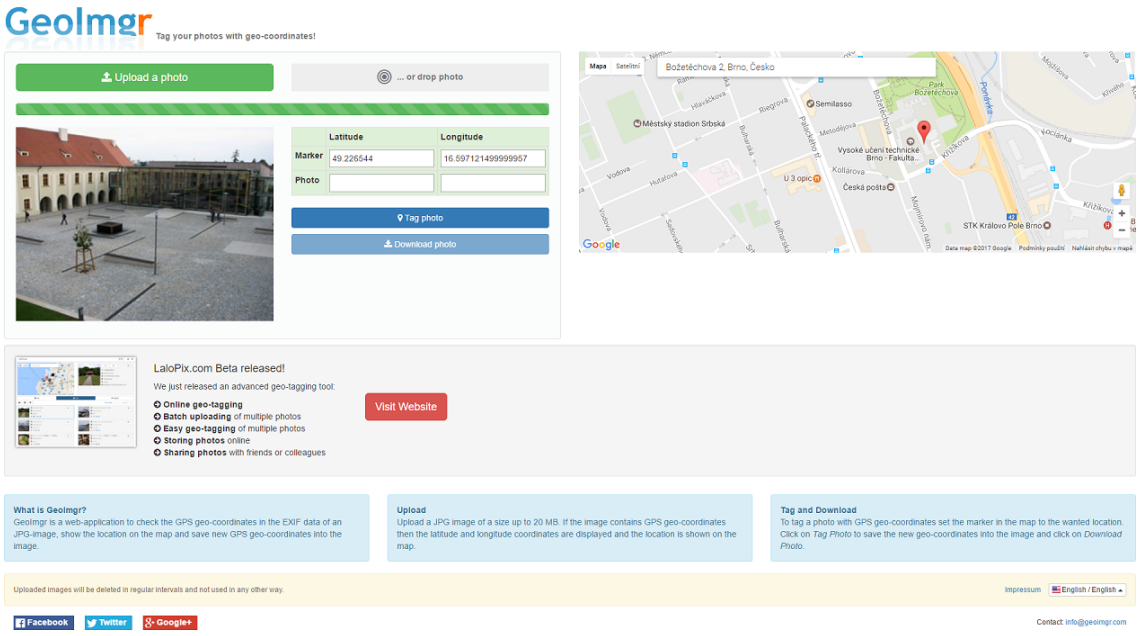
Obrázek 2.2: Ukázka Bing Maps



Obrázek 2.3: Ukázka OpenStreetMap

2.1.5 GeoImgr

[5]GeoImgr je jednoduchý webový nástroj, který z EXIF metadat nahraného obrázku dokáže číst GPS souřadnice v případě, že byly do obrázku zapsány, a zobrazí danou polohu na mapě. Pro vizualizaci mapových dat používá technologii Google Maps, kterou se budu zabývat v kapitole 3. Tato aplikace rovněž umožňuje do obrázku, který geografická data nemá, tato data zapsat snadným výběrem polohy na mapě.



Obrázek 2.4: Ukázka rozhraní GeoImgr

2.1.6 ImageLocationEstimation

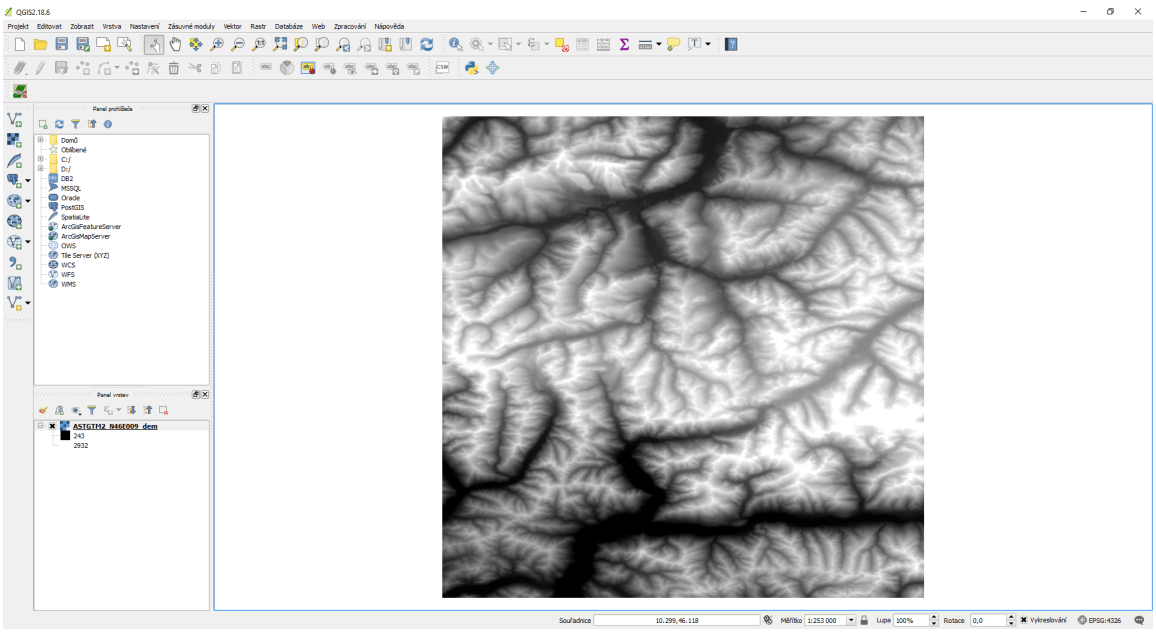
[6]ImageLocationEstimation je projekt, který má za cíl detekovat polohu obrázku pomocí doprovodných textových informací – konkrétněji podle příspěvků autora fotografií na sociální síti Twitter. Pokud v textu příspěvku najde název určitého místa, vyhledá jej na

[5] http://www.geoimgr.com/en/tool
[6] https://github.com/chauff/ImageLocationEstimation

mapě a pokusí se tak určit přesnější geografické souřadnice pořízeného snímku. Zdrojové kódy jsou volně dostupné na GitHub repozitáři, projekt však není provozován jako funkční služba.

2.1.7 QGIS

[7]QGIS (Quantum Geoographic Information System) je multiplatformní desktopová aplikace pro práci s mapovými podklady, umožňuje spravovat a analyzovat různá geografická data, promítat je na mapě a vyrobené mapy poté exportovat do různých formátů. QGIS nabízí aplikaci QGIS Server, která umožňuje dosažené výsledky (vytvořené mapy) prezentovat jako webovou aplikaci.



Obrázek 2.5: Rozhraní aplikace QGIS

2.2 Shrnutí kapitoly

Předvedená řešení (shrnuje tabulka 2.1), aplikace a služby jistě nabízí spoustu zajímavých funkcí a mají své právoplatné využití – GeoImgr je jednoduchá ale užitečná služba, nicméně nabízí pouze jednu základní funkci a to přidávat fotky na mapu či ukládat geografické informace zpět do obrázků, ImageLocationEstimation zase není veřejně dostupná/zprovozněná služba a jedná se spíše o koncept, než o reálnou konkurenceschopnou službu.

Google Earth Engine se ale jeví jako nadějný projekt, který nabízí možnost přidávat různé vrstvy na mapu a tím usnadnit vědcům a analytikům s výsledkem dále pracovat, nicméně ani toto není dostatečně otevřené a komplexní řešení a postrádá některé možnosti a funkce, které jsou pro tuto práci klíčové.

[7] http://qgis.org/en/site/

funkce	**Google Maps**	**Google Earth Engine**	**Bing Maps**	**OpenStreetMap**	**GeoImgr**	**ImageLocationEstimation**	**QGIS**
webové rozhraní	ano	ano	ano	ano	ano	ne	ne
integrace do vlastní stránky	ano	ne	ano	ano	ne	ne	ne
zobrazování bodů na mapě	ano	ano	ano	ano	ano	ne	ano
zobrazování teplotních map	ano	ano	ano	ne	ne	ne	ano
zobrazování výškových map	ne	ano	ne	ne	ne	ne	ano
operace nad výškovými mapami	ne	ano	ne	ne	ne	ne	ne

Tabulka 2.1: Shrnutí existujících řešení

Kapitola 3

Zvolené technologie a nástroje

3.1 Mapové podklady

3.1.1 Google Maps, OpenStreetMap

Jako základní podkladové mapy jsem zvolil Google Maps a OpenStreetMap, o kterých jsem se zmiňoval dříve, poskytují dostatečně přesné a přehledné zobrazení mapových dat a slouží jako základní stavební kámen pro další vizualizační metody, které jsem nad mapou v rámci této práce využil.

3.1.2 QGIS rozšíření QMetaTiles

Jelikož aplikace QGIS nabízí spoustu zásuvných modulů, rozhodl jsem se využít rozšíření `QMetaTiles`, které dokáže vytvořit z QGIS projektu mapové dlaždice (*tiles*). Jako vstupní data lze zvolit celé obrazové plátno projektu, jeho určitou část nebo konkrétní vrstvu. Je možné upravit několik nastavení pro generování – například velikost jednotlivých dlaždic, formát, ve kterém se budou ukládat, kvalitu obrázků a výstupní strukturu, výsledek je možné uložit např. do ZIP archivu, nebo do složky.

Nás zajímá především uložení do složky a možnost uložit dlaždice jako [1]TMS strukturu pro webový server. Nahrál jsem do QGIS některé výškové mapy jako rastrové vrstvy, a pomocí QMetaTiles jsem z těchto vrstev nechal vygenerovat mapové dlaždice pro webové zobrazovače map.

Jediná nevýhoda tohoto pluginu spočívá ve výstupním formátu – dokáže ukládat PNG pouze s 8 bity na barevný kanál, čímž vzniká riziko následných nepřesností ve výškové mapě, které jsem popisoval v kapitole 1.1.4. Prozkoumal jsem i jiná řešení, například skript `gdal2tiles.py`, jež je součástí knihovny GDAL, ale ten osekával výstupní hodnoty pro generování výsledných dlaždic a jednotlivé dlaždice vůbec vizuálně neodpovídaly originální výškové mapě. Otestoval jsem také mnoho dalších nástrojů, např. MapTiler, TileMill aj., ale nenašel jsem jediný dostupný nástroj, který by dokázal vytvářet mapové dlaždice a ukládat je v PNG formátu s 16bitovou reprezentací barevné složky. Řešení, které jsem navrhl, nicméně s vyšším počtem barev počítá a je tedy možné mu dodat vstupní data s vyšší barevnou hloubkou.

[1]TMS (Tile Map Service) je způsob uspořádání a servírování mapových dat, většinou se jedná o požadavky ve tvaru `url_serveru/tiles/zoom/souradnice-x/souradnice-y.pripona`

3.2 Webové technologie

3.2.1 HTML

HTML (HyperText Markup Language) je standardní značkovací jazyk využívaný pro vytváření webových stránek/aplikací a formátování jejich obsahu a určení celkové obsahové struktury. Jazyk je tvořen množinou značek (*tagy*), které jsou uvozeny ostrými závorkami `<` a `>` a určují sémantický význam obsahu, který se mezi značkami nachází. Značky se dělí na párové a nepárové, přičemž párové jsou uvozeny otevírací a uzavírací značkou v následujícím tvaru: `<značka>obsah elementu</značka>` a nepárové obsahují pouze jednu značku: `<značka>` [7].

Značky také mohou obsahovat libovolný počet atributů, jenž určují další vlastnosti pro daný element, zapisují se následovně:

```
<znacka atribut1="hodnota atributu" atribut2="dalsi hodnota">
    obsah elementu
</znacka>
```

Atributy mohou být různého charakteru – přes identifikátor elementu (`id`), jeho třídu (`class`) až po pomocné atributy jako `title`, který určuje vyskakovací titulek po najetí myši na daný HTML element.

Využil jsem nejnovější verzi HTML5, která nabízí některé nové funkce a v dnešní době se těší velké podpoře všech prohlížečů.

3.2.2 CSS

CSS (anglicky Cascading Style Sheets) je standardní jazyk pro popisování vizuálních vlastností (X)HTML prvků na stránce nebo XML dokumentu a používá se pro vytvoření grafické podoby stránky a rozmístění jednotlivých prvků na ní [5].

Syntaxe kaskádových stylů sestává z množiny pravidel, každé pravidlo tvoří tzv. selektor a deklarační blok. Ukážeme si, jak taková syntace vypsadá na následujícím příkladu:

```
selektor {
    deklaracni blok
}
```

Selektor slouží pro určení toho, jaký element bude dané pravidlo ovlivňovat. Je možné využít více elementů naráz, čímž určíme bližší lokaci daného elementu – především jeho zanoření. Například zápis `div p` bude definovat vlastnosti pro všechny odstavce (tag `<p>`), jež jsou potomky elementu `<div>` v libovolné hloubce zanoření. Existují také speciální operátory mezi jednotlivými elementy, které tyto vztahy upřesňují, např. pokud k předešlému příkladu přidáme mezi elementy operátor `>`, tedy: `div>p`, znamená to, že se odkazujeme pouze na přímé potomky `<p>` v elementech `<div>`.

Pro jednodušší správu a rozšiřování stylopisů jsem použil technologii SASS (Sassy CSS), která nabízí značně rozšířenou funkčnost oproti základnímu CSS – například vkládání dalších souborů (inkluze), definování proměnných, funkce pro změnu sytosti barev, vnořené zápisy nebo tzv. mixiny, což jsou v podstatě ekvivalenty funkcí z většiny programovacích jazyků. Hlavní zdrojový soubor SASS je pak nutné přeložit do čistého CSS a výsledkem je jeden jediný CSS soubor, který lze rovnou uložit v komprimovaném tvaru.

3.2.3 JavaScript

Jedná se o interpretovaný jazyk, který umožňuje provádět na stránce interaktivní operace a změny be nutnosti znovu načíst celou stránku – dokáže v reálném čase měnit DOM strukturu stránky a nabízí tak možnosti vytvářet interaktivní webové aplikace [9].

Interpret jazyka JavaScript má každý webový prohlížeč implementován vlastním způsobem, nicméně všechny dodržují zavedené W3C standardy pro společnou kompatibilitu. JavaScript se díky tomu dá často využít na ulehčení zátěže serveru tím, že některé výpočty/operace přenecháme provést klienta.

3.2.4 jQuery

[2]jQuery je javascriptový framework s otevřeným zdrojovým kódem a stal se postupem let základem téměř každé webové aplikace. Obsahuje rozsáhlou knihovnu funkcí, které umožňují psát javascriptový kód úsporněji, přehledněji a s vyšší efektivitou [3]. Navíc je možné pro jQuery velice jednoduše vytvářet zásuvné moduly a na Internetu jsou jich tisíce a většinou šířené pod open-source licencí.

Díky jQuery lze často vyčerpávající zápisy v JavaScriptu napsat velice jednoduše a elegantně. Například vestavěný systém selektorů je syntakticky inspirován tím z CSS, nejlepší bude uvést srovnání na příkladu. Toto je kód v čistém JS pro skrytí `div` elementu s id `ele`:

```
document.getElementById('ele').style.display = 'none';
```

A zde je totožná operace napsaná v jQuery:

```
$('#ele').hide();
```

Mezi další užitečné funkce patří například vestavěný systém animací a nastavování CSS vlastností a vestavěná podpora pro technologii AJAX.

AJAX (Asynchronous JavaScript ad XML) umožňuje zasílat asynchronní HTTP požadavky na libovolné soubory/skripty na stejné doméně (bezpečnostní opatření) a jQuery nabízí knihovnu pro snadnou práci s tímto mechanismem. V době moderních interaktivních webových aplikací je tedy technologie AJAX nepostradatelným nástrojem pro vývojáře.

V mé práci pomocí AJAXu například získávám mapová data po načtení stránky a následně tato data vizualizuji jako vrstvy na mapě.

3.2.5 PHP

[3]PHP (HyperText Preprocessor) je interpretovaný jazyk od společnosti Zend. Může fungovat jak v rámci příkazové řádky na serveru, tak v kombinaci s podporovaným webovým serverem, kde webový server podle konfigurace převádí dané požadavky na PHP interpret (například Apache, Nginx).

Díky jednoduchému použití a přehledné syntaxi je to jeden z nejrozšířenějších jazyků pro tvorbu back-endu webových aplikací a za léta vývoje prošel mnohými změnami, kdy od verze PHP5 začal plně podporovat OOP. Já se rozhodl zvolit jednu z nejnovějších verzí – PHP7, která přináší spolu se Zend OPCache optimalizátorem drastické navýšení výkonu a rychlosti zpracování. PHP má spoustu vestavěných rozšíření, ty nejdůležitější jsou ve výchozím stavu povolené. Nabízí například rozšíření pro práci s MySQL databází, formáty JSON a XML, kryptografické funkce a spoustu dalších.

[2]https://jquery.com/
[3]http://php.net/

3.2.6 Leaflet.js

[4]Leaflet je javascriptový framework pro prezentaci map a mapových dat, je šířen pod otevřenou licencí a nabízí již v základu mnoho možností pro práci s mapami.

Umožňuje použít libovolné mapové podklady – jak veřejně dostupné (Google Maps, OpenStreetMap, ...) tak i vlastní, a dává uživateli i vývojáři možnost přepínat mezi nimi, pokud jich programátor zvolí více. LeafLet poskytuje programátorovi celkovou kontrolu nad všemi ovládacími prvky mapy a operacemi s mapou. Taktéž je možné přidat libovolné množství volně dostupných rozšíření, jejichž katalog se nachází přímo na stránkách frameworku.

Nabízí také přidávání překryvných vrstev několika různých typů, z rastrových vrstev především mapové dlaždice (`TileLayer`), vrstva bodů na mapě (`Markers`) a překrytí obrázkem (`ImageOverlay`), z vektorových pak např. plátno (`Canvas`), polygony, kruhy aj. Tyto vrstvy pak lze také slučovat do skupin (`LayerGroup`) a do uživatelského rozhraní lze přidat možnost je libovolně zobrazovat/skrývat.

3.2.7 Heatmap.js

[5]Heatmap.js je JS knihovna sloužící pro zobrazování heat map ve webových aplikacích. Funguje na jednoduchém principu – vývojář vytvoří novou instanci heat mapy a načte do ní potřebná data v tomto případě souřadnice x, y a četnost výskytu na dané souřadnici.

Heatmap.js je možné použít pro různé statistické aplikace, pro vizualizaci chování uživatele na stránce (aktivní pozice kurzoru myši apod.) a především pro mapová data. Díky dostupným rozšířením je možné heat mapy zobrazit nad Google Maps pomocí Google Maps API nebo, a to především, na Leaflet mapě díky pluginu Leaflet-Heatmap. Tuto knihovnu jsem využil na přidání další mapové vrstvy v podobě dynamické heat mapy.

[4] http://leafletjs.com/
[5] https://www.patrick-wied.at/static/heatmapjs/

Kapitola 4

Návrh a implementace

Probrali jsme teoretický základ, zhodnotili jsme situaci kolem existujících řešení a rozvrhli a popsali nezbytné technologie pro vývoj naší aplikace, nyní se zaměříme na návrh konkrétního řešení, definujeme požadavky, cílové skupiny a rozebereme samotnou implementaci.

4.1 Návrh řešení

4.1.1 Cílové skupiny uživatelů

Sada webových nástrojů má za cíl prezentovat různorodé výsledky, statistická data aj. z oblasti geografie a geolokalizace – tyto výsledky v podobě dynamické webové mapy jsou určeny jak pro běžné uživatele, tak pro odbornou veřejnost – vědce či analytiky.

Využití na akademické půdě připadá v úvahu především pro obor počítačové grafiky, kde lze tyto nástroje využít pro demonstraci dosažených výsledků, jak již samotný název práce napovídá. Z vědeckého pohledu pak může být relevantní webové rozhraní a vizualizace různých výpočtů nad nadefinovaným terénem.

Samozřejmou cílovou skupinou budou také vývojáři, kteří mohou využít jak serverovou sadu nástrojů, tak tu klientskou a rozšířit ji podle potřeby.

4.1.2 Požadavky na projekt

Většina požadavků vyplývá ze základních principů přístupnosti, uživatelské přívětivosti a flexibilního technického zpracování.

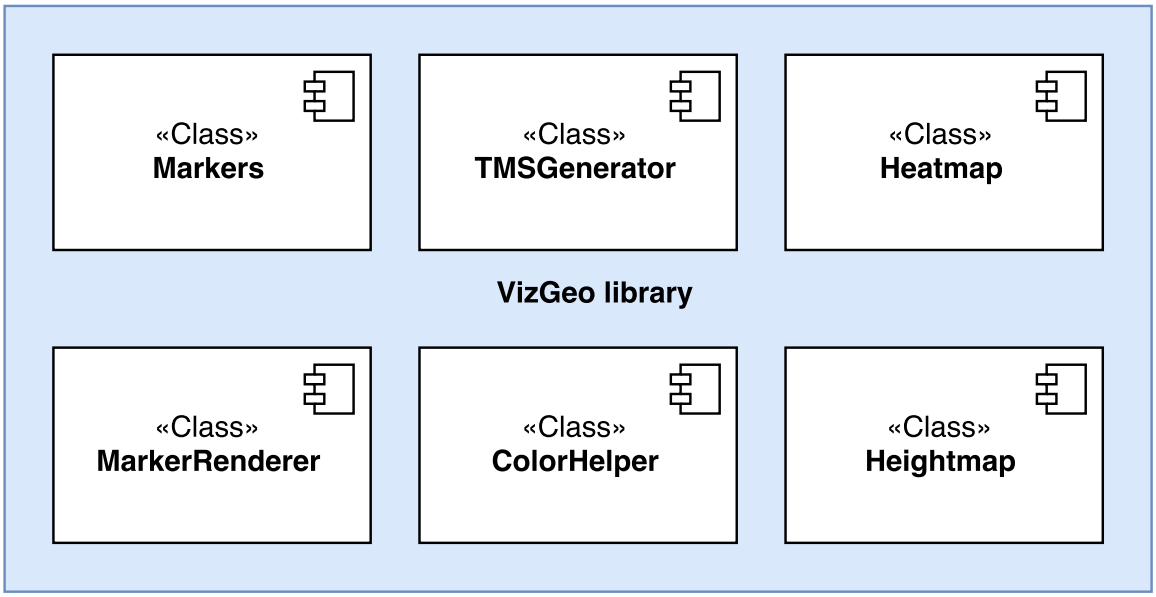
Webové rozhraní viditelné uživatelem musí být jednoduché, nikoliv však na úkor funkčnosti, snadno ovladatelné a intuitivní. Všechny ovládací prvky musí být logicky popsané a viditelně označené.

Back-endová část musí dodržovat všechny zaběhlé konvence ve stylu psaní zdrojového kódu, musí být rozčleněna do větších logických celků a kvalitně zdokumentována – to umožní vývojářům práci dále upravovat a rozšiřovat. Veškerá serverová část projektu by měla klást co nejmenší nároky na prostředky serveru, aby bylo dosaženo maximálního výkonu a efektivity, především pak v náročných úkonech, kde se dobrá optimalizace značně projeví na době vykonávání nejrůznějších operací, například generování několika desítek tisíc mapových dlaždic.

4.2 Knihovna VizGeo

Pro sjednocení všech nástrojů, které jsem implementoval, jsem se rozhodl vytvořit knihovnu s názvem `VizGeo`, která je napsaná v jazyce PHP 7. Navrhl jsem ji tak, aby bylo možné ji použít jak s rozhraním, které jsem pro ni vytvořil, tak naprosto samostatně. To zvyšuje flexibilitu použití a umožňuje knihovnu snadno aktualizovat či rozšířit nezávisle na systému, ve kterém je použita. Přehled součástí knihovny je znázorněn diagramem na obrázku 4.1.

Součástí knihovny je několik samostatných nezávislých tříd, každá se zaměřuje na jednu konkrétní oblast pro podporu webového rozhraní. To je postaveno na jednoduchém MVC frameworku a o front-end se stará javascriptová knihovna `VizMap`.



Obrázek 4.1: Diagram komponent celé knihovny

4.2.1 Generování bodů na mapě

Tato část knihovny se skládá ze dvou tříd, `Markers` a `MarkerRenderer`. Třída `Markers` slouží pro práci s body zobrazovanými na mapě – načítá databázi těchto bodů z poskytnutého souboru a získaná data zpracuje a připraví pro manipulaci a následný výstup. Třída je popsána diagramem na obrázku 4.2.

K tomu slouží metoda `parseFile()`, která jako vstupní argument přijímá cestu k souboru. Ten je rozdělen na 2 části – definice tříd a definice samotných bodů. Každá třída je definována unikátním názvem a barvou ve formátu RGBa (klasické RGB a tzv. alfa kanál, který nabývá hodnot $\langle 0, 127 \rangle$ a čím vyšší, tím průhlednější daná barva je). Body jsou potom dány zeměpisnými souřadnicemi v desetinném tvaru, velikostí daného bodu v pixelech, třídou a doprovodným popiskem, který se na mapě zobrazí po kliknutí na konkrétní bod.

Výsledný soubor má tedy následující formát:

```
nazev_tridy R G B a

zemepisna_sirka zemepisna_delka velikost trida doprovodny popis
```

A ukázkový soubor pak může vypadat třeba takto:

```
testovaci_trida 255 0 0 64
validacni 0 0 0 0

25.4912 9.3945 25 testovaci_trida popis bodu na mape
18.1248 11.3912 50 validacni popis druheho bodu
```

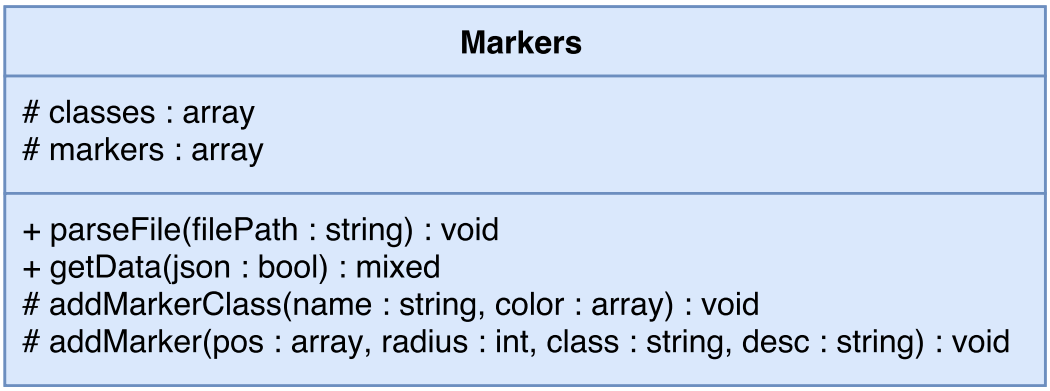
Metoda tedy soubor rozdělí na 2 části podle prázdného řádku, pro první část pak rozdělí data podle mezer a podle řádků a zavolá metodu `addMarkerClass()`, které jako první argument předá název třídy a jako druhý pole obsahující barvu třídy. Tento pár se pak vloží jako další objekt do třídní proměnné `classes`.

Pro druhou část pak opět rozdělí data podle řádků a řádky podle mezer a předá data metodě `addMarker()`, která uloží všechny údaje o bodě do objektu a ten opět vloží do třídní proměnné, tentokrát pole `markers`.

Pro získání výsledků zpracovaných dat slouží metoda `getData()`, která vrátí asociativní pole ve tvaru:

```
[
    'classes' => pole_objektu_trid,
    'markers' => pole_objektu_markeru
]
```

Pole je pak snadné převést například do formátu JSON a data vracet pro požadavky technologie AJAX, jak to dělám v mé práci.



Obrázek 4.2: UML diagram třídy `Markers`

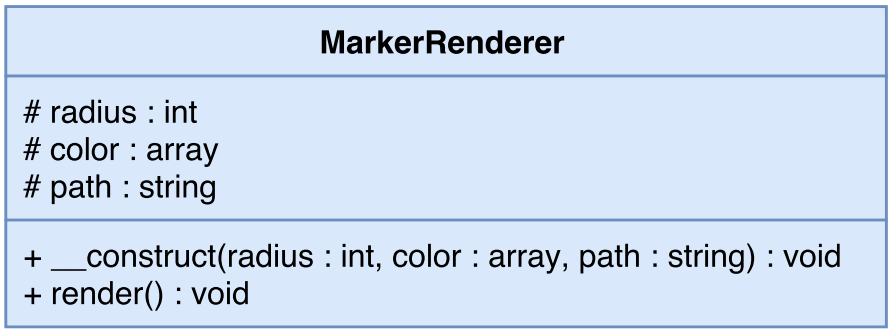
Třída `MarkerRenderer.class.php` (obr. 4.3) je pak určena ke generování a ukládání obrázků bodů v podobě puntíků na průhledném pozadí ve formátu PNG. Do konstruktoru třídy se jako parametry předají rozměr (průměr kruhu) v pixelech, pole reprezentující barvu a cesta ke složce, která bude sloužit jako úložiště.

O celý proces vytvoření se pak stará metoda `render()`, která nejdříve zjistí, jestli již marker s podobnými vlastnostmi neexistuje, a pokud ano, tak jej rovnou zobrazí a pokud ne, přichází na řadu samotný proces generování obrázku. Na to je využita grafická knihovna GD2, která je již v základu součástí PHP. Vzhledem k tomu, že výsledkem má být kruh a plátno obrázků je čtvercové, skript nastaví velikost plátna o 20 % větší, než je zvolený rozměr, aby nedošlo k oříznutí obrázku.

Protože knihovna GD v základu neumí vyhlazovat hrany pro elipsy, rozhodl jsem se použít jednoduchou techniku převzorkování. Obrázek je nejdříve vytvořen v dvojnásobné

velikosti, vygeneruje se kruh a poté se obrázek zmenší na polovinu, resp. na originální velikost, tím dojde k převzorkování obrázku a ke značnému zlepšení kvality hran.

Pro testovací účely jsem také vytvořil statickou metodu `generateRandomMarkers()`, která do zadaného souboru vygeneruje zadaný počet náhodných tříd a náhodných bodů, které k daným třídám přiřadí. Účelem funkce je vytvořit velký objem dat, aby bylo možné otestovat, jak rychle dokáže třída generovat například tisíce markerů najednou.



Obrázek 4.3: UML diagram třídy `MarkerRenderer`

4.2.2 Generování heat mapy

Pro zpracování a generování dat pro heat mapy jsem vytvořil třídu `Heatmap` (viz obrázek 4.4). Konstruktor při vytváření nové instance třídy očekává jako vstupní argument soubor, který bude sloužit pro ukládání a čtení dat (cache).

Základním vstupním bodem je opět metoda `parseFile()`, která jako argumenty přijímá cestu k souboru, který má importovat a typ souboru. Typy souboru mohou být 2 – maticový (výchozí) a soubor souřadnic. Maticový soubor obsahuje zeměpisné souřadnice rohů pomyslné matice a tabulku hodnot odpovídajících počtu četností pro danou oblast. Souřadnice konkrétních hodnot četností se pak rozpočítávají rovnoměrně podle daných krajových hodnot.

Vstupní maticový soubor má následující formát:

```
x_a,y_a x_b,y_b x_c,y_c x_d,y_d
hodnoty_cetnosti
```

Kde první řádek obsahuje čtyři páry souřadnic $[x, y]$ pro jednotlivé rohy a následující řádky už jsou samotnými hodnotami matice.

Ukázkový maticový soubor pak může vypadat následovně:

```
-3,0  2,0  -3,2  2,2
1  9  7  4
9  7  1  4
4  1  8  3
3  7  1  4
```

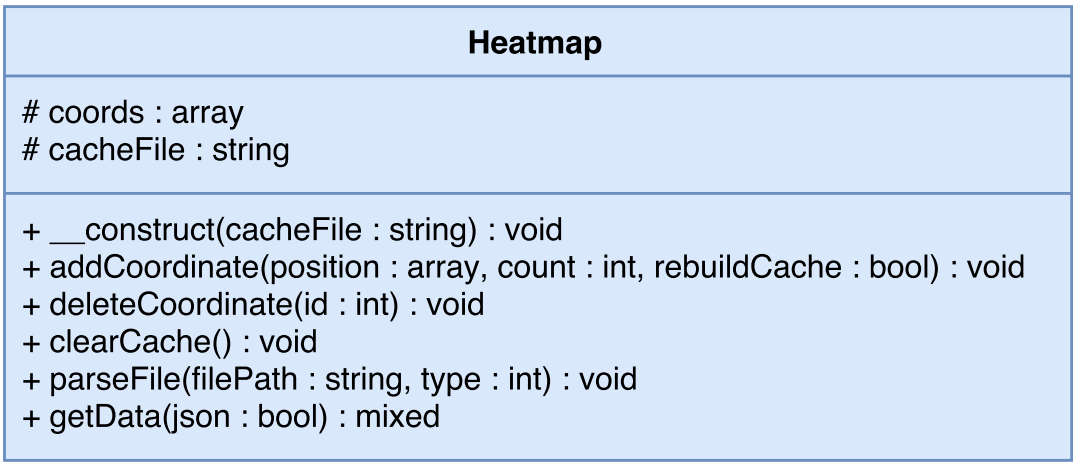
Po zpracování souboru pak přichází na řadu metoda `setupMatrix()`, která jako argumenty přijímá 4 rohové páry souřadnic a nepovinný argument pro určení vzdálenosti mezi jednotlivými body, ve výchozím stavu se počítá automaticky podle souřadnic a velikosti matice. Následuje metoda `addMatrixData()`, která ve výchozím stavu spočítá rozteč

mezi jednotlivými body a postupně pak volá metodu `addCoordinate()` s nově spočítanými souřadnicemi bodu a počtem výskytů.

Druhý formát – soubor souřadnic, má pak velice jednoduchý formát, souřadnice zeměpisné šířky a délky jsou oddělené čárkou a jednotlivé páry pak mezerami. Zde rovnou přichází na řadu metoda `addCoordinate()`, jenž je postupně volána pro všechny souřadnice ze souboru.

Po dokončení běhu metody `parseFile()` se automaticky volá metoda `buildCache()`, která ze zpracovaných dat vygeneruje soubor ve formátu JSON a uloží jej pro účely cachování.

Pro získání dat pro teplotní mapu je určena metoda `getData()`, která jako jediný parametr očekává booleovskou hodnotu `true` nebo `false`, a to podle toho, zda chceme výsledek rovnou získat ve formátu JSON nebo jej chceme převést na asociativní PHP pole. V mém projektu používám první možnost a výsledek vracím jako odpověď na AJAX požadavky.



Obrázek 4.4: UML diagram třídy `Heatmap`

4.2.3 Pomocná třída ColorHelper

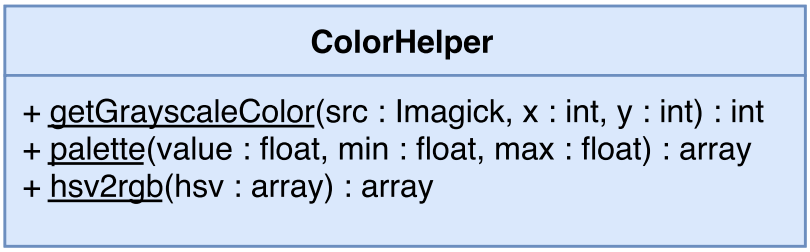
Jelikož v mé práci využívám četné operace s barvami, různými barevnými prostory a paletami, vytvořil jsem pomocnou třídu `ColorHelper`, která dokáže tyto úkony usnadnit. Třída je popsána diagramem na obrázku 4.5.

Pro získání černobílé hodnoty pixelu z obrázku slouží metoda `getGrayscaleColor()`, kterou využívám pro čtení hodnot z výškové mapy. Základní grafická knihovna v PHP (`GD2`) neumožňuje pracovat s obrázky s vyšší bitovou hloubkou než 8 bitů na barevný kanál, proto jsem zvolil rozšíření `Imagick` a závislou knihovnu `ImageMagick`, která podle konkrétního sestavení dokáže zpracovávat obrázky s vyšší barevnou hloubkou – například verze v sestavení Q16 dokáže číst 16bitové hodnoty. Jako argument pro `getGrayscaleColor()` se předává odkaz na právě zpracovávaný obrázek (instance třídy `Imagick`) a souřadnice pixelu, který chceme určit.

Další součástí je metoda `palette()`, která převádí vstupní hodnotu podle intervalu, který metodě také dodáme, na barvu z palety JET (1.2.2). Výslednou barvu vrací v podobě HSV modelu, jelikož s ním pak dále pracuji, měním saturaci dané barvy apod.

Poslední funkcí je metoda `hsv2rgb()`, která vstupní HSV barvu převádí zpět na RGB. Této funkce bylo v práci zapotřebí kvůli již zmíněným převodům palety do HSV, poté

provádím úpravy v HSV modelu a barvu převádím zpět do RGB. Důvody a konkrétní popis této situace uvedu v následující kapitole.



Obrázek 4.5: UML diagram třídy `ColorHelper`

4.2.4 Generování mapy sklonů

Protože podobné úkony umí většinou pouze robustní desktopové GIS aplikace, rozhodl jsem se pro různé analytické, verifikační a vědecké účely, implementovat některé operace nad výškovými mapami. Veškerou práci s nimi zaštiťuje třída `Heightmap` (obr. 4.7).

Jedním z výstupů této třídy je například mapa sklonů. Jedná se o obrázek obsahující informace o úhlech sklonů svahů a jejich velikostech zakódované do jednotlivých pixelů obrázku, a to následovně: úhel svahu určuje použitá barva z palety JET (1.2.2), čím větší úhel, tím vyšší hodnota v paletě. Velikost svahu je pak určena sytostí konkrétní barvy.

Směr sklonu je dán matematickým gradientem, ze kterého je možné spočítat jeho úhel a velikost. Mějme tedy gradient pro osu x a y označený g_x a g_y. Úhel sklonu potom spočítáme následujícím vzorcem:

$$\Theta = atan\left(\frac{g_y}{g_x}\right) \tag{4.1}$$

Výsledný úhel odpovídá hodnotám z intervalu $(-\frac{\pi}{2}; \frac{\pi}{2})$ dle definice funkce arkus tangens. Velikost je pak dána velikostí samotného gradientu a spočítá se následovně:

$$M = \sqrt{{g_x}^2 + {g_y}^2} \tag{4.2}$$

Základní vstupní metodou je `loadFromFile()`, která načte obrázek obsahující výškovou mapu ze souboru uvedeného argumentem. Z něj potom extrahuje informace o jednotlivých výškách (resp. hodnotách jednotlivých černobílých pixelů) pomocí zmíněné metody `ColorHelper::getGrayScaleColor()` a všechny hodnoty uloží do dvourozměrného třídního pole `map`.

Operaci vypočítání a vygenerování mapy sklonů zajišťuje metoda `computeSlopes()`, která má jeden nepovinný parametr – umístění pro výstupní obrázek, pokud není zadán, metoda rovnou vrací obrázek na výstup. Základem metody je dvojitý cyklus, který prochází získané hodnoty výšek jednu po druhé. Protože budeme pro obraz počítat konvoluci (1.2.1), prvním úkolem je pro každý bod zjistit jeho okolní pixely, k čemuž slouží metoda `createMask()`. Ta podle okolí pixelu vytvoří matici 3×3, pokud se jedná o krajní pixel a některé sousední pixely neexistují, využije se hodnota samotného středového pixelu. Tuto matici pak funkce vrátí jako výsledek zpět metodě `computeSlopes()`.

Následuje operace konvoluce metodou `convolution()`, která očekává jako vstup spočítanou masku hodnot. Konvolucí obrazu získáme pro každou hodnotu gradient na ose x a y,

spočítáme tedy dvě konvoluce s jádrem $[-1, 1]$ (resp. $[-1, 1]^T$ pro osu y). Protože pracujeme s diskrétním obrazem a získali jsme hodnoty pro celé okolí pixelu, konvoluční masky pro dané jádro budou vypadat takto:

Pro x:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

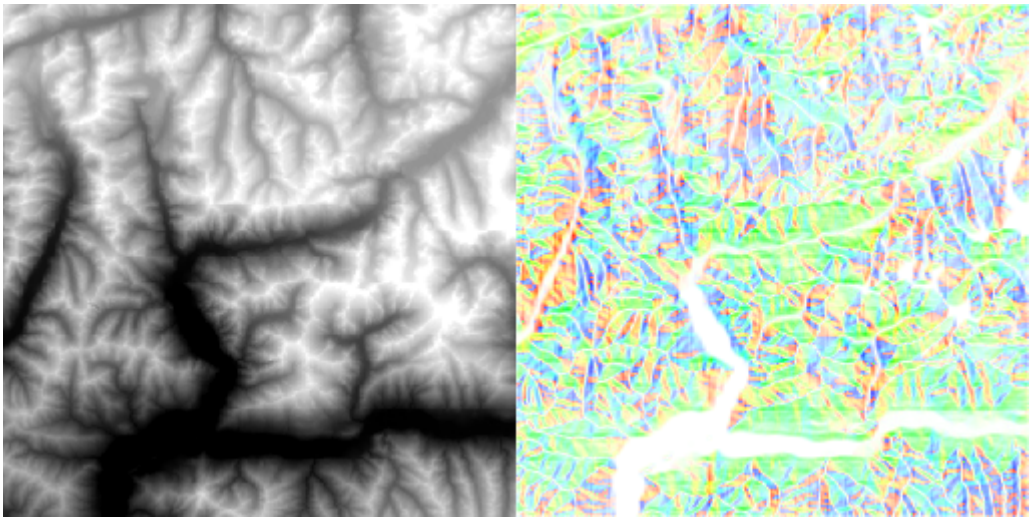
Pro y (transponovaná matice):

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Dosazením matic do vzorce pro konvoluci diskrétního obrazu získáme postupně 2 hodnoty gradientu – g_x a g_y, tyto hodnoty pak metoda `convolution()` vrací jako pole. Podle vzorců 4.1 a 4.2 pak metoda spočítá úhel a velikost sklonu.

V tuto chvíli má skript již všechny potřebné údaje, úhel sklonu nechá pomocí metody `ColorHelper::palette()` převést do barvy z JET palety reprezeotované HSV barevným prostorem, následně vydělí aktuální velkost úhlu s maximální pro dosažení hodnoty v rozsahu $\langle 0; 1 \rangle$ a tuto hodnotu použije jako saturaci dané barvy v HSV – tím docílíme toho, že čím delší úhel bude, tím sytější barva se na daném pixelu vykreslí. Modifikovaná HSV barva se pak pomocí metody `ColorHelper::hsv2rgb()` převede zpět do RGB podoby a získaná barva se zakóduje do nově generovaného obrázku mapy sklonů.

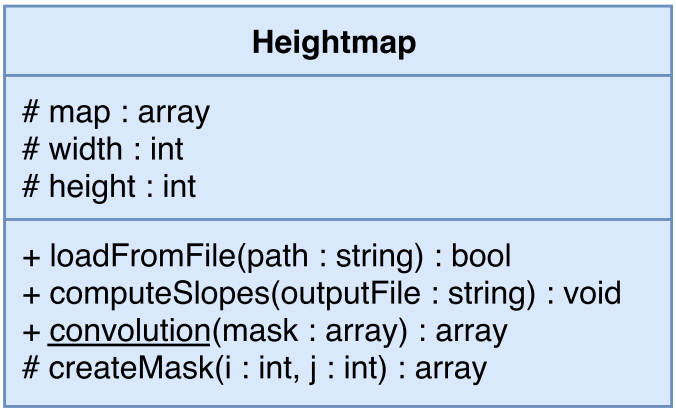
Jakmile proběhnou tyto operace pro všechny pixely obrázku, obrázek se buď rovnou vykreslí nebo uloží do souboru, dle nastavení uživatele. Výsledná mapa sklonů může vypadat například tak, jak znázorňuje obrázek 4.6.



Obrázek 4.6: Ukázková dlaždice (*tile*) vstupní výškové mapy vlevo a výsledná mapa sklonů vpravo

4.2.5 Dynamický generátor dlaždic

Jelikož pracujeme s mapovými daty a webové mapy fungují na principu zobrazování mnoha obrázkových dlaždic poskládaných k sobě, pro generování a prezentování různých výpočtů



Obrázek 4.7: UML diagram třídy `Heightmap`

potřebujeme opět sadu dlaždic. Pro tyto účely jsem vytvořil třídu `TMSGenerator`, která dokáže replikovat strukturu souborů a složek libovolných mapových dlaždic (především formát TMS), aplikovat na ně libovolné operace a uložit jako stejnou strukturu do nového adresáře.

Například pro již hotovou strukturu s dlaždicemi obsahujícími výškové mapy můžeme tuto třídu použít, abychom pro každou výškovou mapu spočítali mapu sklonů a všechny tyto výsledky (soubory) uložili do vlastní složky. Tím získáme nový zdroj dat pro webovou mapu. Třída je navržena tak, že může provádět libovolné množství operací a to jak sekvenčně tak paralelně.

Jelikož se jedná o časově náročný úkon, je žádoucí jej provádět spíše v prostředí příkazové řádky než jako HTTP požadavek, protože skript v příkazové řádce nemá omezenou dobu běhu a není závislý ani na ukončení spojení ze strany klienta jako v případě HTTP. Pro tento účel jsem vytvořil také statickou funkci `runInBackground()`, která jako vstupní parametr očekává příkaz, který poté spustí v příkazové řádce daného operačního systému. Aby bylo možné zadat z webového rozhraní požadavek na spuštění této operace, příkaz se spouští na pozadí, aby HTTP požadavek nečekal na dokončení běhu PHP skriptu. V případě Windows využívá metoda vestavěné funkce `popen()`, resp. `pclose()`, která spustí paralelní příkaz v příkazové řádce. Základní tvar příkazu vypadá takto:

```
CMD /C start /B <prikaz>
```

Jak jsem zjistil, toto bylo jediné spolehlivé řešení pro systém Windows tak, aby došlo skutečně ke spuštění příkazu a nedošlo k čekání ze strany skriptu na dokončení dané operace. Pro systémy postavené na Linuxu příkaz spouštím vestavěnou funkcí `exec()`, a příkaz vypadá následovně:

```
<prikaz> >/dev/null &
```

Pro běh celého systému v příkazové řádce jsem vytvořil vstupní skript `cli.php`, který je pro to uzpůsoben. Jako jediný argument příkazové řádky skript očekává cestu ekvivalentní té z webové verze, celý příkaz tak vypadá například takto:

```
php cli.php ajax/heightmap/tmsBatch >/dev/null &
```

Základní nastavení požadovaných operací se provádí v konstruktoru třídy, kterému se předá asociativní pole s třídami, které budou operace provádět – tzv. `handlers`. Klíči tohoto

pole budou samotné třídy ve [1]FQN formátu. Samotné hodnoty pro dané klíče obsahují v přesném pořadí nastavení metody, která se má z dané třídy volat, vstupní složka, která se má zpracovávat, a výstupní složka, do které se má uložit výsledek.

Druhým nepovinným parametrem je cesta ke skriptu, který by dané operace prováděl paralelně (tzn. více operací probíhajících souběžně) – je možné jej nastavit, pokud nám to dané prostředí dovolí (nefungovalo by například na sdíleném webhostingu apod.). Vytvořil jsem proto skript `tmsproc.php`, který v odlehčené podobě (aby vše proběhlo co možná nejrychleji) načte potřebné součásti systému a podle 4 parametrů provede požadovanou operaci. Parametry vypadají následovně:

```
php tmsproc.php -f <vstupni_slozka> -h <handler> -m <metoda> -o <
    vystupni_slozka>
```

Takové prvotní nastavení TMS generátoru může vypadat například takto:

```
<?php
use VizGeo;

$handlers = [ // handlery, resp. tridy, ktere budou provadeny
    'VizGeo\\Heightmap' => [
        'computeSlopes', // volana metoda
        STATIC_DIR.'/assets/tiles/heightmap', // vstupni slozka
        STATIC_DIR.'/assets/tiles/slopemap' // vystupni slozka
    ]
];

$tms = new VizGeo\TMSGenerator($handlers, CORE_DIR.'/tmsproc.php'
    ); // a cesta ke skriptu pro paralelni zpracovani
?>
```

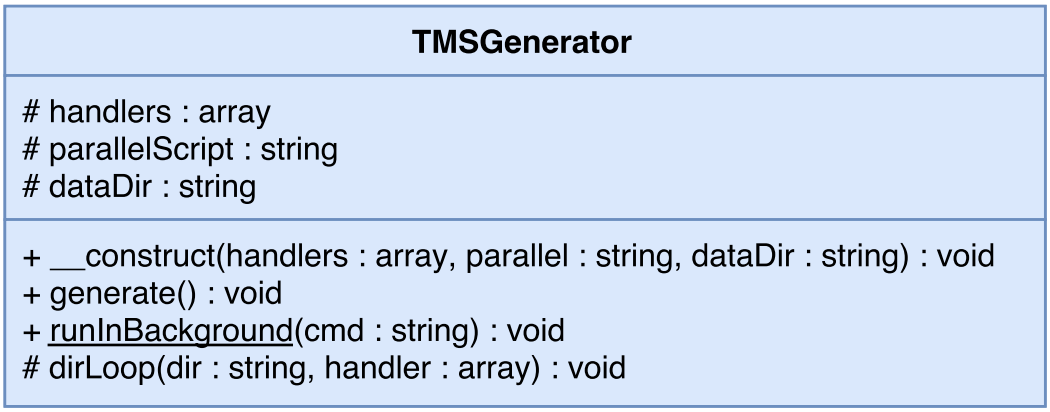
V tomto případě říkáme, že chceme použít třídu `Heightmap` pro práci s výškovými mapami a chceme pro ně spočítat mapy sklonů metodou `computeSlopes()`.

Jako hlavní spouštěč pak existuje metoda `generate()`, která projde všechny zadané třídy a pro každou spustí metodu `dirLoop()`, které předá výchozí cestu z handleru a handler samotný. Tato metoda následně rekurzivně projde všechny podsložky a jejich obsah, a pro každý soubor spustí operaci daného handleru – v případě, že byl zadán parametr specifikující cestu k paralelnímu skriptu, metoda pro daný soubor spustí paralelní běh skriptu `tmsproc.php` a pokračuje dále v prohledávání složek, nečeká tedy na dokončen. V opačném případě se data zpracovávají sekvenčně a skript nejprve čeká na dokončení každé operace, než přejde k procházení dalších souborů.

4.2.6 Knihovna pro JavaScript

Pro jednodušší vkládání mapy do stránek a její konfiguraci jsem vytvořil knihovnu v JavaScriptu. Celá javascriptová implementace je uložena v souboru `vizgeo.js` – obsahuje pár pomocných funkcí a třídu `VizMap`, popsanou diagramem na obrázku 4.9. Knihovna pro základní funkcionalitu vyžaduje jQuery a Leaflet, v případě některých funkcí jsou vyžadovány další knihovny či API.

[1]Fully qualified name – celý unikátní název třídy/objektu bez ohledu na aktuální kontext



Obrázek 4.8: UML diagram třídy `TMSGenerator`

Základní vložení a vykreslení mapy obstarává samotný konstruktor třídy, který jako parametry očekává identifikátor HTML elementu, do kterého se má mapa vložit a seznam základních mapových podkladů. Vytvoření základní mapy založené na OpenStreetMap může vypadat následovně:

```
var map = new VizMap('mapContainer', ['openstreetmap']);
```

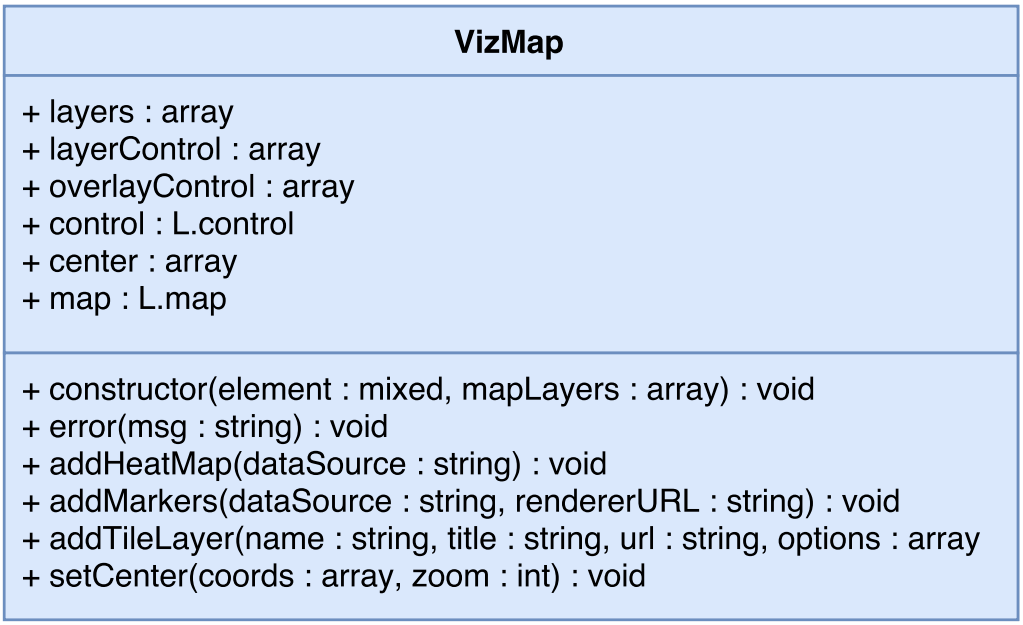
Povolené hodnoty pro seznam podkladových map jsou `openstreetmap`, `gmaps_roadmap`, `gmaps_terrain` a `gmaps_satellite`.

K dispozici jsou pak metody pro přidávání různých vrstev. Pro vložení bodů na mapu slouží metoda `addMarkers()`, která přijímá 2 argumenty – URL, na kterou má zaslat AJAX požadavek pro získání dat o bodech (je očekáván formát generovaný PHP třídou `Markers`) a URL, na které operuje třída `MarkerRenderer`. Dále je k dispozici metoda `addHeatmap()`, která má jako argument URL, ze které má získat JSON data pro teplotní mapu a následně ji přidá na mapu jako další vrstvu. Pro přidávání libovolných doplňkových dlaždicových vrstev (např. vrstva výškových map) slouží metoda `addTileLayer()`, které náleží 4 argumenty – interní název vrstvy, titulek pro zobrazení v uživatelském rozhraní, URL pro získání jednotlivých dlaždic (Tile Service) a nastavení pro danou vrstvu dle specifikací Leaflet.

4.3 Jednoduchý MVC framework

Na trhu je spousta dostupných open-source frameworků (české Nette, Zend, Laravel, ...), všechny jsou ale velmi komplexní, jejich zdrojové kódy mají desetitisíce řádků a obsahují spoustu funkcí, které jsou pro účely této práce zbytečné – projekt obsahuje minimalistické webové rozhraní rozdělené na veřejnou a administrační část.

Pro jednoduchý vývoj, přehledný kód a snadnou rozšiřitelnost jsem vyvinul vlastní, velmi úsporný framework, který je postaven na paradigmatu Model-View-Controller (MVC), kde Model implementuje veškerou logiku a interní operace, View je v podstatě šablona, která slouží pro výstup dat a Controller je spojovacím bodem mezi modelem a šablonou – přijímá signály od uživatele a podle toho získává/předává data konkrétním modelům a získaná data předává šabloně pro zobrazení uživateli. Součástí frameworku jsou tedy základní třídy `BaseModel`, `BaseController` a `View`, které je možné dědit a využívat jejich funkcionalitu. Základní princip fungování MVC modelu je popsán diagramem na obrázku 4.10. Veškeré



Obrázek 4.9: UML diagram JS třídy `VizMap`

zdrojové kódy pro tento framework jsou uloženy ve složce `Lib` a používají také stejný jmenný prostor (*namespace*). Vlastní implementace postavená na MVC je obsažena ve složce `App`.

Součástí MVC je také tzv. router, který převádí požadavky specifikované URL adresou z webového prohlížeče na konkrétní operace. Základní tvar adresy je následující:

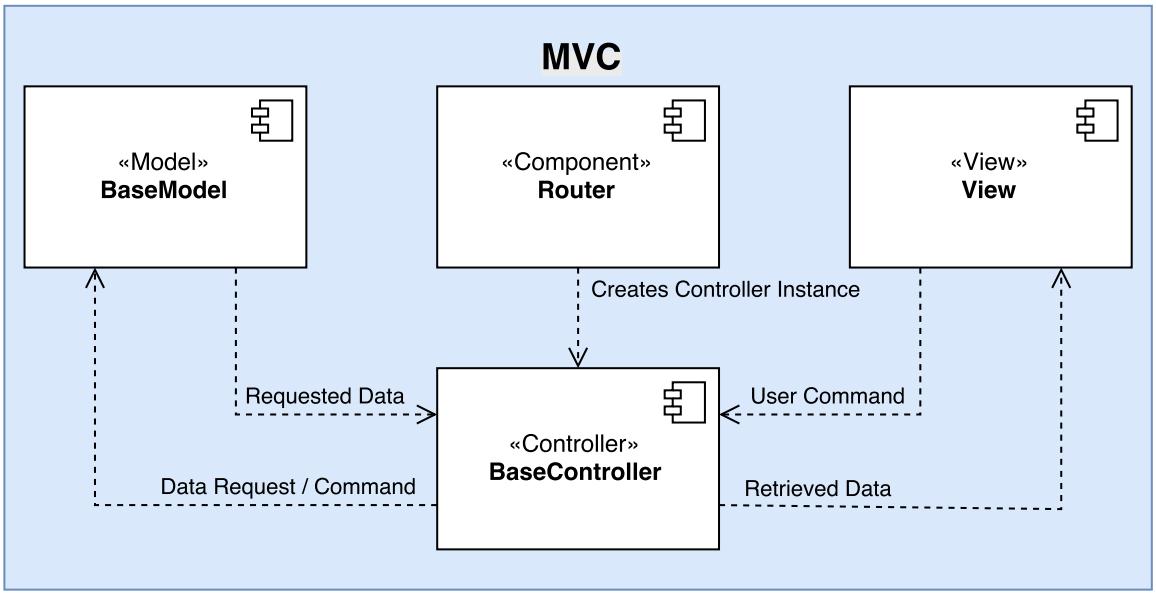
```
http://adresa_serveru/<kontroler>/<akce>/[<parametry/,...>]
```

Router zajistí, že pro takovou URL adresu se vytvoří instance třídy `<kontroler>Controller` a zavolá její metoda `<akce>Action()`, které se pomocí vnitřní třídní proměnné `params` předá pole parametrů (ty jsou rozděleny podle lomítek).

Pro práci s uživateli jsem v rámci implementace vytvořil třídu `User`, která dědí z `Lib\BaseModel` a pomocí technologie PHP sessions uchovává dočasná data o aktuálním přihlášení uživatele. Uživatelská data jsou uchovávána v textových souborech, kde název souboru je uživatelské jméno a obsah souboru je zašifrované heslo. Tyto soubory nejsou veřejně dostupné z webového serveru pro zajištění bezpečnosti.

4.4 Webové rozhraní

Jako uživatelská část této práce slouží minimalistické webové rozhraní, které obsahuje mapu se všemi potřebnými vrstvami určenými k prezentaci dosažených výsledků. Přítomna je také sekce O projektu, která stručně shrnuje cíl a účel projektu a informace o mně jakožto autorovi, dále je zde sekce Dokumentace, která obsahuje kompletní dokumentaci knihovny `VizGeo` vygenerovanou nástrojem DoxyGen a dokumentaci třídy `VizMap` vytvořenou pomocí JSDoc. Součástí je také administrační rozhraní pro pohodlnou správu dat, která se mají na mapě zobrazit.



Obrázek 4.10: Diagram komponent MVC modelu

4.4.1 Mapa s dynamickými vrstvami

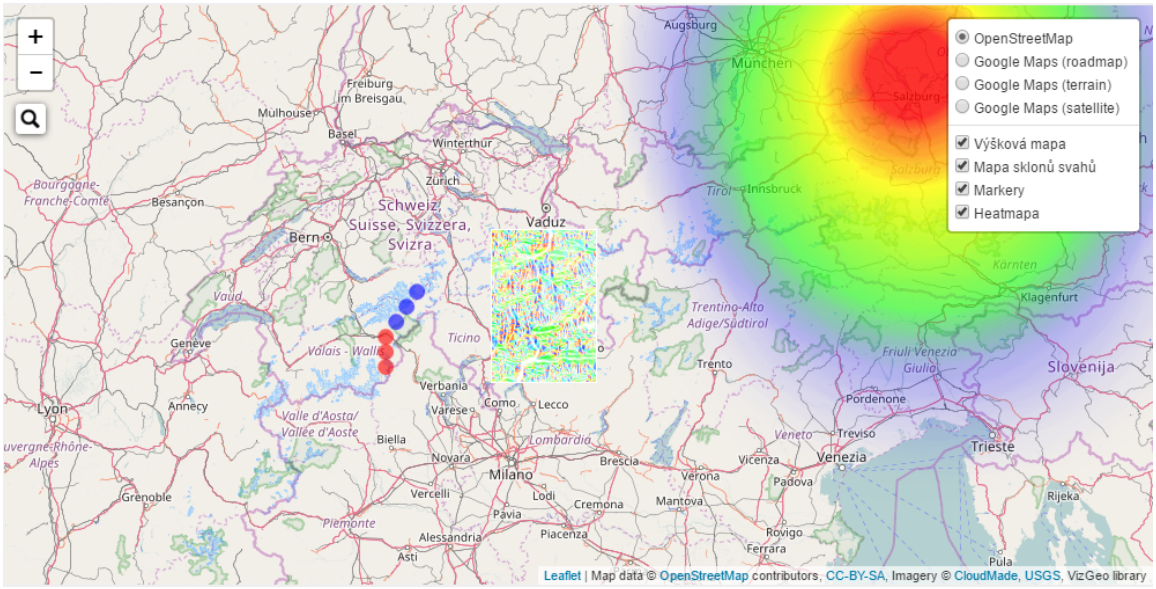
Hlavní součástí rozhraní je dynamická mapa, její základ je postaven na javascriptovém frameworku Leaflet. Nad ním jsem vytvořil již zmiňovanou třídu `VizMap`, která zapouzdřuje jeho funkcionalitu a usnadňuje integraci s nástroji vyvinutými v rámci této práce. Mapa umožňuje přepínat mezi zvolenými podkladovými vrstvami (Google Maps, OpenStreetMap, ...) a taktéž nabízí možnost skrývat či zobrazovat vlastní vrstvy.

Z knihovny `VizGeo` jsou k dispozici vrstvy Markery (body na mapě), Heat mapa, Výškové mapy a Mapy sklonů. Obrázek 4.11 znázorňuje webovou mapu s otevřeným menu vrstev a je na ní vygenerovaná jednoduchá teplotní mapa a stovky náhodných markerů pro demonstraci funkčnosti.

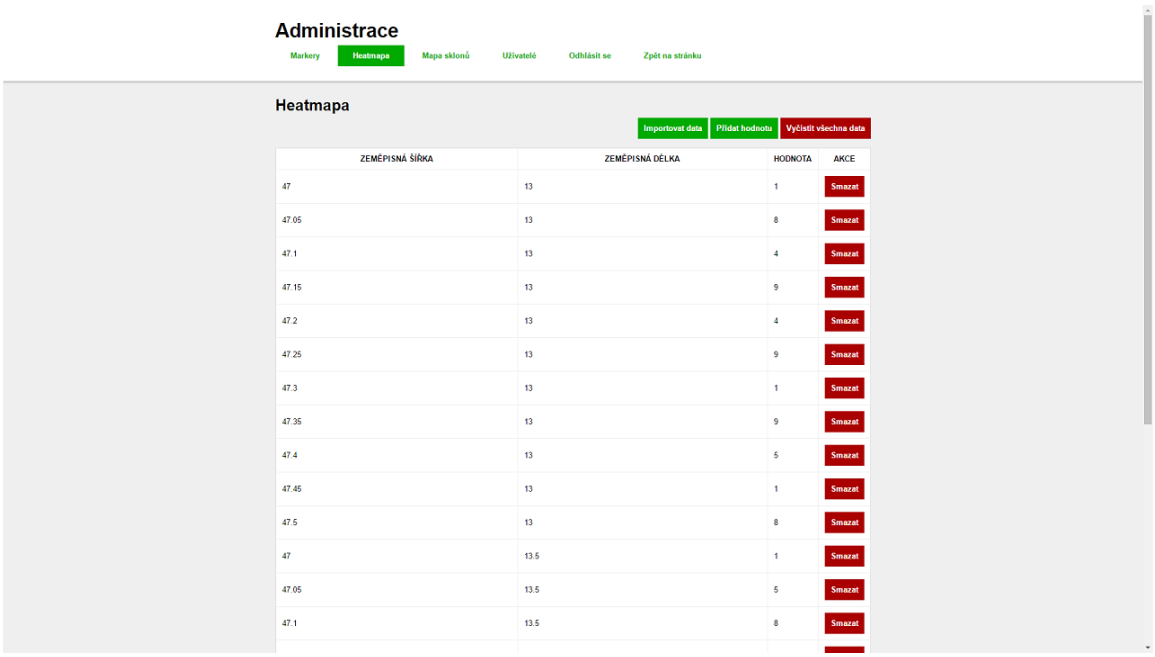
4.4.2 Administrační rozhraní

Nedílnou součástí webové aplikace je také přehledné administrační rozhraní umožňující spouštět generování nové mapy sklonů na pozadí, správu bodů na mapě a správu dat pro teplotní mapu.

Vedoucím práce mi byly nastíněny představy o fungování některých součástí administračního rozhraní. Při samotné implementaci a testování jsem však narazil na některé nedostatky, které by uživateli ztěžovaly práci. Navrhl jsem několik vylepšení a následně je implementoval. Jednalo se například o rozšíření funkcionality a upravení implementace pro správu dat teplotní mapy – původně bylo v plánu data dodávat pouze v podobě textových souborů. Pro snadnější a přehlednější správu dat jsem nicméně implementoval webové rozhraní, nové řešení je více flexibilní a umožňuje snadno přidávat/odstraňovat konkrétní data nebo importovat nová – to odstraňuje možné potíže při vizualizaci dat pro více samostatných dílčích oblastí. Dále jsem implementoval funkci na spuštění procedury generování nových mapových dlaždic přímo z webového rozhraní, není tedy nutné ručně zasahovat do příkazové řádky a je možné operaci provést i na dálku. K dispozici je také rozhraní pro správu uživatelských účtů administrace.



Obrázek 4.11: Ukázka webové mapy s různými vrstvami



Obrázek 4.12: Ukázka administračního rozhraní

Kapitola 5

Testování

Testování je součástí úspěšného a efektivního vývoje softwaru. Abych podložil a prověřil funkčnost mého řešení, bylo nutné provést několik testů, jak uživatelských, tak strojových. Všechny testy probíhaly na 64bitovém operačním systému Windows 10, hardware byl postaven na čtyřjádrovém procesoru Intel Core i5 6600 taktovaném na 3,9 GHz, kterému sekundovalo 16 GB operační paměti RAM. Jako primární systémový SSD disk byl použit Samsung 850 EVO a jako datový disk 7200otáčkový Western Digital Blue. Pro uživatelský test byl dále použit webový prohlížeč Google Chrome 58.0.3029.110 (64-bit) a editor zdrojových kódů Sublime Text 3.

5.1 Uživatelský test

Abych prověřil, zda je mnou vytvořené řešení efektivní, dobře implementované a zdokumentované, rozhodl jsem se provést uživatelský test. Vzhledem k povaze práce a jednoduchosti webového rozhraní jsem přistoupil spíše na otestování samotné implementace knihovny `VizGeo` a přidružených funkcí. Smyslem uživatelského testu je zjistit, zda a popř. kde se nachází slabiny implementace nebo dílčí nedostatky, nebo kde je dokumentace nedostatečná či nepřesná. V případě zjištěných nedostatků navrhnu jejich adekvátní řešení [11].

Pro test jsem si vybral 3 uživatele – v tomto případě neprobíhalo žádné řízení pro výběr kandidátů, zvolil jsem je dle vlastního uvážení. Všichni 3 kandidáti jsou kolegové z mé fakulty a většina z nich má zkušenosti s vývojem webových aplikací. Uživatelům jsem dodal zadání, potřebné nástroje a prostředí pro splnění úkolů a během jejich vypracování jsem je pozoroval a zaznamenával zjištěné skutečnosti.

5.1.1 Zadání úkolů

Účastníci testu dostali celkem 5 dílčích úkolů, všechny se týkaly základního zprovoznění mapy s vrstvami a jednoduché implementace funkcí pomocí knihovny `VizGeo`. Dostali k dispozici předem připravenou šablonu pro implementaci javascriptové mapy a šablonu pro implementaci PHP skriptu, který poskytoval mapě potřebná data. Také dostali k dispozici kompletní dokumentaci PHP i JS knihovny. Úkoly v přesném znění vypadaly následovně:

1. Pomocí JS třídy `VizMap` vložte do stránky základní mapu s podklady OpenStreetMap a Google Maps roadmap, pro mapu je přichystaný element s ID `mapContainer`.

2. (a) Pomocí PHP třídy `VizGeo\Markers` implementujte servírování dat pro JS knihovnu – data jsou uložena v souboru `data/markers.txt`. Výsledná data vypište jako JSON.
   (b) Pomocí PHP třídy `VizGeo\MarkerRenderer` implementujte generování obrázkových bodů pro mapu. Složka pro generování je `markers`.
   (c) Poté v části rozhraní přidejte vrstvu s vytvořenými body a napojte ji na vytvořená data a na nastavený renderer.
3. (a) Pomocí PHP třídy `VizGeo\Heatmap` implementujte servírování dat pro JS knihovnu – cache s daty je uložena v souboru `data/cache.json`, výsledná data vypište jako JSON.
   (b) Poté v části rozhraní přidejte vrstvu s heatmapou do mapy a napojte ji s právě vytvořeným výstupem dat.
4. Pomocí PHP třídy `VizGeo\Heightmap` načtěte výškovou mapu ze souboru `data/tile.png` a na výstup vygenerujte mapu sklonů.
5. Přidejte na mapu vrstvu s výškovými mapami z adresy

```
http://bp.lh/static/assets/tiles/heightmap/{z}/{x}/{y}.png
```

   Jako název vrstvy zvolte `heightmap` a titulek `Výšková mapa`. Pro vrstvu použijte následující nastavení:

```
{
        maxZoom: 12,
        tms: true
}
```

5.1.2 Provedená zjištění

Průměrná doba trvání testu byla 41 minut od přečtení zadání až po dokončení posledního úkolu, všichni kandidáti byli schopni zadané úkoly splnit bez větších obtíží, nicméně bylo zjištěno několik nedostatků.

Uživatel 1 měl obtíže se způsobem, jakým byla zdokumentována metoda `addMarkers()` v javascriptové knihovně `VizMap`, konkrétně pak URL adresa směřující na generátor obrázkových bodů. Dále došlo k menšímu nedorozumění s implementací třídy `Heatmap`, i když bylo v zadání uvedeno, že data jsou již uložena v cache a je potřeba je pouze načíst, chtěl použít metodu `parseFile()`, která však slouží na importování nových dat.

Stejný problém s dokumentací metody `addMarkers()` měl i uživatel 2, opět se mu zdálo, že dokumentace je zavádějící. Taktéž se mu zdálo neintuitivní pojmenování mapové vrstvy v konstruktoru JStřídy `VizMap`, konkrétně zkratka pro OpenStreetMap – `osm`.

Třetí a poslední kandidát měl méně zkušeností s vývojem webových aplikací, především pak s jazykem JavaScript, nicméně s pomocí dodané dokumentace byl schopen úkoly splnit – dokumentaci i implementaci hodnotil jako velmi intuitivní. U úkolu s generátorem markerů opět nastal stejný problém, jako v případě všech předchozích kandidátů.

5.1.3 Návrh řešení

Na základě získaných dat a posudků z testu jsem navrhl několik menších změn, především pak v dodané dokumentaci a také je implementoval. Jednalo se především o tyto úpravy:

- Vrstvu `osm` v konstruktoru třídy `VizMap` jsem přejmenoval na `openstreetmap` pro větší intuitivnost.
- Přepsal jsem dokumentaci pro metodu `addMarkers()`, aby bylo více zjevné, jaké parametry a v jakém formátu je nutné dodat.
- U téže metody jsem taktéž přejmenoval klíčové slovo `{rgb}` na `{color}`, toto klíčové slovo slouží jako substituce v URL adrese pro `MarkerRenderer`.

5.2 Strojový test výkonu

Časově nejnáročnější operace, kterou knihovna `VizGeo` vykonává, je počítání a generování map sklonů svahů do mapových dlaždic pomocí komponenty `TMSGenerator` a `Heightmap`, proto jsem se rozhodl co nejvíce optimalizovat kód, přidal jsem možnost paralelního zpracování jednotlivých mapových dlaždic (vizte 4.2.5) a následně jsem změřil rychlost generování. Tabulka 5.1 ukazuje, jaké jsou rozdíly v průměrné době sekvenčního a paralelního zpracování, každý test byl proveden 10× a hodnota byla zprůměrována. Na měření jsem použil vestavěnou funkci v PHP – `microtime(true)`, kterou jsem použil před začátkem časově náročné operace a po jejím dokončení, oba časy jsem odečetl a tím získal výsledný čas běhu.

generovaných dlaždic	**sekvenčně** [s]	**paralelně** [s]
327	114.71	32.92
674	218.32	63.72
1021	328.75	96.71
2042	653.25	191.67
10 366	3441.89	946.21

Tabulka 5.1: Výsledky provedených testů

Jak je z provedených testů patrné, paralelní zpracování dosáhne stejného výsledku za téměř čtvrtinu času oproti sekvenčnímu zpracování a čas potřebný pro dokončení stoupá úměrně s počtem generovaných dlaždic. Pokud to tedy dostupné prostředí dovolí, je použití paralelního zpracování víc než žádoucí. Jedinou nevýhodou, kterou jsem vypozoroval, je zvýšení zátěže CPU při provádění výpočtů, jelikož běží několik procesů souběžně, nicméně v porovnání s výhodami, jaké paralelní zpracování přináší, je to zanedbatelný nedostatek.

Kapitola 6

Závěr

Cílem práce bylo seznámit se s metodami reprezentace dat v kartografii, prozkoumat existující řešení nástrojů pro vizualizaci geografických dat, zjistit, jaké jsou dostupné technologie a vhodně je zvolit a navrhnout sadu webových nástrojů pro vizualizaci různých geografických jevů a dat. Na základě získaných vědomostí pak byla vytvořena sada webových nástrojů. Pro podložení dosažených výsledků bylo celé řešení otestováno uživateli i automatizovaným testem.

6.1 Dosažené výsledky

Požadována byla možnost zobrazovat na mapě body, teplotní mapy, libovolné vrstvy a možnost počítat mapy sklonů z dodaných výškových map. Dále bylo doporučeno, aby řešení bylo možné použít buď jako hotový celek v podobě webové aplikace, nebo samostatně vložit do libovolného projektu.

Všechny tyto požadavky byly splněny – sada nástrojů byla implementována především pomocí PHP a JavaScriptu a byla dodána v podobě webové aplikace s jednoduchým administračním rozhraním. Řešení bylo navrženo tak, aby bylo možné jej využívat i zcela samostatně v libovolných projektech. Hlavní součástí práce je webová mapa, na které se pomocí různých vrstev zobrazují všemožná geografická a statistická data – například sada dynamicky generovaných bodů, teplotní mapy, vrstvy výškových map aj. K práci byla dodána také kompletní dokumentace i testovací zdrojová data.

6.2 Budoucí vývoj

Celý systém byl navržen tak, aby bylo snadné jej rozšířit bez nutnosti většího zásahu do již hotových součástí – webová aplikace je postavena na modelu MVC, který umožňuje jednoduché a instantní rozšiřování funkcionality a samotná PHP knihovna byla vytvořena jako soubor samostatných tříd (resp. nástrojů). Jako možná rozšíření do budoucna připadají v úvahu například počítání fraktálnosti terénu z výškových map a zobrazování na výsledné mapě jako teplotní mapa. To by ovšem vyžadovalo získání nebo vytvoření nástroje pro řezání mapových dlaždic s vyšší bitovou hloubkou než 8 bitů na kanál, aby bylo možné určovat co nejpřesnější výšku z terénu – což samo o sobě tvoří námět na další rozšíření nebo dokonce samostatnou kvalifikační práci.

Literatura

[1] Čadík, M.; Vašíček, J.; Hradiš, M.; aj.: Camera Elevation Estimation from a Single Mountain Landscape Photograph. In *BMVC 2015*, 2015.

[2] Baros, M.: *Heightmap Terrain Rendering*. 2006.
URL http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/2006/0606/060508mb01/060508mb01.html

[3] Chaffer, J.; Swedberg, K.: *Mistrovství v jQuery*. Brno: Computer Press, první vydání, 2013, ISBN 9788025141038.

[4] Hays, J.; Efros, A. A.: IM2GPS: estimating geographic information from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, 2008.

[5] Lazaris, L.: *CSS okamžitě*. Brno: Computer Press, první vydání, 2014, ISBN 9788025141762.

[6] Ozyesil, O.; Singer, A.: Robust Camera Location Estimation by Convex Programming. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] Písek, S.: *HTML: začínáme programovat*. Praha: Grada, Čtvrté vydání, 2014, ISBN 9788024750590.

[8] Žára, J.; Beneš, J.; Sochor, J.; aj.: *Moderní počítačová grafika*. Brno: Computer Press, vyd 1. vydání, 2004, ISBN 8025104540.

[9] Žára, O.: *JavaScript*. Brno: Computer Press, první vydání, 2015, ISBN 9788025145739.

[10] Saurer, O.; Baatz, G.; Köser, K.; aj.: Image based geo-localization in the alps. *International Journal of Computer Vision*, ročník 116, č. 3, 2016.

[11] Unger, R.; Chandler, C.: *A project guide to UX design*. Berkeley: New Riders, druhé vydání, 2012, ISBN 9780321607379.

[12] Voženílek, V.; Kaňok, J.: *Metody tematické kartografie*. Olomouc: Univerzita Palackého v Olomouci pro katedru geoinformatiky, 2011, ISBN 9788024427904.

Příloha A

Obsah přiloženého CD

- Adresář `webtools` – kompletní zdrojové kódy webového rozhraní včetně knihovny `VizGeo`
  - Adresář `data` – textová data pro knihovnu VizGeo
  - Adresář `src` – zdrojové kódy pro PHP, knihovny, šablony
  - Adresář `static` – statická data pro webovou aplikaci: CSS, JavaScript, obrázky
  - Adresář `doc` – dokumentace
- Adresář `testdata` – obsahuje testovací data pro VizGeo knihovnu: markery, data pro teplotní mapu a dlaždice výškových map
- Adresář `text` – text této bakalářské práce ve formátu PDF včetně všech zdrojových souborů

Příloha B

Manuál

Pro zprovoznění webové aplikace se všemi jejími funkcemi je vyžadován následující software:

- Webový server Apache
- Modul `rewrite` pro Apache a povolené soubory .htaccess
- PHP verze 7.0 a novější
- PHP rozšíření GD2 pro generování obrázků
- PHP rozšíření Imagick pro správné fungování operací nad výškovými mapami
- Knihovna ImageMagick, se kterou pracuje PHP Imagick

Je také důležité v konfiguraci PHP nastavit `short_open_tags` na `On`.

Pokud je nainstalován veškerý potřebný software, do adresáře, který je nastaven ve webovém serveru, zkopírujte kompletní obsah složky `webtools` z přiloženého CD. Následně je potřeba nastavit cesty v konfiguračním souboru `config.json`, který se nachází ve složce `src/App`. Především je potřeba nastavit následující parametry:

- `site_url` – URL, na které je aplikace dostupná, bez lomítka na konci, např. `http://app.local`
- `admin_url` – URL, na které je dostupná administrace, musí být ve tvaru `<site_url>/admin`, opět bez lomítka na konci, např. `http://app.local/admin`
- `assets_url` – URL ke složce `static/assets`, opět bez koncového lomítka, např. `http://app.local/static/assets`
- `assets_dir` – serverová cesta ke složce `static/assets`, lze použít klíčové slovo `{DOCROOT}`, které aplikace nahradí za cestu uvedenou jako `DOCUMENT_ROOT`

Dále je potřeba v souboru `src/bootstrap.php` zkontrolovat, zda nastavené konstanty cest odpovídají a případně je upravit.

V případě použití systému na bázi Unixu je nutné nastavit oprávnění pro zápis na následující adresáře:

- `data` a všechny jeho podadresáře/soubory

- `static/assets/tiles/slopemap`
- `static/img` a všechny jeho podadresáře

Pokud bylo vše správně nakonfigurováno, webové rozhraní by mělo být plně funkční. Základní přihlašovací údaje do administračního rozhraní jsou `admin` : `pass`. Zkušební verze projektu je také dostupná na adrese http://fit.hodoval.cz/, jedná se však o sdílený webový hosting, není tedy možné spouštět náročné operace jako generování mapy sklonů. Všechny dostupné mapy jsou zde nicméně vygenerovány.

Data pro markery jsou uložená v souboru `data/markers/data.txt`, cache heat mapy pak v `data/heatmap/cache.json`, výškové mapy je možné nahrát do adresáře `static/assets/tiles/heightmap` (je třeba dodržet strukturu TMS). MVC framework je obsažen v adresáři `src/Lib`, knihovna `VizGeo` v `src/VizGeo` a složka `src/App` slouží pro vlastní implementaci. Součástí frameworku je autoloader, není proto nutné načítat další třídy ručně.