



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova**

BAKALÁŘSKÁ PRÁCE

Jméno Příjmení

Název práce

Název katedry nebo ústavu

Vedoucí bakalářské práce: Vedoucí práce

Studijní program: studijní program

Studijní obor: studijní obor

Praha ROK

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne
Podpis autora

Poděkování.

Název práce: Název práce

Autor: Jméno Příjmení

Katedra: Název katedry nebo ústavu

Vedoucí bakalářské práce: Vedoucí práce, katedra

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Name of thesis

Author: Jméno Příjmení

Department: Name of the department

Supervisor: Vedoucí práce, department

Abstract: Abstract.

Keywords: key words

Obsah

Úvod	4
1 Analýza požiadavkou	6
1.1 Požiadavky	6
1.2 Funkčné požiadavky	6
1.2.1 Funkčne požiadavky ako používateľské príbehy	7
1.2.2 Funkčne požiadavky ako use case scenáre	11
1.2.3 Zamietnuté funkčne požiadavky	14
1.3 Nefunkčné požiadavky	15
2 Mapové webové aplikácie	17
2.1 Google Maps	17
2.2 Mapy.cz	19
2.3 OpenStreetMap	20
3 Analýza	21
3.1 The Resource Description Framework	21
3.1.1 RDF Dátový Model	21
3.1.2 Typy RDF dát	22
3.1.3 Viaceré grafy	22
3.1.4 RDF slovníky	22
3.2 Wikidata	23
3.3 Wikidata Query Service	26
3.3.1 SPARQL pre Wikidata	26
3.3.2 Label service	29
3.3.3 Geopriestorové vyhľadávanie -Around service	30
3.3.4 MediaWiki API Query Service	30
3.4 Wikimedia service	31
4 Návrh	33
4.1 Návrh softvérovej architektúry	33
4.1.1 Systém Kontext diagram	34
4.1.2 Kontajner diagram	35
4.1.3 Server Komponent diagram	35
4.1.4 SPA Komponent diagram	36
4.2 Návrh databázy	37
5 Implementácia aplikácie	42
5.1 Technológie použité pre vývoj	42
5.1.1 Backend	42
5.1.2 Frontend	43
5.2 Implementácia SPARQL dotazov	43
5.2.1 Vyhľadávanie mapových objektov	44
5.2.2 Popis a obrázok	47
5.2.3 Odkaz na Wikipédia článok	47
5.2.4 Detaily mapového objektu	48

5.2.5	Vyhľadávanie kategórii	51
5.2.6	Vyhľadávanie administratívneho územia a regiónov	53
5.2.7	Vlastnosti ako filtre	55
5.2.8	Obmedzenia vlastnosti	57
5.2.9	Vyhľadávanie hodnôt u filtrov	59
5.2.10	Vyhľadávanie skupiny mapových objektov na základe parametrov	60
5.3	Štruktúra projektu	65
5.4	Backend	65
5.4.1	Vytváranie SPARQL dotazov	66
5.4.2	Posielanie a získavanie dát z Wikidata	67
5.4.3	Databáza	68
5.4.4	REST API	72
5.5	Frontend	77
5.5.1	Funkcionálne komponenty	77
5.5.2	Vytvorenie React aplikácie	79
5.5.3	Implemenátacia World collection	79
6	Užívateľská dokumentácia	91
6.1	Prehľad kolekcii	91
6.2	Vyhľadávanie mapových objektov	94
6.3	Pridávanie mapových objektov	99
6.4	Spravovanie kolekcii	100
7	Testovanie	103
7.1	Playwrig	103
7.2	Inštalácia a spustenie	103
7.3	Playwright testy	104
7.4	Prehľad testov	106
8	Ukážky	108
8.1	Jednoduché příklady	108
8.2	Matematické vzorce a výrazy	109
8.3	Definice, věty, důkazy,	110
9	Tabulky, obrázky, programy	112
9.1	Tabulky	112
9.2	Obrázky	113
9.3	Programy	113
10	Odkazy na literaturu	118
10.1	Několik ukázk	118
11	Formát PDF/A	119
Závěr		120
Seznam použité literatury		121
Seznam obrázků		122

Seznam tabulek	123
Seznam použitých zkratek	124
A Přílohy	125
A.1 První příloha	125

Úvod

Na internote existujú stránky, ktoré ponúkajú funkciu spojenú s mape. Jedná sa o klasické pozeranie máp, vyhľadávanie objektov na mape, alebo znázornenie dát pomocou mapy. V súčasnej dobe na internote neexistuje webová aplikácia pre cestovateľov alebo aj bežných ľudí, ktorá by vhodne umožňovala spravovanie objektov na mape. Pod pojmom "spravovanie" myslime nasledovné funkcionality:

- Zaznamenávať si objekty na mape.
- Priradiť objekt do kolekcie objektov.
- K objektu priradiť poznámky a nastaviť objektu ikonu, ktorá reprezentuje objekt ako bod na mape.
- Funkciu návštevnosti, teda možnosť zaznamenať, či daný objekt bol používateľom navštívení a poprípade poznamenať aj dátum návštevy.

Ďalšia chýbajúca funkciu je detailnejšie vyhľadávanie objektov, ktoré splňujú požiadavky na základe parametrov zadaných používateľom ako:

- Definovať aké typy objektov majú byt vyhľadávane, typom myslime skupinu objektov, ktoré majú spoločné vlastnosti a človek ich zaradí do rovnakej skupiny. Typ môže byt napr. hrad, mesto, múzeum...
- Vymedziť územie, kde hľadáme objekty.
- Definovať a zadať rôzne vlastnosti objektu, ktoré sú rozumne pre daný typ objekt. Napríklad pre objekt reprezentujúci mesto vieme vymedziť vo vyhľadávaní aby nájdene mesta splňovali vlastnosť, a tou je vlastnosť populácia, čo reprezentuje číslo počtu obyvateľstva daného mesta aby bolo väčšie ako jeden milión.

Posledná funkciu je možnosť poskytovania informácií o konkrétnom objekte.

Webová služba Google maps(to do : odkaz) poskytuje možnosť spravovania objektov, ale bez rozumnej funkciu nastaviť návštevnosť. Vieme zaznačiť objekt na mape ako bod, ktorému vieme nastaviť ikonu a farbu, napísat poznámky a priradiť do kolekcie objektov. Avšak nevieme rozumne zaznamenať, či bol daný objekt navštívený. Museli by sme si to jedine poznamenať do poznámok a pre grafickú reprezentáciu by sme mohli napríklad používať sivú farbu ikonky pre zatial nenavštívený objekt. Google maps taktiež poskytuje možnosť vyhľadávať konkrétné objekty podľa názvu objektu, alebo hľadaním v blízkosti objekty podľa zadaného kľúčového slova, ktoré opisuje nejakú skupinu objektov. Jedna sa o typ objektov, ako sme už spomenuli. Nenájdeme tu možnosť zdefinovať lepšie naše požiadavky, ktoré by ovplyvňovali a vymedzovali vyhľadávanie. Možnosť poskytovania informácií o konkrétnom objekte Google maps neposkytuje. Informácie by sme museli vyhľadávať cez rôzne stránky, poskytujúce informácie.

Tento pracou chceme vytvoriť webovú aplikáciu, ktorá by obsahovala uvedené funkcionality. Vďaka tejto aplikácii by sa uľahčilo spravovanie a vyhľadanie mapových objektov pre skupinu používateľov ako sú napríklad cestovatelia.

Cieľom práce je:

- analyzovať používateľské požiadavky
- analyzovať zdroje a získavanie dát, ktoré bude aplikácia využívať
- navrhnúť a implementovať webovú aplikáciu
- navrhnúť dátový model, ktorým budeme ukladať dátá do databázy a vybrať vhodnú databázu pre tento projekt
- aplikáciu otestovať

Práca opisuje základne technológie a princípy, ktoré sme v prací použili, analýzu používateľských požiadavok, ktoré sú očakávané od aplikácie. Následne sú zanalyzované dátá, ktoré potrebujeme a dotazy pre získavanie týchto vyhľuvujúcich dát na základe vstupných parametrov. Rozoberanie do detailov aké sú možnosti definovať obmedzenia vyhľadávania objektov a ako museli byt optimálizované aby sme dostali výsledky v rozumnom čase.

Ďalej popíšeme navrch architektúry aplikácie.

V implementačnej časti je opísaný vývoj aplikácie. Aké technológie boli použité, podrobne si popíšeme jednotlivé časti programu a ako medzi sebou tieto časti interagujú.

V užívateľskej časti je opísaný jednoduchý postup ako pracovať s aplikáciou, spravovať objekty a hlavne ako vyhľadávať objekty na základe parametrov. Nájdeme tam aj konkrétné príklady, ktoré ukazujú konkrétné vyhľadávania s danými parametrami.

Následne v poslednej časti ukážeme ako sme aplikáciu otestovali.

1. Analýza požiadavkou

V tejto kapitole sú najprv uvedené jednoduché definície požiadavkou ktoré sa zameriavajú na funkciaľnosť aplikácie. Uvedieme skupiny do ktorých sa požiadavky delia na základe zámeru požiadavkou. Následne sú analyzované funkčné a kvalitatívne (nefunkčné) požiadavky očakávane od aplikácie. Jednotlivé požiadavky sú spísane v tejto kapitole. Na konci kapitoly sú uvedené aj požiadavky ktoré boli počas návrhu a implementácie zamietnuté.

1.1 Požiadavky

Pre navrhnutie a vytvorenie úspešného softwarového projektu je potrebné na začiatku identifikovať a definovať rozumné vlastnosti, ktoré stakeholderi požadujú a očakávajú od aplikácie. V rámci tejto práce je stakeholder používateľ, ktorý bude používať našu aplikáciu. Používateľom môže byť hoci aký človek, ale zámer našej aplikácie prevažne mieri na užšiu doménu používateľov. Do tejto domény patria cestovatelia, turisti a návštevníci kultúrnych miest.

Spomíname vlastnosti je potrebné vyjadriť a opísat v štruktúrovanej forme. Požiadavka opisuje a definuje vlastnosť, ktorú používateľ vyžaduje od aplikácie. Vďaka požiadavkám je jednoduchšie navrhnuť architektúru aplikácie, implementovať aplikáciu a následne nám to pomôže aj aplikáciu otestovať.

Požiadavky sa delia do dvoch hlavných skupín. Jednou skupinou sú takzvané funkčné požiadavky, druhou sú kvalitatívne požiadavky. Kvalitatívne požiadavky sa môžu označovať aj ako nefunkčné požiadavky.

Funkčné požiadavky sa dajú jednoducho definovať ako služby, ktoré sú poskytované aplikáciou. Inak povedané čo všetko dokáže aplikácia urobiť, akú funkciaľnosť je schopná poskytnúť používateľovi. Ďalej definuje kto bude využívať konkrétnu službu. Funkčné požiadavky zároveň opisujú vstupné dátu a snažia sa prezentovať očakávaný výstup. Ak aplikácia nesplňa niektorý z funkčných požiadaviek, potom je možné aplikáciu prehlásiť za nefunkčnú.

Kvalitatívne požiadavky alebo aj nefunkčné požiadavky sa dajú jednoducho definovať ako požiadavky, ktoré opisujú ako dobre aplikácia poskytuje svoje služby. Zameriavajú sa na to ako systém zvládne splniť funkčnú požiadavku. Na tieto požiadavky je potrebne sa pozerať z pohľadu rôznych kvalít ako sú napríklad výkon a dostupnosť.

1.2 Funkčné požiadavky

Na základe úrovni poskytnutých detailov sa funkčné požiadavky delia na :

1. Používateľské požiadavky. Tie sú spísane pre používateľov aplikácie v prirodzenom jazyku, ktorý je zrozumiteľný aj bez technických znalostí. Na vyjadrenie používateľských požiadaviek sa používajú takzvané používateľské príbehy.
2. Systémové požiadavky. Tie sú na rozdiel od používateľských požiadaviek štruktúrované vo forme scenárov. Na vyjadrenie sa používajú takzvané use

cases(prípady použitia), kde sú detaility opísané do väčšej hĺbky. Zároveň definujú čo by malo byť implementované.

Na začiatku prace prebehla analýza funkčných požiadavkou, ktoré by používateľ mohol očakávať od aplikácie.

Počas analýzy a samotnej implementácie aplikácie sme niektoré funkčné požiadavky pridali a tým obohatili funkcionality aplikácie na základe získania nových vedomostí . Na druhu stranu boli niektoré zamietnuté z dôvodu časovej tiesne alebo usúdenia, že tieto funkčné požiadavky sa nedajú rozumne implementovať.

1.2.1 Funkčne požiadavky ako používateľské príbehy

V tejto sekcii je výpis všetkých funkčných požiadavkou vyjadrených formou používateľských príbehov.

Používateľský príbeh je vysvetlenie funkcie aplikácie neformálnou formou z pohľadu koncového používateľa aplikácie. Cieľom používateľského príbehu je jednouchého, bez podrobnejších detailov, opísat funkčne požiadavky. Každý používateľsky príbeh má nasledujúcu formu:

Ako <druh používateľa>,
som schopný/potrebujem <urobiť niečo>,
protože <dôvod>.

Používateľské príbehy sú súčasťou agilného vývoja, pretože dôraz kladú na potreby používateľa, ktoré sa môžu počas vývoja meniť.

Nasleduje výpis jednotlivých funkčných požiadavkou, ktoré sú rozdelene do dvoch skupín na základe funkcionality. Prvou skupinou sú funkčne požiadavky pre vyhľadávanie a získavanie dát a tou druhou sú funkčné požiadavky spravujúce mapové objekty.

Ešte predtým než uvedieme funkčne požiadavky si vysvetlime zopár pojmov, ktoré sa v nich vyskytujú.

- Pojmom "objekt"je myslený mapový objekt, ktorý je možne reprezentovať ako bod na mape.
- Pojmom "kolekcia"je myslený zoznam, list , do ktorého sa priradzujú, ukladajú objekty. Každá kolekcia má svoje jedinečne meno. Kolekciu si vytvára sám používateľ aplikácie a je na ňom aké objekty do kolekcie budú patrīť.
- Pojmom "zoznam tried, ktorých je objekt inštanciou"rozumieme zoznam všetkých tried, respektíve kategórii ktoré definujú konkrétnejšie objekt. Napríklad pre objekt mesto "Praha" tento zoznam obsahuje triedy ako "veľké mesto", "hlavné mesto" a "turistická destinácia". Tento zoznam poskytuje používateľovi informáciu čo je daný objekt zač, pod akú kategóriu spadá.
- Pojmom "detailey"objektu rozumieme zoznam dvojíc, kde prvý člen je kľúč definujúci názov vlastnosti objektu a druhý člen je hodnota tejto vlastnosti. Napríklad dvojica "kontinent" a "Európa" nás informuje o tom, že objekt je súčasťou kontinentu Európa.

Funkčne požiadavky pre vyhľadávanie a získavanie dát:

1. Používateľ je schopný vyhľadať konkrétny mapový objekt podľa zadania názvu objektu, pretože si ho môže chcieť uložiť do svojej kolekcie.
2. Používateľ potrebuje od aplikácie aby mu pre vyhľadaný objekt získala informácie definujúce objekt ako súradnice, meno, popis a zoznam tried, ktorých je objekt inštanciou, aby mal používateľ o objekte informácie identifikujúce objekt.
3. Používateľ potrebuje od aplikácie aby mu pre vybraný objekt získala obrázok objektu, aby používateľ mal predstavu ako objekt vyzerá.
4. Používateľ je schopný požiadať aplikáciu aby mu pre vybraný objekt vyhľadala odkaz na článok, sídliaci na Wikipédii, ak článok existuje, aby používateľ mal možnosť sa o objekte dozvedieť viac informácií vo forme článku.
5. Používateľ je schopný požiadať aplikáciu aby mu získala podrobnejšie informácie o objekte(budeme označovať ako detaile objektu), medzi ktorými vie používateľ filtrovať podľa názvu informácie, aby používateľ získal konkrétnu informáciu o objekte.
6. Používateľ je schopný vyhľadať skupinu objektov, ktoré spĺňajú všetky parametre zadané používateľom, aby si používateľ mohol vyhľadať skupinu objektov na základe svojich preferencií, ktoré si môže následne uložiť do svojej kolekcie.
7. Používateľ je schopný v procese vyhľadávania objektov zadať parameter definujúci triedu, kategóriu, ktorá obmedzí výsledky vyhľadávania, pretože používateľ chce vyhľadávať objekty, ktoré definuje táto trieda. Triedou môže byt napríklad hrad, mesto alebo múzeum. Trieda definuje, že vyhľadané objektu musia byť hrady, mesta alebo múzea.
8. Používateľ potrebuje od aplikácie možnosť vyhľadať možné a rozumné triedy definujúce objekty podľa názvu triedy, aby sa následne trieda dala použiť ako parameter.
9. Používateľ je schopný v procese vyhľadávania objektov definovať respektíve vymedziť územie, ktoré obmedzí vyhľadávanie, pretože používateľ chce vyhľadávať objekty nachádzajúce sa na konkrétnom území.
10. Používateľ je schopný vyhľadať podľa názvu a definovať parameter územia ako súčasne existujúce územie, aby používateľ vedel vyhľadávať objekty na území kontinentu, krajiny alebo administratívneho územia.
11. Používateľ je schopný definovať parameter územia ako kruhovú oblasť, ktorú používateľ definuje nastavením súradníc bodu stredu kruhu a nastavaním polomeru kruhu, aby vedel používateľ vyhľadávať objekty v okolí zvoleného miesta.
12. Pre spomínanú definíciu kruhu, v ktorom sa budú objekty vyhľadávať, používateľ je schopný stred tohto kruhu nastaviť na mape posúvaním bodu,

alebo vyhľadaním konkrétneho miesta pomocou vyhľadávania. V takom prípade sa stred kruhu presunie na súradnice nájdeného a vybratého objektu. Pretože používateľ môže chcieť vyhľadávať objekty v blízkom okolí okolo konkrétneho miesta.

13. Používateľ je schopný v procese vyhľadávania objektov definovať nejaké vlastnosti objektu, ktoré musia nájdene objekty splňovať, aby používateľ mohol zadať vlastnosti, ktoré očakáva, že nájdené objekty budú splňovať.
14. Používateľ potrebuje od aplikácie aby mu poskytla zoznam možných vlastností, ktoré vie definovať v procese vyhľadávania objektov, aby používateľ vedel aké vlastnosti je schopný definovať.
15. Používateľ potrebuje od aplikácie prostriedky pre zadávanie konkrétnych hodnôt vlastnosti, aby používateľ vedel tieto vlastnosti definovať.

Funkčne požiadavky pre spravovanie mapových objektov:

1. Používateľ je schopný vytvoriť prázdnú kolekciu neobsahujúcu na začiatku žiadne objekty, aby používateľ vedel do nej priradzovať objekty.
2. Používateľ je schopný si nájdený objekt uložiť do niektoréj z existujúcich kolekcii, pretože sa používateľ rozhodol tento objekt začleniť do svojej zbierky.
3. Používateľ je schopný editovať názov kolekcie, pretože sa používateľ rozhodol zmeniť jej názov.
4. Používateľ je schopný odstrániť kolekciu, ktorá pri zmazaní zmaže aj svoj obsah, teda všetky objekty, ktoré obsahovala, pretože používateľ sa rozhodol, že už ďalej nechce spravovať tuto kolekciu a objekty vnej.
5. Používateľ je schopný editovať názov uloženého objektu. Názov nemusí byt unikátny.
6. Používateľ je schopný odstrániť konkrétny objekt z kolekcie, pretože už ho nechce mať nadalej uložený.
7. Používateľ je schopný napísat a uložiť pre každý uložený objekt textové poznámky, ktoré aplikácia umožní stále editovať, aby si používateľ mohol zaznačiť nejaké poznámky pre konkrétny objekt.
8. Používateľ je schopný nastaviť ikonu objektu na mape pre každý uložený objekt, z pred definovateľného výberu možných obrázkov ikoniek, aby vedel požívateľ z podhľadu na mapu určiť kategóriu objektu na základe tejto ikonky.
9. Používateľ potrebuje od aplikácie aby mu umožnila vidieť jeho zoznam kolekcií, aby ich vedel spravovať.
10. Používateľ potrebuje od aplikácie aby pre vyberanú kolekciu sa na mape zobrazili body s ikonkou reprezentujúce všetky uložené objekty v tejto kolekcií, aby si vedel používateľ prezerať a spravovať svoje objekty.

11. Používateľ potrebuje aby mu aplikácia umožnila nastaviť pre konkrétny uložený objekt informáciu o návštevnosti, aby si mohol používateľ označiť, že daný objekt už navštívil.
12. Používateľ je schopný zadať pri nastavovaní návštevnosti aj dátum návštevy, ktorý môže byť vyjadrený ako konkrétny dátum, mesiac v roku alebo rok. Zároveň je používateľ schopný vyjadriť dátum návštevy ako rozsah časového obdobia, od kedy , do kedy a to ako dátumy alebo mesiace v roku. Pretože používateľ si chce zaznamenať aj dátum návštevy objektu alebo časové obdobie návštevy.
13. Používateľ potrebuje aby mu aplikácia prezentovala informáciu o navštívení objektu, aby používateľ vedel či tento objekt už navštívil.
14. Používateľ je schopný zlúčiť dve kolekcie do jednej, výberom jednej kolekcie, ktorá sa vymaže a jej obsah bude prenesený do druhej vybranej kolekcie, aby mohol používateľ rýchlo zlúčiť obsah dvoch kolekcií.
15. Používateľ je schopný nastaviť aby všetky uložené objekty v danej kolekcie mali rovnaký obrázok ikonky zobrazení na mape, pretože používateľ nechce pre každý objekt v kolekcii samostatne nastavovať ikonku, ak chce aby všetky objekty v danej kolekcii mali rovnakú ikonku.
16. Používateľ môže kliknúť na niektorú ikonu reprezentujúcu uložený objekt. Po kliknutí sa objaví obdĺžnik s informáciami o danom objekte.
17. Používateľ potrebuje od aplikácie aby mu v zozname kolekcii, prezentovala informáciu kolko každá kolekcia obsahuje celkovo objektov, a kolko z týchto objektov sú označene používateľom ako navštívene, aby používateľ mal informácie o svojich kolekciách.
18. Používateľ je schopný nájdene objekty v procese vyhľadávania objektov uložiť do niekorej zo svojich kolekcií, aby používateľ následne nemusel samostatne vyhľadávať objekty, ktoré si chce uložiť do kolekcie.
19. Používateľ potrebuje od aplikácie aby mu nájdene objekty v procese vyhľadávania objektov prezentovala buď v tabuľke alebo na mape, kde každý výsledok je zaznačený bodom na mape, aby mal používateľ prehľad o nájdenej skupine objektov.
20. Používateľ je schopný filtrovať podľa mena v skupinke nájdených objektov, pretože používateľ chce zistiť či sa v tejto skupine nachádza konkrétny objekt.
21. Používateľ je schopný zmeniť meno objektu alebo niektorý nájdený objekt odstrániť zo zoznamu výsledkov pred uložením do kolekcie, pretože používateľ chce niektoré objekty pred uložením editovať.
22. Používateľ potrebuje od aplikácie aby mu vedela vrátiť späť vykonanú akciu pri editovaní a mazaní nájdených výsledkov.

1.2.2 Funkčne požiadavky ako use case scenáre

V tejto pod-sekcii sú vyjadrené funkčne požiadavky ako use cases scenáre. Z dôvodu veľkého množstva funkčných požiadaviek sú ako use case scenáre popísane iba niektoré vybraté funkčne požiadavky, pre ilustrovanie toho ako by vyzerala funkčná požiadavka vyjadrená ako use case scenár.

Use case scenár je štruktúrovaný opis interakcie, medzi hercami (actors) a systémom, definujúcich správanie. V našej práci hercom je používateľ aplikácie. Každý use case reprezentuje správanie, ktoré môžeme špecifikovať ako use case scenár.

Tento štruktúrovaný opis ma nasledujúcu formu:

1. situácia na začiatku - opis stavu systému v akom je pred začiatkom interakcie
2. normálna interakcia – normálna interakcia medzi používateľom a systémom
3. alternatívna interakcia - uvádzajú sa alternatívne scenáre ak sa niečo počalo počas normálneho interagovania
4. stav systému po skončení - opis stavu systému v akom sa systém nachádza po interakcii

Výpis niektorých vybratých funkčných požiadaviek ako use case scenáre:

- Používateľ je schopný vyhľadať konkrétny mapový objekt podľa zadania názvu objektu, pretože si ho môže chcieť uložiť do svojej kolekcie.
 - Situácia na začiatku:
 1. Používateľ má otvorenú webovú stránku.
 - Normálna interakcia:
 1. Používateľ klikne v menu na tlačidlo "Add collectible".
 2. Používateľ si otvorí bočné menu.
 3. Používateľ vyplní časť alebo celý názov objektu do vyhľadávacieho pola.
 4. Používateľ si vyberie a klikne na objekt z ponúknutého výberu nájdených objektov.
 - Alternatívna interakcia:
 1. Zadaný názov neodpovedá žiadnemu objektu. V takomto prípade aplikácia neposkytne žiadnený vyber objektov.
 2. Služba poskytujúca toto vyhľadávanie z nejakého dôvodu nefunguje. V takom prípade aplikácia upozorní používateľa, že vyhľadávanie zlyhalo z dôvodu nefunkčnosti služby.
 - Stav systému po skončení:
 1. Vybratý objekt je zabránený na mape a sú k nemu poskytnuté informácie.
 2. Vybratý objekt je pripravený na možnosť uloženia do niektornej z kolekcií.

- Používateľ je schopný požiadať aplikáciu aby mu pre vybraný objekt vyhľadala odkaz na článok, sídliaci na Wikipédii, ak článok existuje, aby používateľ mal možnosť sa o objekte dozvedieť viac informácií vo forme článku.
 - Situácia na začiatku:
 1. Používateľ ma konkrétny objekt uložený v niektornej kolekcii.
 2. Požívateľ má otvorenú webovú stránku.
 - Normálna interakcia:
 1. Používateľ klikne v menu na tlačidlo "Home", alebo už je stránka v danom stave.
 2. Používateľ si otvorí bočné menu.
 3. Používateľ vyberie kolekciu zo zoznamu svojich kolekcii.
 4. Aplikácia zobrazí objekty na mape.
 5. Používateľ klikne na konkrétny objekt na mape.
 6. Aplikácia zobrazí UI pre mapový objekt.
 7. Používateľ klikne na tlačidlo "Show Details".
 8. Aplikácia získava detaily o objekte, poslaním POST metódy s parametrami identifikujúcimi objekt na server.
 9. Aplikácia zobrazí získané detaily, medzi ktorými je aj odkaz na článok sídliaci na Wikipédii.
 10. Používateľ klikne na odkaz.
 - Alternatívna interakcia:
 1. Vybraný objekt nemá článok na Wikipédii. V takomto prípade sa v UI neobjaví odkaz na článok v zozname detailov.
 - Stav systému po skončení:
 1. Používateľ má otvorené nové okno kde je článok objektu.
 2. Aplikácia ukazuje nadalej detaily objektu.
- Používateľ je schopný vytvoriť prázdnú kolekciu neobsahujúcu na začiatku žiadne objekty, aby používateľ vedel do nej priradzovať objekty.
 - Situácia na začiatku:
 1. Používateľ ma otvorenú stránku, kde je aplikácia v stave ukladania objektov do kolekcie.
 2. V aplikácii sú vybraté nejaké mapové objekty.
 - Normálna interakcia:
 1. Používateľ klikne na tlačidlo "Create a new Collection".
 2. Aplikácia zobrazí nové UI.
 3. Používateľ zadá do pola názov kolekcie, ktorú chce vytvoriť.
 4. Používateľ klikne na tlačidlo "Save"
 5. Aplikácia pošle na server príkaz na vytvorenie novej kolekcie.
 - Alternatívna interakcia:

1. Zadaný názov kolekcie ma menej ako tri znaky. V takomto prípade aplikácia neumožní používateľovi kliknúť na tlačidlo "Save" pre vytvorenie novej kolekcie a upozorní používateľa o probléme.
 2. Zadaný názov už je zabratý niektorou existujúcou kolekciou. V takomto prípade aplikácia neumožní používateľovi kliknúť na tlačidlo "Save" pre vytvorenie novej kolekcie a upozorní používateľa o probléme.
- Stav systému po skončení:
1. V systéme je uložená nová prázdna kolekcia.
- Používateľ je schopný odstrániť kolekciu, ktorá pri zmazaní zmaže aj svoj obsah, teda všetky objekty, ktoré obsahovala, pretože používateľ sa rozhodol, že už ďalej nechce spravovať tuto kolekciu a objekty v nej.
- Situácia na začiatku:
1. Používateľ ma otvorenú stránku.
 2. V systéme je uložená aspoň jedna kolekcia.
- Normálna interakcia:
1. Používateľ klikne na tlačidlo "Edit Collections".
 2. Aplikácia zobrazí editačnú tabuľku s kolekciami používateľa.
 3. Používateľ nájde alebo vo vyhľadávacom poli zadá názov kolekcie a vyberie riadok s konkrétnou kolekciou.
 4. Používateľ klikne na tlačidlo "Remove" v danom riadku.
 5. Aplikácia pošle na server príkaz na vymazanie danej kolekcie.
- Alternatívna interakcia:
1. Server nevykoná príkaz, alebo nie je funkčný. V takom prípade sa kolekcia nevymaže a používateľ ju má stále uloženú v systéme.
- Stav systému po skončení:
1. Kolekcia je úspešne vymazaná zo systému.
 2. Všetky objekty, ktoré boli priradené k vymazanej kolekcii sú tiež všetky úspešne vymazané zo systému.
- Používateľ potrebuje aby mu aplikácia umožnila nastaviť pre konkrétny uložený objekt informáciu o návštěvnosti, aby si mohol používateľ označiť, že daný objekt už navštívil.
- Situácia na začiatku:
1. V systéme je uložená aspoň jedna kolekcia s aspoň jedným objektom, ktorý do nej patrí.
 2. Používateľ ma otvorenú stránku.
- Normálna interakcia:
1. Používateľ klikne v menu na tlačidlo "Home".
 2. Používateľ si otvorí bočné menu.
 3. Používateľ vyberie kolekciu zo zoznamu svojich kolekcii.

4. Aplikácia zobrazí objekty na mape.
 5. Používateľ klikne na konkrétny objekt na mape.
 6. Aplikácia zobrazí UI pre mapový objekt.
 7. Používateľ klikne na tlačidlo "Visitation"
 8. Aplikácia zobrazí UI so zaškrťvacím okienkom na označenie návštevy.
 9. Používateľ zaznačí zaškrťávacie okienko.
 10. Používateľ klikne na tlačidlo "Save".
 11. Aplikácia pošle na server príkaz potrebnými parametrami pre nastavenie návštevnosti objektu.
 12. Aplikácia zobrazí používateľovi, že daný objekt bol navštívený.
- Alternatívna interakcia:
1. Server nevykoná príkaz, alebo je v nefunkčnom stave. V takom prípade používateľ uvidí v aplikácii, že daný objekt navštívil, ale v systéme je zaznačene, že objekt je ešte nenavštívený. Po znova načítaní stránky používateľ uvidí, že objekt nie je navštívený.
- Stav systému po skončení:
1. Objekt ma v systéme uložene dátá o tom, že bol používateľom navštívený.

1.2.3 Zamietnuté funkčne požiadavky

Nasleduje výpis funkčných požiadaviek, ktoré boli zamietnuté na základe dvoch faktorov.

1. Čas. Z dôvodu časovej tiesne boli niektoré požiadavky zamietnuté. Tieto požiadavky by ale mohli byť v budúcnosti implementované.
2. Druhy faktor je zamietnutie po analýze dátových zdrojov a možností, po ktorých sme došli k záveru, že ich nevieme rozumne implementovať. Tým nehovoríme, že by sa nedali implementovať.

Zamietnuté funkčne požiadavky:

1. Používateľ je schopný ako oblasť, v ktorej sa bude skupina objektov vyhľadávať, definovať aj nie súčasne existujúce územia. Jedna sa napríklad o štáty, ktoré v dnešnej dobe neexistujú.
2. Aplikácia dôkaze mapový objekt reprezentovať aj ako územnú oblasť definované množinou bodov na mape. Teda aplikácia dôkaze vykresliť na mape aj polygón definujúci oblasť objektu. Vhodné na zaznačovanie navštívených oblastí ako napríklad štáty sveta.

1.3 Nefunkčné požiadavky

Nasledujúce požiadavky môžeme rozdeliť do kategórii podľa zamerania ako napríklad výkon aplikácie, spoľahlivosť dát alebo rozšíriteľnosť. Týchto rôznych kategórii existuje obrovské množstvo a preto sú uvedene iba niektoré konkrétnie, ktoré sa tykajú tejto aplikácie.

1. Aplikácia je dostatočne vykoná. Vyhľadávanie dát na základe zadania názvu v niektorom z vyhľadávačov. netrvá viac ako sekundu.
2. Aplikácia je spoľahlivá. Vyhľadávanie vo konkrétnych vyhľadávačov poskytuje rozumne výsledky, teda výsledky ktoré sú spojene s tým čo je vyhľadávané.
3. Vyhladievanie skupiny objektov na zakladá vstupných parametrov skončí do minúty. Ak vyhľadávanie neskončí do tohto času, tak aplikácia vyhlásí vyhľadávanie ako neúspešne z dôvodu zadania náročných a nevhodných parametrov pre vyhľadávanie.
4. V prípade nefunkčnosti vyhľadávania aplikácia sa správa rozumne, teda poskytne informáciu používateľovi o naskytnutom probléme.
5. Pri zmene dát uloženého mapového objektu, aplikácia poskytne používateľovi správu o tom, či zmena bola úspešne uložená do systému. Napríklad po editovaní poznamok alebo zadania návštevnosti.
6. Pre pomenovanie názvu kolekcie, ktoré už iná kolekcia používa, aplikácia neumožní vytvoriť novu kolekciu a nahlásí používateľovi informáciu o tom, že sa snaží použiť už zabratý názov.
7. Aplikácia je navrhnutá tak aby ju používateľ intuitívne vedel ovládať.
8. Aplikácia je implementovaná ako single-page webová stránka.
9. Aplikácia je rozdelená na časti backend s databázou a frontend, ktoré medzi sebou komunikuje cez API.
10. Frontend je implementovaný v súčasnosti moderným frameworkom.
11. Backend je implementovaný jednoduchým frameworkom, ktorý HTTP dotazy smeruje na jednotlivé funkcie.
12. Používateľské rozhranie aplikácie je responzívne pre desktopové a mobilné zariadenia.
13. Vizuál stránky, tým je myslené ako stránka vyzerá, je jednoduchý z dôvodu, že hlavný zámer stránky je funkciu a nie vizuálny vzhľad.
14. Aplikácia využíva Wikidata Query service pre získavanie všetkých potrebných dát.
15. Aplikácia je určite kompatibilná s prehľadávačom Google Chrome.

16. Dáta si aplikácia ukladá do databázy, ktorá je vhodná a jednoduchá na implementovanie.
17. Zdrojový kód aplikácie je prehľadný. Objekty, metódy a funkcie sú zdokumentovane.
18. Komponenty aplikácie sú navrhнутé tak aby sa dali recyklovať a aby nedokázalo k duplikovaniu kódu.

2. Mapové webové aplikácie

V tejto kapitole si prejdeme konkrétnie vybrané webové aplikácie, ktoré pracujú s mapovými dátami a poskytujú používateľom určitú funkciu. Pre každú vybranú webovú aplikáciu popíšeme funkcie z pohľadu ponúknutia rôznych druhov máp, vyhľadávania objektu a skupiny objektov na základe zadaných parametrov, poskytovania informácií o objekte a spravovania mapových objektov. Spomínane funkcionality si popíšeme, následne porovnáme a uvedieme, v akých funkciách sa bude naša aplikácia odlišovať od ostatných analyzovaných aplikácií. Pojmom odlišovať myslime aké funkcie budú v našej aplikácii chýbať, ktoré budú upravene a na druhu stranu aké funkcie budú rozširovať funkciu našej aplikácie.

Webových aplikácií zaobrájúcich sa mapovými dátami existuje veľké množstvo. Z tohto dôvodu vyberieme iba zopár tých najznámejších na zakladá našich vlastných poznatkov a skúsenosti s nimi. Vybrane webové mapové aplikácie sú Google Maps, Mapy.cz a OpenStreetMap.

2.1 Google Maps

Google Maps je online webová aplikácia zaobrájúca sa prácou s mapami. Túto webovú aplikáciu vlastní a poskytuje verejnosti spoločnosť Google. Podľa metriky k roku 2020 používa mesačne Google Maps služby vyše než jedna miliarda aktívnych užívateľov.(1) Aplikácia ponúka rožné druhy máp ako satelitnú mapu, Street view, mapu zobrazujúcu cestnú premávkou, terénnu mapu a tak ďalej. Naša aplikácia bude poskytovať používateľovi iba jeden druh mapy a to klasicky.

Aplikácia ponúka používateľovi vyhľadávať mapové objekty na základe zadania klúčovej hodnoty názvu objektu. Zároveň vyhľadávanie podporuje možnosť zadáť názov kategórie, ako napríklad "hrady" alebo "múzea", a následne vyhľadávať skupinu objektov, ktorá spadá do tejto kategórie. Vyhľadávanie skupiny objektov podľa kategórie je obmedzené iba na vyhľadávanie objektov v okolí konkrétneho miesta. Respektíve v obdĺžnikovej oblasti, ktorá sa používateľovi zobrazuje na mape. Ak má používateľ otvorenú mapu, ktorá zobrazuje na mape mesto Praha a používateľ uvedie do vyhľadávania kľúč "múzea", Aplikácia vyhľadá múzea iba v meste Praha. Keď používateľ ma dostatočne oddialenú mapu tak aby mu mapa ukazovala celý svet a následne uvedie do vyhľadávania názov kategórie a potvrď vyhľadávanie, aplikácia nájde nejaké výskytu objektov v danej oblasti. Nájdená skupina objektov je samozrejme neúplná. To znamená, že neobsahuje všetky objekty v hľadanej oblasti. Kým na malom území vyhľadávanie nájde všetky objekty patriace ku hľadanej kategórii, tak čím väčšia oblasť tak tým sú výskoty viacej rozptýlene, teda vyhľadávanie nájde iba niektoré objekty. Od našej aplikácie očakávame, že umožní vyhľadávať všetky objekty na území, ktoré používateľ definuje. Zatial čo Google Maps vyhľadávajú iba v zobrazenej oblasti, tak naša aplikácia umožní si vybrať a definovať spôsob vymedzenia oblasti vyhľadávania. Jedná sa o možnosť vybrať a definovať územie ako administratívnu oblasť, krajinu, región alebo vyhľadávať objekty na celom svete.

Vyhľadávanie podľa kategórie nepodoprú definovať výnimky vo vyhľadávaní. Napríklad pre hodnotu "hrad" nám aplikácia nájde hrady, ale aj objekty zrúcanín

hradov. Ak používateľ zadá ako kľúč "zrúcanina hradu" tak mu aplikácia vyhľadá iba zrúcaniny hradov. Možnosť vyhľadať skupinu objektov reprezentujúcu hrady ale vylúčiť z nej objekty zrúcanín hradov Google Maps nepodporujú. Od našej aplikácie túto voliteľnú možnosť budeme očakávať. Inými slovami používateľ bude schopný pre vybranú kategóriu zvoliť pod-kategóriu, ktorá obmedzí vyhľadávanie.

Vyhľadávanie taktiež neumožní používateľovi definovať a zadat nejaké vlastnosti hľadaných objektov, ktoré by nájdene objekty museli splňovať. Uvedme ako príklady zopár rozumných vlastností. Napríklad vlastnosť "nadmorská výška", kde používateľ uvedie hodnotu v niektornej z rozumných jednotkách a vyberie operátor porovnávania. Nájdene objekty musia na základe operátora porovnania splňať nadmorskú výšku. Napríklad musia byť položene vo viac ako 500 metrov nad morom. Vlastnosť "architekt budovy", kde používateľ potrebuje vyhľadať architekta a vyhľadávanie nájde všetky budovy, ktoré uvedený architekt navrhhol. Posledná uvedená vlastnosť ako príklad je "čas vzniku", kde by používateľ uviedol časový udaj ako napríklad storočie a aplikácia by mu vyhľadala objekty, ktoré vznikli respektíve boli postavené v tomto storočí. Týchto vlastností existuje obrovské množstvo a od našej aplikácie očakávame, že umožní vyhľadávať aj s uvedenými obmedzeniami definovanými ako spomínané vlastnosti.

Google Maps poskytujú používateľovi rôzne informácie o objekte na základe, kategórie čo daný objekt reprezentuje. Napríklad pre objekty, ktoré sú mesta aplikácia poskytne informácie ako lokálny čas v danom meste a aktuálne počasie. Pre objekty reštaurácie poskytne informácie ako hodnotenie reštaurácie a recenzie na danú reštauráciu od používateľov. Ako posledný príklad uvedieme hotely, kde aplikácia získa informáciu o cenne hotela za noc a ponúkne odkazy na stránky, kde používateľ je schopný si objednať noc v hoteli. Takéto informácie podľa kategórie od našej aplikácie nebudeme očakávať, z dôvodu potreby získavania informácií z rôznych iných služieb. Naša aplikácia bude využívať iba informácie, ktoré poskytujúcu Wikidata.

Pre všetky objekty služba poskytne obrázok objektu, a základne informácie ako adresu, kde sa konkrétny objekt nachádza, webovú stránku objektu, telefónne číslo, otváracie hodiny a recenzie používateľov. Uvedene informácie sú statické. Iné informácie ako nadmorská výška a podobne služba používateľovi neposkytne. Naša aplikácia síce nebude podporovať poskytovanie používateľských recenzii, ale pokúsime sa aby poskytovala rôzne informácie, ktoré Wikidata ponúkajú. Napríklad pre mestá nám naša aplikácia získa aj údaj o počte obyvateľstva.

Mapové objekty Google maps umožnia používateľovi si pridať objekt do listu, kde bude objekt uložený. Pre každý uložený objekt je používateľ schopný napísať poznámky k objektu, ktoré vie vždy editovať. Google Maps poskytujú možnosť zaznamenávania histórie návštevnosti. To funguje nasledovným spôsobom. Používateľ povolí aby Google Maps mohli používať jeho GPS dát a z toho automaticky dôkaze aplikácia usúdiť, že používateľ navštívil konkrétny objekt. Od naša aplikácia očakávame podobnú možnosť zaznamenávania návštevnosti, ale bez automatického zaznamenávania návštevy objektov. Používateľ našej aplikácie bude musieť návštevnosť manuálne nastaviť. Naša aplikácia bude zároveň podporovať to, že ikonky na mape reprezentujúce objekty, ktoré ešte neboli navštívene budú bez farby, teda iba čisto sivé, a navštívene budú obsahovať plnú farbu ikonky obrázka. To používateľovi umožní odlišovať navštívene objekty od zatiaľ nena-vstívených.

Google Maps poskytujú aj možnosť vytvorenia a editovania vlastnej mapy, kde používateľ je schopný uložiť mapové body. Týmto bodom je schopný nastaviť farbu a obrázok ikonky z ponúknutého výberu, napísať poznámky k objektu a pridať k objektu obrázok alebo video. Naša aplikácia z tohto bude podporovať vyber obrázka ikonky ale bez farby, pretože farba reprezentuje v našej aplikácii, či bol konkrétny objekt navštívený. Zároveň aplikácia umožní používateľovi vytvoriť poznámky pre objekt.

Google Maps je webová aplikácia, ktorá umožňuje široké spektrum funkcionality a poskytovanie zaujímavých informácií o objekte. V tomto aspekte bude naša aplikácia o niečo horšia. Na druhu stranu našu aplikáciu obohatíme o precíznejšie vyhľadávanie objektov na základe zadaných parametrov ako definovanie kategórie a pod-kategórie objektu, územie a vlastnosti, ktoré musí nájdený objekt spĺňať.

<https://sites.google.com/a/pressatgoogle.com/google-maps-for-iphone/google-maps-metrics>

2.2 Mapy.cz

Mapy.cz je webová aplikácia od českej spoločnosti Seznam.cz, ktorá ponuka nielen mapu českej republiky, ale aj mapu celého sveta vďaka spolupráci s OpenStreetMap.

Aplikácia ponúka používateľovi na vyber rôzne náhľady máp ako základnú, dopravnú s dopravnými informáciami, turistickú, leteckú, inak povedané satelitnú a ešte zopár ako na ako napríklad fotografickú. Fotografická mapa zobrazuje na mape maličké fotky objektov na miestach kde sa konkrétny objekt nachádza. Aplikácia ponuka netradičný náhľad ako na mapu z 19. storočia. Tento náhľad však funguje iba pre českú republiku. Ako už bolo spomenuté naša aplikácia bude používať iba jeden náhľad a to základný.

Používateľ je schopný na Mapy.cz vyhľadávať objekty na základe zadania názvu alebo časti názvu objektu ale aj na základe zadania názvu kategórie podobne ako u aplikácie Google Maps.

Mapy.cz nájdu skupinu objektov v obdĺžnikovej oblasti ktorá sa pravé zobrazuje používateľovi na mape. Rovnako ako u Goole Maps čím je zobrazená oblasť väčšia je tým nájdene objekty sú viac rozptylé. Teda aplikácia nájde iba niektoré objekty z každej časti mapy. Aplikácia nenájde všetky objekty patriace do kategórie. Keď si používateľ priblíži na menšiu oblasť a znova zadá vyhľadávanie, tak aplikácia nájde viac objektov v danej oblasti. Aby aplikácia používateľovi vyhľadala všetky objekty v danej kategórii musí sa vyhľadávať na malom území. Mapy.cz neumožňujú vyhľadávať v oblasti ktorú si definuje používateľ. Jedná sa o rovnaký problém ako u Google Maps, ktorý v našej aplikácii budeme riešiť možnosťou definovať oblasť vyhľadávania.

Zároveň ani Mapy.cz neposkytujú možnosť definovať pod-kategóriu, ktorá obmedzuje vyhľadávanie aby sa objekty tejto pod-kategórii nevyhľadávali.

Možnosť obmedziť vyhľadávanie skupiny mapových objektov, tým že používateľ definuje vlastnosti, ktoré musia nájdene objekty, splňať podobne ako Google Maps aj aplikácia Mapy.cz tuto možnosť nepodporuje.

Pre vyhľadané alebo zvolené objekty Mapy.cz poskytujú používateľovi informácie ako obrázok objektu, popis objektu získaní z Wikipédií a informáciu o počasí podobne ako Google Maps, ale oproti Google Maps Mapy.cz poskytujú aj

detailnejšie informácie na základe toho čo sa dá o danom objekte povedať. Ako príklad si uvedieme, že pre objekt kategórii mesto Mapy.cz poskytli informáciu o populácii a nadmorskej výške. Tieto informácie Google Maps neposkytujú pre mesto.

Týchto informácií o objekte nebolo obrovské množstvovo a používateľ v nich nenájde všetky možne informácie. Od našej aplikácie budeme očakávať, že pre objekt získa viac informácií z Wikidata. A to v podobe vypísania vlastnosti objektu a uvedenia ich hodnôt. Pre objekt nebudeme staticky definovať, ktoré vlastnosti to budú. Aplikácia poskytne všetky rozumne vlastnosti, ktoré sa budú dať získať z Wikidata.

Mapy.cz taktiež poskytujú možnosť si ukladať a spravovať mapové objekty.

2.3 OpenStreetMap

OpenStreetMap poskytujú mapové dátá pre webové a mobilné aplikácie. Dátá na OpenStreetMap sú otvorené. To znamená, že sa môžu verejne šíriť a upravovať.

Aj služba OpenStreetMap ponúka používateľovi viaceré druhy máp. Sú to štandardná, cyklistická mapa, ktorá poskytuje a zobrazuje používateľovi dátá o cyklistických cestách formou farebného vyznačenia ciest, dopravná alebo humanitárna mapa. Nevýhodou pre používateľa oproti spomínaným aplikáciám je absencia satelitnej mapy.

OpenStreetMap poskytujú zaujímavú funkciu "Prieskum prvkov". Tato funkcia vyhľadá okolité objekty okolo bodu, na ktorý používateľ klikne na mape. Táto oblasť je ale veľmi mala. Vyhľadávanie nájde rôzne objekty ako aj cesty, chodníky alebo dokonca stromy. Aplikácia poskytuje ku každému objektu informácie, ktoré opisujú a charakterizujú konkrétniejsie objekt.

Vyhľadávanie objektu alebo objektov je na OpenStreetMap odlišne od Mapy.cz a Google Maps. Nájdene výsledky sa nezobrazia rovno na mape. Zobrazia sa iba ako zoznam objektov. Bod na mape sa objaví iba jeden, ktorý reprezentuje ten objekt, na ktorý používateľ klikol v zozname nájdených objektov. Pre každý nájdený objekt aplikácia poskytne názov kategórie, pod ktorú objekt najviac spadá.

Aplikácia neumožňuje používateľovi vyhľadávať objekty podľa kategórie. Táto funkciu tu značne chyba oproti spomínaným dvom aplikáciám.

Na rozdiel od Google Maps a Mapy.cz sa OpenStreetMap líšia hlavné neposkytovaním funkcionality na manažovanie objektov. OpenStreetMap sa zameriava skôr na poskytovanie mapových dát. V OpenStreetMap si používateľ nevie uložiť objekt do listu a následne k objektu si napísat poznámky ,alebo pridať fotku.

OpenStreetMap sú v tejto trojici najmenej používateľsky vhodné a neposkytujú takú funkciu ako zvyšné dve aplikácie. Najsilnejšou stránkou je ale možnosť voľného používania a šírenia dát. Naša aplikácia podobne ako Mapy.cz bude OpenStreetMap využívať a to v podobe používania štandardnej mapy.

3. Analýza

3.1 The Resource Description Framework

Na internete nájdeme obrovské množstvo informácií o zdrojoch. Zdrojom môže byť čokolvek. Napríklad dokument, človek alebo hmatateľný objekt. Informácie o zdrojoch je potrebné spracovať rôznymi aplikáciami, nie len umožniť človeku tieto informácie čítať. Na to bol úmyselne navrhnutý The Resource Description Framework.

The Resource Description Framework (v skratke označovaný ako RDF) je framework, ktorého zámer je vyjadriť informácie o zdrojoch. RDF poskytuje bežné obecne známy prístup ako vyjadrovať informácie. Vďaka tomu si dokážu rôzne aplikácie navzájom vymieňať informácie a to bez straty významu informácií.

RDF môže byť použitý na prepájanie dát na internete. Uvedme si jednoduchý príklad. Majme dáta reprezentujúce mesto Praha, v ktorých je zahrnutá informácia, že Praha je hlavným mestom Českej republiky. Tato informácia môže byť vyjadrená ako odkaz na dátu reprezentujúce Českú republiku. Získanie týchto dát nám poskytne viac informácií o Českej republike. Takto prechádzaním na odkazy vedúce k ďalším zdrojom si dokáže človek alebo automaticky proces aplikácie zbierať informácie o rôznych veciach. Takéto využitie sa často kvalifikuje ako Linked Data [LINKED-DATA].

3.1.1 RDF Dátový Model

Pre vyjadrovanie informácií v RDF je potrebné požívať obecný formát, ktorý vedia stroje čítať. V RDF je formát štruktúrovaný ako nasledujúca trojica:

```
<predmet> <predikát> <objekt>
```

Formát vyjadruje vzťah medzi predmetom a objektom. Obe sú zdroje, ktoré nejako spolu súvisia. Predikát v tomto formáte vyjadruje vzťah medzi predmetom a objektom. Tento vzťah je formulovaný smerom od predmetu ku objektu a v RDF sa nazýva ako vlastnosť. Pretože na vyjadrenia v RDF sa používa štruktúra o troch prvkov tak nazývame tieto vyjadrenia "trojica"(anglicky "triple")

Pre lepšiu predstavu uvedme zopár konkrétnych príkladov trojíc:

```
<Karlov most> <je pomenovaný po> <Karol IV>
<Karlov most> <je> <kamenný most>
<kamenný most> <je> <most>
<kamenný most> <je z materiálu> <kameň>
```

Konkrétny zdroj je často odkazovaný vo viacerých trojiciach. Taktiež v jednej trojici môže vystupovať ako predmet a v inej ako objekt. To nám umožní získať prepájanie medzi trojicami čo dáva RDF dôležitú a silnú schopnosť zbierať informácie.

Trojice vieme vizuálne vyjadriť ako spojený graf. Tento graf je tvorený vrcholmi, ktoré reprezentujú predmety a objekty. Hrany medzi vrcholmi reprezentuje predikát.

Kedť sú informácie vyjadrené v spojenom grafe, tak môžeme na nich použiť SPARQL pre získavanie informácií.

3.1.2 Typy RDF dát

V trojici sa vyskytujú tri typy dát: IRI, literál a prázdný vrchol

IRI

Úlohou IRI ("International Resource Identifier") je identifikovať zdroj. Poznáme aj URL (Uniform Resource Locators) na adresovanie webových stránok. URL je jedna z foriem IRI. Ostatné formy IRI identifikujú zdroj bez uvedenia kde je zdroj umiestnený alebo ako k nemu pristupovať. IRI je zovšeobecnením identifikátora URI (Uniform Resource Identifier), ktorý umožňuje použitie znakov iných ako ASCII v refazci znakov IRI.

URI sa môže vyskytovať na všetkých troch pozíciah v trojici.

Literal

Jednoducho povedané literal je hodnota, ktorá nie je URI. Literalom môžu byť textové reťazce, dátumy alebo čísla. S literalom sa viaže aj jeho dátum typ. Ten pomáha pri správnom interpretovaní hodnoty.

Literal sa môže vyskytnúť v trojici iba ako objekt. To nám reprezentuje také hodnoty objektov, ktoré nie sú nijaký iný zdroj, ale iba napríklad číslo, text alebo dátum.

Prázdný vrchol Niekedy je užitočné rozprávať o zdrojoch bez toho aby sme sa zaoberali použitím globálneho identifikátora. Ako príklad si uvedieme obraz "Mona Lízi". Tento obraz má svoj identifikátor, ale na obraze v pozadí môžeme vidieť strom, ktorý nemá svoj identifikátor, ale vieme o aký je to strom. Zdroje bez globálneho identifikátora ako tento strom môžeme v RDF reprezentovať ako prázdný vrchol. Tento prázdný vrchol je ako jednoduchá premenná v algebre. Tá nám predstavuje nejakú vec bez toho aby sme uviedli aká je hodnota tejto veci.

Prázdný vrchol sa môže vyskytovať na pozícii predmetu a objektu.

3.1.3 Viaceré grafy

V praxi keď tvoríme a spravujeme informácie, tak by sa nám zišiel mechanizmus, vďaka ktorému by sme vedeli hovoriť o nejakej podmnožine kolekcie trojíc. Z toho dôvodu RDF poskytuje mechanizmus na zoskupenie vyjadrený RDF do viacerých grafov a priradiť takému grafu IRI. Viaceré grafy sú nedávnym rozšírením RDF dátového modelu.

Viaceré grafy v RDF tvoria RDF dataset, čo je kolekcia RDF grafov. Viaceré grafy boli prvý krát predstavené v RDF dotazovacom jazyku SPARQL.

3.1.4 RDF slovníky

RDF dátový model poskytuje spôsob ako o zdroju uviesť vyjadrenie. Problémom je, že tento model netvrdí nič o tom čo konkrétnu IRI znamenajú. Z tohto dôvodu sa v praxi zvykne RDF kombinovať so slovníkmi alebo inými konvenciami, ktoré poskytnú sémantickú informáciu o zdrojoch.

Pre podporu definovať slovník poskytuje RDF jazyk RDF Schéma. Tento jazyk nám dovoľuje definovať sémanticky význam RDF dát. RDF Schéma je sémantické rozšírenie RDF.

V RDF Schéme nájdeme systém tried a vlastnosti, ktorý je podobný systému aký sa používa v objektovo orientovaných programovacích jazykov. Na rozdiel od iných takýchto systémov, kde sa trieda definuje z hľadiska vlastnosti, sa v RDF Schéme systém odlišuje pravé v tom, že používa pojem triedy na špecifikáciu kategórii, ktoré môžeme použiť na klasifikáciu zdrojov. Vzťah medzi triedou a jej inštanciami je daný prostredníctvom vlastnosti type. Pomocou RDF Schémy vieme vytvárať hierarchiu tried, pod-tried a vlastnosti, pod-vlastnosti. Taktiež je možné definovať typové obmedzenia pre predmety a objekty jednotlivých trojíc prostredníctvom obmedzení domény a zadaniu rozsahu.

Pomocou RDF Schémy vieme zostaviť model údajov RDF.

Pre lepšiu predstavu uvedme jednoduchý neformálny príklad:

```
<Pražský hrad> <typ> <Hrad>
<Hrad> <pod-trieda> <opevnenie>
<nachádza v správnom územnom celku> <typ> <vlastnosť>
<nachádza v správnom územnom celku> <doména> <správny územný celok>
<nachádza v správnom územnom celku> <rozsah> <správny územný celok>
<nachádza v správnom územnom celku> <pod-vlastnosť> <lokácia>
```

3.2 Wikidata

Wikidata sú bezplatná, vedomostná a sekundárna databáza obsahujúca otvorené dátá. Tuto databázu môžu čítať a upravovať ľudia a aj stroje. Pojmom "sekundárna databáza" myslíme, že Wikidata neobsahujú iba čisto natvrdo vyjadrené informácie, ale aj ich zdroje a prepojenia na iné databázy. Z toho plynne rozmanitosť a dostupnosť poznatkov.

Wikidata fungujú ako centrálné úložisko pre štruktúrované údaje pre partnerské projekty ako napríklad Wikipédia.

Wikidata sú taktiež mnoho-jazyčné, teda informácie tu nájdeme v rôznych jazykoch.

Wikidata pozostávajú hlavne z položiek, ktoré sa označujú ako "items". Každá položka má štítok (label), popis (description) a číslo udávajúce počet aliasov.

Štítok je najbežnejšie meno označujúce danú položku, ktorá je dobre známa pod týmto označením. Štítok zároveň nemusí byť jedinečný. Viacero položiek môže mať rovnaké pomenovanie.

Popis na Wikidatach je krátky opis, fráza ktorou úlohou je odlišiť položky s rovnakými alebo podobnými štítkami. Popis nemusí byť jedinečný rovnako ako štítok, ale žiadne dve položky nemôžu mať zároveň rovnaký štítok a aj popis.

Alias je alternatívne meno, alebo označenie položky, pod ktorým je položka taktiež známa. Na Wikidatach môžeme vyhľadávať položky podľa štítku alebo aliasu.

Aby vedeli Wikidata jednoznačne od seba položky jednoznačne odlišiť, tak každá položka má svoj unikátny identifikátor. Tento identifikátor obsahuje na začiatku vždy veľké písmeno Q a za ním nasleduje číslo. Tento identifikátor označme ako "QNumber". Napríklad mesto Praha ma identifikátor Q1085.

Vyjadrenie (anglicky statement) je trojica predmet, predikát a objekt reprezentujúci trojicu z RDF. Statement vyjadruje informácie, zaznamenané na Wiki-

datach, o položke. Statement pozostáva z dvojice vlastnosť (anglicky property) a hodnota (anglicky value).

Každá vlastnosť reprezentuje informáciu a má k sebe priradenú aspoň jednu hodnotu, ale môže nadobúdať aj viacero. Vlastnosti na Wikidatach majú taktiež ako položky svoj identifikátor, ktorý je rovnaký ako u položiek až nahradenie písmena Q za písmeno P. Napríklad vlastnosť "lokácia" má identifikátor P276.

Hodnoty v statement-och sú dátia opisujúce položku. Ako už bolo spomenuté vlastnosť ak je to rozumné môže nadobúdať viacero hodnôt. Napríklad osoba môže mať vlastnosť "deti", kde hodnota je odkaz na položku reprezentujúcu dieťa danej osoby. Osoba ale môže mať viacero detí, a teda je povolené aby vlastnosť mohla mať viacero hodnôt. Ideálne je, keď každá vlastnosť má iba jedno hodnotu. Napríklad vlastnosť "populácia" by mala mať len jednu hodnotu, ktoré udáva počet obyvateľov. Problém ale je, že to tak vždy nie je. Aj takáto vlastnosť môže nadobúdať viaceré hodnoty, z dôvodu, že rôzne zdroje uvádzajú rôzne hodnoty. Preto sa k týmto hodnotám priradzujú ďalšie pridané informácie poskytujúce. Sú to kvalifikácie (anglicky qualifiers), ranky (Ranks) a referencie (references).

Referencie odkazujú na konkrétny zdroj, ktorý poskytol Wikidatamu hodnotu pre daný statement.

Ranky poskytujú mechanizmus anotovať viaceré hodnoty statementu. Na Wikidatach sa používajú tri druhy rank-ov:

1. preferovaný rank - hodnota najlepšie opisujúca danú vlastnosť
2. normalny rank - defaultne nastavený rank
3. zastaraný (deprecated) rank - hodnota s týmto rankom je zastaraná a najskôr neopisuje dobre vlastnosť

Kvalifikácie taktiež ako ranky a referencie umožňujú expandovať, anotovať statement. Statement je iba dvojica vlastnosť a hodnota a pre spresnenie hodnoty, respektíve pre ďalší opis vlastnosti sa používajú kvalifikácie.

Každá vlastnosť nadobúda hodnoty určitého dátového typu. Uvedme niektoré z nich, ktoré sú pre naše využitie najdôležitejšie:

1. Kvantitatívne hodnoty - sú uvedené ako desatinné číslo. Vlastnosti tohto dátového typu by mali obsahovať aj informácie o obmedzeniach ako maximálna a minimálna možná číselná hodnota, ktorú môže číselná hodnota nadobúdať. Keď vlastnosť udáva hodnotu, ktoré môže byť uvedené v nejakej meracej jednotke (anglicky unit), tak vlastnosť by mala obsahovať obmedzenie, ktoré udáva v akých jednotkách sa môže hodnota vyjadriť.
2. Item hodnoty - najbežnejší typ hodnoty na Wikidatach, kde hodnota je referencia na inú položku na Wikidatach. Vlastnosti tohto dátového typu by mali obsahovať obmedzenia, ktoré udávajú aké položky sú priateľné, neprijateľné ako hodnoty v pre danú vlastnosť.
3. Textová hodnota - textový retazec vyjadrený ako string.
4. Časová hodnota - bod v čase. Týmto bodom môže byť napríklad dátum, rok, storočie alebo milénium.
5. Geografické súradnice - dvojica čísel, ktoré vyjadrujú súradnice položky.

Prague Castle (Q193369)

castle complex in Prague, Czech Republic, dating from the 9th century

In more languages

Statements

instance of	castle	edit
▼ 1 reference archINFORM project ID 10786 stated in archINFORM retrieved 31 July 2018		+ add reference
tourist attraction ▼ 0 references		edit
hillfort start time 9. century end time 12. century		edit
+ add value		

Legenda :

- štítok (label)
- popis (description)
- identifikátor (QNumber)
- vlastnosť (property)
- hodnoty (values)
- kvalifikácie (qualifiers)
- referencia (reference)
- rank

Obrázek 3.1: Ukažka Wikidát s popisom

3.3 Wikidata Query Service

Wikidata obsahujú obrovské množstvo dát. Pre získanie konkrétnych dát je potrebne Wikidatam položiť otázku. Jazyk v ktorom sa formulujú otázky na tabázy, respektíve povedané dotazy, je SPARQL. Dotazom rozumieme špeciálnu formu otázky, ktorej stroj rozumie a dokáže na ňu poskytnúť odpoveď.

Wikidata Query Service (WDQS) je softwarový balíček a zároveň verejne dostupná služba navrhnutá na poskytovanie koncového bodu SPARQL, ktorý umožňuje sa pýtať na dátu poskytujúce Wikidatami. WDSQ pracuje na dátach reprezentovaných v RDF formáte.

V nasledujúcich pod-sekciách si ukážeme základy potrebne na formovanie dotazov v jazyku SPARQL.

3.3.1 SPARQL pre Wikidata

Základom je trojica predmet, predikát a objekt, ktoré reprezentujú statement na Wikidatach. Statement "Zem sa nachádza v Mliečnej dráhe" pozostáva z predmetu "Zem"(Q2), predikátu "nachádza sa v"(P276) a objektu "Mliečna dráha (Q321). Tento statement sa dá vyjadriť ako tri URI.

```
<http://www.wikidata.org/entity/Q2> // predmet  
<http://www.wikidata.org/prop/direct/P276> // predikát  
<http://www.wikidata.org/entity/Q321>. // objekt
```

Zakaždým písat všetko ako URI je príliš zdĺhavé, a preto sa zaviedli prefixy. Predmet a predikát musia byť vždy uložené ako URI. Objekt je buď URI alebo literal. Zoberme si "Zem"(Q2), ktorá je uložená ako <http://www.wikidata.org/entity/Q2>. Vďaka prefixom vieme dlhé URI nahradit za krátku formu: wd:Q2.

Statement s použitím prefixov môžeme vyjadriť nasledovne:

```
wd:Q2 wdt:P276 wd:Q321 .
```

Entita (wd:) reprezentuje entitu na Wikidatach, respektíve položku, čo má priradený QNumber. Prefix wdt: ukazuje priamo na objekt. WDSQ rozumie aj ďalším iným prefixom ako napríklad p, ps a pq.

Jednoduchý SPARQL dotaz pre Wikidata, ktorý vráti všetky entity, ktoré sú inštanciou triedy "hrad" môže vyzerať následovne:

```
SELECT ?item  
WHERE  
{  
    ?item wdt:P31 wd:Q23413. # položka musí byť hrad  
}
```

Za SELECT nasledujú výpis premenných, ktoré má dotaz vrátiť ako výsledky. Premenná vždy začína s prefixom ?. WHERE blok obsahuje obmedzenia na dotaz vyjadrené ako trojice. Keď sa spustí dotaz, WDSQ služba do premenných dosadí skutočne hodnoty, podľa trojíc uvedených v dotaze. Vo vyššie uvedenom príklade sa do premennej ?item dosadia všetky položky, ktoré sú inštanciou triedy hradu.

Dáta na Wikidatach sú usporiadane v hierarchii. Položka je inštanciou nejakej inej položky reprezentujúcu triedu. Trieda je pod-trieda inej triedy. Vlastnosť "je inštanciou" označujeme P31 a "je pod-triedou" označujeme P279.

Cesta

Často do premennej dosadzujeme inštancie triedy, ktorá je ale pod-triedou určitej triedy. V takom prípade aby sme nemuseli písat viacero trojíc môžeme predikát vyjadriť ako "cestu" medzi dvoma položkami. Pridať element cesty sa dá pomocou "/". Predikátom vyzerá nasledovne: wdt:P31/wdt:P279

Tato cesta vyjadruje len, že položka je inštanciou triedy, ktorá je pod-triedou inej položky. Cesta sa da predĺžiť. Aby sme ale zbytočne neopakovali elementy v ceste a nemuseli presne špecifikovať dĺžku cesty, tak za element cesty sa dá použiť symbol "*" alebo symbol "+".

Symbol * vyjadruje "nula alebo viac ako tento element" a + vyjadruje "jeden alebo viac ako tento element".

LIMIT a ORDER BY

Za blokom WHERE môžeme pridať:

- LIMIT - vyjadruje maximálny počet vrátených výsledkov. Napríklad LIMIT 10 vráti iba 10 výsledkov, aj keď ich môže byť viacej.
- ORDER BY - zoradí výsledky dotazu podľa výrazu alebo premennej, ktorú vieme zabaliť do ASC() alebo DESC(), čo určí poradie zotriedenia.

OPTIONAL a MINUS

Vrátené výsledky musia splňať všetky trojice. Je užitočné niekedy ale označiť trojicu za voliteľnú. V takom prípade sa vrátia aj výsledky nespĺňajúce danú trojicu. Voliteľné trojice sa píšu do vnútra bloku OPTIONAL .

Na druhú stranu blok MINUS umožňuje vybrať tie výsledky, ktoré nezodpovedajú trojiciam vo vnútri bloku. Používa sa keď je jednoduchšie zadať, že výsledky nemajú splňať trojice vo vnútri bloku ako vytvárať negáciu.

VALUES

V SPARQL dotazov vieme do premennej manuálne priradiť položky. Formát je nasledovný:

```
VALUES ?item { wd:Q2 wd:Q513 }
```

Teraz premenná ?item obsahuje dve položky.

FILTER

Pre filtrovanie položiek na základe výrazu sa používa blok FILTER(), kde do vnútra vložíme výraz. Tento výraz musí vrátiť pravdivostnú hodnotu True alebo False.

Vo výraze sa používajú bežne matematické operátory ako +, -, *, /. Výrazy na porovnávanie <, >, =, >=, <=. Výrazy sa dajú kombinovať použitím logických operátorov.

Taktiež je možné odfiltrovať konkrétné položky.

```
FILTER ( ?item not in ( wd:Q193369 ) ) # odfiltruj Pražský hrad.
```

GROUP BY

Niekedy nechceme aby dotaz vrátil výsledky, ale napríklad počet výsledkov. Na to slúžia agregované funkcie.

Prehľad agregovaných funkcií:

1. COUNT - počet elementov v premennej.
2. SUM - suma všetkých hodnôt v premennej.
3. AVG - priemer všetkých hodnôt v premennej.
4. MIN, MAX - minimum, alebo maximum z hodnôt.
5. SAMPLE - ľubovoľný jeden element z premennej.
6. GROUP CONCAT - zretazenie elementov.

Aby sa v dotaze mohla použiť agregovaná funkcia na premennú, musí byť táto premenná braná ako skupina. To dosiahneme tým, že za blok WHERE sa pridá GROUP BY a nasleduje výpis premenných branných ako skupina.

Kvantifikátory

Prefix wdt: ukazuje priamo na objekt v statement-e. Prefix p: neukazuje priamo na objekt, ale na statement uzol. Tento uzol je následne predmetom v ďalšej trojici. Pre tuto trojicu vieme použiť prefix ps:, ten ukazuje na objekt a prefix pq:, ktorý ukazuje na kvalifikátor.

Príklad k získaniu populácie mesta Praha, zároveň so získaním dátumu namerania populácie vyjadrený ako kvalifikátor hodnoty:

```
SELECT ?population ?date
WHERE
{
    # Praha , populácia , statement uzol
    wd:Q1085 p:P1082 ?populationNode.
    # statement uzol , populácia , premenná pre populáciu
    ?populationNode ps:P1082 ?population.
    # statement uzol , "k dátumu" , premenná pre vlasnosť
    ?populationNode pq:P585 ?date.
}
```

3.3.2 Label service

Label service je užitočná služba pre získavanie štítkov položiek, pretože znížuje náročnosť dotazov, ktoré bez nej by inak boli náročnejšie pre dosiahnutie rovnakého efektu.

WDQS služba podporuje túto službu, ktorá rozširuje štandardné schopnosti SPARQL.

Služba vieme použiť v dvoch režimoch. Automaticky a manuálne.

V automatickom režime stačí do dotazu pridať iba šablónu služby, kde je potrebné uviesť jazyk, ktorý služba bude preferovať. Následne WDQS automaticky vygeneruje štítky na základe:

1. Ak ma neviazaná premenná v SELECT bloku názov “?<názov premennej>Label”, potom do nej WDQS vytvorí štítok (rdfs:label) pre položku v premennej “?<názov premennej>”.
2. Ak ma neviazaná premenná v SELECT bloku názov “?<názov premennej>AltLabel”, potom do nej WDQS vytvorí alias (skos:altLabel) pre položku v premennej “?<názov premennej>”.
3. Ak ma neviazaná premenná v SELECT bloku názov “?<názov premennej>Description”, potom do nej WDQS vytvorí popis (schema:description) pre položku v premennej “?<názov premennej>”.

Je možné uviesť viac jazykov, ktoré sa zvažujú v poradí ako sú uvedene. Za [AUTO LANGUAGE] WDQS automaticky nahradí jazyk momentálneho užívateľského rozhrania.

Príklad automatického módu:

```
SERVICE wikibase:label {  
    bd:serviceParam wikibase:language "en" .  
}
```

Automatický režim kontroluje iba projekciu dopytu. Napríklad pri použití SELECT ?aLabel (SAMPLE(?bLabels) AS ?bLabel) WDQS rozpozná iba prvý štítok a teda v premennej ?bLabel nebude štítok. V takýchto prípadoch je potrebné použiť manuálny režim.

V manuálnom režime je potrebne explicitne naviazať premenné štítkov v rámci servisného volania.

```
SELECT *  
WHERE {  
    SERVICE wikibase:label {  
        bd:serviceParam wikibase:language "en, de" .  
        wd:Q2 rdfs:label ?q2Label .  
        wd:Q2 skos:altLabel ?q2Alt .  
        wd:Q2 schema:description ?q2Desc .  
    }  
}
```

3.3.3 Geopriestorové vyhľadávanie - Around service

Službu umožňujúcu vyhľadávať položky so súradnicami, ktoré sa nachádzajú v určitom okruhu od zadaného stredného bodu.

Prvý riadok around služby musí byť vo formáte: ?item (ľubovoľný názov premennej) wdt:P625 ?location. Nájdene výsledky sa priradia do premennej ?item a do ?location sa priradia súradnice týchto položiek. Predikát wdt:P625 je získanie hodnoty vlastnosti zemepisné súradnice.

Ďalšie parametre sú nasledovne:

- wikibase:center - bod okolo ktorého sa vyhľadáva. Buď sa poskytuje premenná obsahujúca súradnice, alebo literal so súradnicami vo formáte textového reťazca: "Point(<zemepisná šírka> <zemepisná výška>)" eo:wktLiteral
- wikibase:radius - vzdialenosť od bodu v kilometroch, ktorá definuje kruh vyhľadávania.

Nasleduje príklad dotazu, ktorý vyhľadá všetky hrady vzdialene 100 km od mesta Praha.

```
SELECT ?place ?placeLabel ?location ?dist WHERE {  
# Praha súradnice  
wd:Q1085 wdt:P625 ?súradnicePrahy .  
SERVICE wikibase:around {  
?item wdt:P625 ?location .  
bd:serviceParam wikibase:center ?súradnicePrahy .  
bd:serviceParam wikibase:radius "100" .  
}  
# Položka je hrad  
FILTER EXISTS { ?item wdt:P31/wdt:P279* wd:Q23413 }  
SERVICE wikibase:label {  
bd:serviceParam wikibase:language "en" .  
}  
}
```

3.3.4 MediaWiki API Query Service

MediaWiki API Query Service (MWAPI) dovoluje zo SPARQL volať MediaWiki API a vrátiť výsledky naspať do SPARQL. Do dotazu sa služba zavolá použitím bloku SERVICE wikibase:mwapi . Do vnútra bloku sa vkladajú parametre služby. Parametre "wikibase:api" a "wikibase:endpoint" sú povinné parametre. "Endpoint definuju" koncového hostiteľa. "Api" definuje ktorá API bude na koncovom hostiteľovi vykonaná.

Z default nastavený táto služba vracia všetky výsledky. Parametrom "wikibase:limit" sa dá nastaviť počet vrátených výsledkov. Má to rovnaký efekt ako LIMIT, ale ten sa aplikuje, až kde API volanie skončí a preto je vhodnejšie použiť tento parameter.

Ostatné parametre začínajú prefixom "mwapi" ako napríklad mwapi:language "en", ktorý definuje jazyk dotazu.

Pre definovanie výstupných parametrov, ktoré sa vrátia z volania API, to vyzierá nasledovne:

```
?item wikibase:apiOutput mwapi:title .
```

Do premennej “?item” sa dosadí výsledok definovaný výstupnou premennou “title”. Je dôležite zdôrazniť, že použite výstupne parametre očakávajú prítomnosť určitých vstupných parametrov.

Predikát “wikibase:apiOrdinal”, ktorý berie ako objekt pravdivostnú hodnotu, dovoľuje definovať výsledkom čísla poradia, ktoré sa nastavia podľa poradia aké originál API vráti.

Ukážme si jednu konkrétnu podporovanú službu, ktorou je “EntitySearch”. Táto služba vyhľadáva Wikibase položky na základe názvu.

Vstupne parametre pre túto službu sú:

- search - zadanie názvu, podľa ktorého sa vyhľadáva
- language – jazyk použitý vo vyhľadávaný
- limit - maximálny počet vrátených výsledkov
- type

Výstupne parametre pre túto službu sú:

- item - vracia položky ako objekt, ktorý môže byť predmetom v iných trojiciach
- label - vráti iba štítok nájdených položiek

Dotaz pre vyhľadanie všetkých hradov, ktoré v názve majú hrad by vyzeral nasledovne:

```
SELECT * WHERE {
  SERVICE wikibase:mwapi {
    # koncový hostiteľ
    bd:serviceParam wikibase:endpoint "www.wikidata.org";
    wikibase:api "EntitySearch"; # názov API služby
    mwapi:search "castle"; # hľadané slovo
    mwapi:language "en". # jazyk anglicky
    # do ?item prirať nájdené položky
    ?item wikibase:apiOutputItem mwapi:item.
    # priradenie do ?num číslo poradia nájdenia v API
    ?num wikibase:apiOrdinal true.
  }
  ?item wdt:P31/wdt:P279* wd:Q23413 # výsledok je hrad
} ORDER BY ASC(?num)
```

3.4 Wikimedia service

Wikimedia prevádzkuje verejnú inštanciu WDQS. Služba je dostupná na URL <http://query.wikidata.org/>. Existuje GUI verzia a verejný SPARQL koncový uzol.

GUI dovoľuje editovať a odoslať SPARQL dotaz na vyhľadávací nástroj. Výsledky sú zobrazene ako HTML tabuľka.

SPARQL dotazy môžu byť odoslané na koncový uzol buď GET alebo POST metódou. Výsledky sú vrátene ako XML alebo vieme nastaviť formát JSON.

Výsledky dotazu sú serializované ako jeden JSON objekt najvyššej úrovni. Objekt sa skladá z hlavičky "head" a výsledkov "results".

Nasledujúci príklad ukazuje výsledok SELECT dotazu.

```
{  
  "head": { "vars": [ "book" , "title" ]  
  } ,  
  "results": {  
    "bindings": [  
      {  
        "book": {  
          "type": "uri" ,  
          "value": "http://example.org/book/book6"  
        } ,  
        "title": {  
          "type": "literal" ,  
          "value": "Harry Potter and the Half-Blood Prince"  
        }  
      } ,  
    ]  
  }  
}
```

V hlavičke sú premenne Select-u v objekte s klúčom "vars" a hodnotou reprezentujúcou pole premenných. V členovi výsledky je objekt s klúčom "bindings" a hodnotou, ktorá je pole obsahujúce výsledky. Každý výsledok je objekt, ktorý pozostáva z dvojíc kľúč a hodnota, kde kľúč je názov premennej a hodnota je objekt reprezentujúci hodnotu premennej.

4. Návrh

V tejto kapitole je uvedený návrh architektúry softwarovej aplikácie pomocou C4 modelu. Nasleduje návrh štruktúry databázy, kde nájdeme aj schémy vytvárajúce tabuľky v databáze.

Zároveň tu nájdeme aj návrh jednotlivých SPARQL dotazov, ktoré sú potrebné pre fungovanie aplikácie.

4.1 Návrh softvérovej architektúry

Softwarovú architektúru vyjadrieme pomocou C4 modelu. Model pomáha opísat softwarovú architektúru počas návrhu samotnej architektúry a pri spätnom dokumentovaní už existujúceho kódu.

C4 model využíva prístup "abstrakcia na prvom mieste" k vytváraniu diagramov softwarovej architektúry, ktoré vznikli na základe abstrakcie reprezentujúcej uvažovanie architekta nad softvérom.

Na začiatok je ale potrebné definovať spoločnú sadu abstrakcii na vytvorenie jazyka vhodného na opis statickej štruktúry softvérového systému. Najvyššia úroveň abstrakcie je Softwarový systém. Ten opisuje prínosnú hodnotu používateľovi používajúci systém, či už je to človek alebo iný systém. Softwarový systém je tvorení z jedného alebo viacerých kontajnerov. Kontajner v C4 modely reprezentujú aplikácie a dátové úložiská. Pre funkčné fungovanie softvérového systém je potrebné aby kontajneri boli spustené, bežali.

Každý kontajner je tvorený jedným alebo viacerými komponentami, ktoré sú implementované jedným alebo viacerými kódovými elementami. Pojmom "kódový element" myslíme triedy, objekty, funkcie a podobne. V kontexte C4 modelu je pojem "komponentom" myslené zoskupenie súvisiacich funkcií zapuzdených za dobre definovaným rozhraním. Softwerový systém môžu využívať osoby, reprezentujúce jedného zo skupiných ľudí používajúcich softwerový systém.

Model pozostáva zo štyroch úrovni:

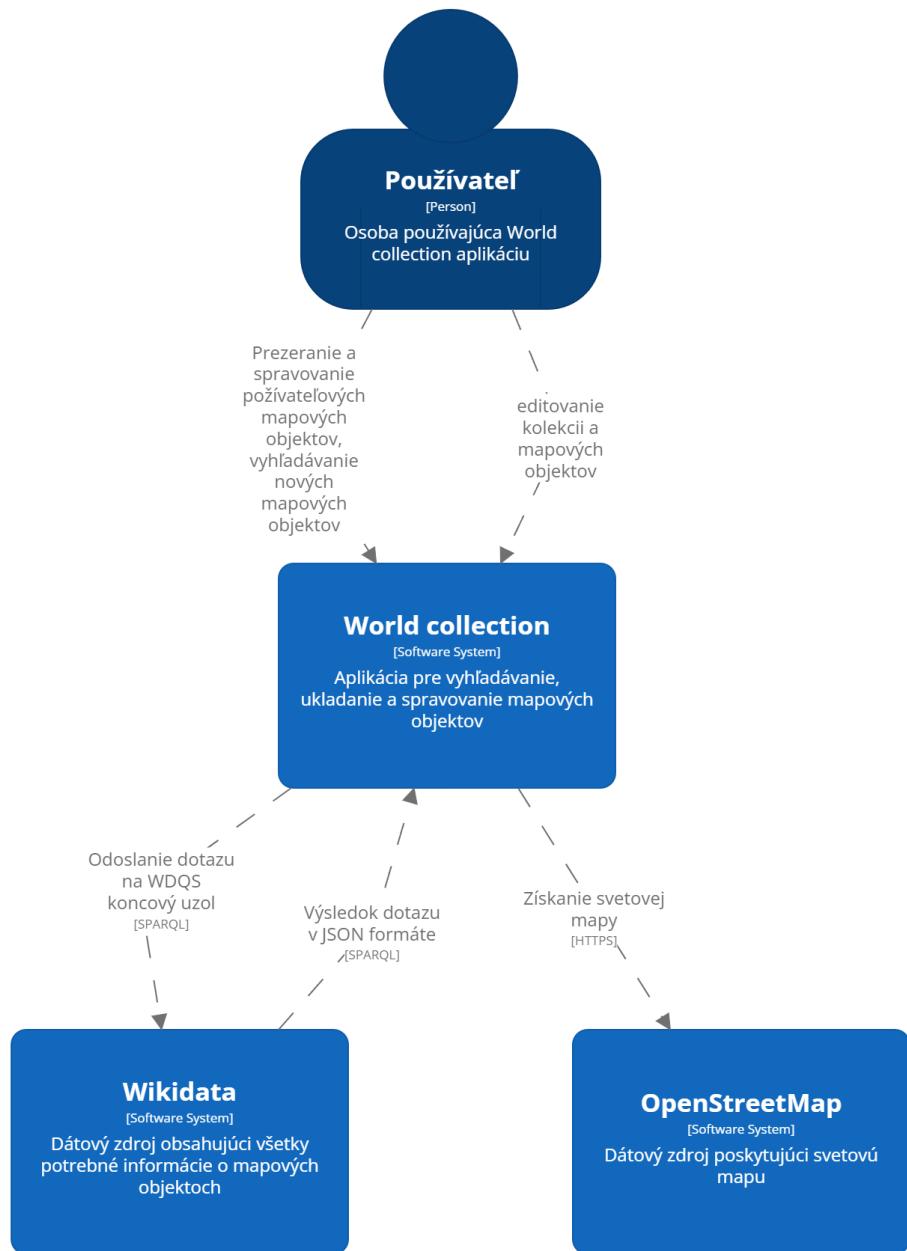
1. Systém Kontext diagram - zobrazuje systém ako krabicu v strede, obklopenú osobami a inými systémami, ktoré interagujú so systémom. Detaily ako technológie, protokoly a iné low-level detaily na tejto úrovni nie sú dôležité. Dôraz kladieme na osoby a softwarový systém.
2. Kontajner diagram - zobrazuje high-level tvar softvérovej architektúry, ako sú v nej rozdelené zodpovednosti. Zároveň zobrazuje technologické rozhodnutia a spôsob ako kontajneri medzi sebou komunikujú.
3. Komponent diagram - zobrazuje z čoho pozostáva konkrétny kontajner, aké sú jeho komponenty, ich zodpovednosť a technologické detaily.
4. Kód diagram - zobrazuje ako sú komponenty implementované kódom

Pre vyjadrenie softvérovej architektúry nie je potrebne využiť všetky úrovne. Často krát je postačujúce využiť iba Kontextovú a Kontajnerovú úroveň.

V nasledujúcich sekciách vyjadríme pomocou C4 modelu našu softwarovú architektúru v každej úrovni, okrem najnižšej úrovne Kód pomocou UML diagramov, ktoré nájdeme na obrázkoch.

zdroj : <https://c4model.com/>

4.1.1 Systém Kontext diagram



Obrázek 4.1: Systém Kontext diagram

V najvyššej úrovni abstrakcie je najdôležitejšia krabica v strede, ktorá reprezentuje náš softwarový systém “World collection”. Tento systém je používaný

používateľom. Používateľ je osoba využívajúca systém na vyhľadanie a spravovanie mapových objektov. Okolo nášho systému sú aj iné systémy reprezentujúce dátové zdroje, s ktorými náš systém interaguje. Tieto systémy sú nevyhnutnou súčasťou celej architektúry a bez nich by aplikácia nevedela fungovať.

Pre získanie svetovej mapy bude náš systém komunikovať s OpenStreetMap. Tie poskytnú systému dátu svetovej mapy pre vytvorenie interaktívnej mapy sveta.

Všetky informácie a dátu ohľadom mapových objektov bude systém získať z Wikidát.

4.1.2 Kontajner diagram

World collection je single-page webová aplikácia. Z toho dôvodu aplikácia pozostáva z dvoch hlavných časti: backend a frontend.

Backend je tvorený kontajnerom Server a Databáza. Na druhú stranu frontend je tvorený kontajnerom Single-page aplikácia (v skratke SPA).

Kontajner Server beží na inštancii serveru. Zo SPA bežiacej na strane používateľa prídu na server HTTP požiadavky obsahujúce parametre, ktoré používateľ zadal. Tieto požiadavky sú následne na strane serveru spracované pomocou technológie Flask, ktorá má za hlavnú úlohu priradiť požiadavku s konkrétnou cestou k danej funkcie. Flask zároveň vyrieši nasmerovanie jednotlivých požiadaviek a zavolá pre nich tu správnu funkciu. Server je implementovaný v jazyku Python. Úlohou serveru je komunikovanie s databázou, teda čítanie z a zapisovanie do databázy. Ďalšou úlohou Serveru je zstrojenie a posielanie dotazov na Wikidata. Wikidata následne vrátia výsledky dotazov a tie server spracuje a odošle na SPA.

SPA bežiacie na strane používateľa je implementované v jazyku Typescript pomocou frameworku React. Zároveň úlohou SPA je získať svetovú mapu z dátového zdroja OpenStreetMap, ktorý poskytne svetovú mapu pre našu aplikáciu.

4.1.3 Server Komponent diagram

Kontajner Server REST API, ktoré je rozdelené do dvoch API. WorldCollectionAPI a WikidataAPI. Požiadavka príde na server a na zakladá URL Flask technológia nasmeruje požiadavku na konkrétnu koncový uzol, kde sa zavola príslušná funkcia.

WorldCollectionAPI poskytuje rozhranie na čítanie z a zapisovanie do databázy. Na to API využíva databázové CRUD operácie. Tieto operácie poskytujú všetky potrebné operácie nad databázou, ktoré aplikácia potrebuje využívať. Každá CRUD operácia reprezentuje základnú operáciu nad databázou ako je napríklad vytvorenie kolekcie, zmenenie názvu kolekcie alebo odstránenie kolekcie.

Pre získanie spojenia s databazou sa použije komponenta Databázové spojenie.

Bokom existuje komponenta Inicializácia databázy. Ta poskytuje mechanizmus vytvorenia a inicializovania databázy na serveri. Tuto funkciu môže využívať iba správca serveru a preto je táto komponenta stojí bokom. Z toho dôvodu API neposkytuje možnosť zavolať tuto inicializáciu databázy. Tato funkcionalita je pridaná do Aplikácie Flaska ako príkaz, ktorý vie správca serveru zadať do príkazovej riadky.

WikidataAPI sa skladá z funkcií, kde každá funkcia reprezentuje jeden dotaz, ktorý je potrebne poslat na WDQS. Tieto dotazy je potrebne vytvoriť na základe parametrov. Jedným riešim by bolo napísat pre každú funkciu samostatný celý dotaz kde stačí len dosadiť vstupné parametre. To by bola ale strašná duplicita a preto navrhнемe a implementuje hierarchiu builderov, ktoré nám skonštruujú dotazy na základe vstupných parametrov. Tuto hierarchiu nám bude reprezentovať komponenta Wikidata Queries Builders.

Všetky tieto buildere vytvoria iba text dotazu. Ten bude následne poskytnutý komponente "SparqlPoint". Tá ma za úlohu poslať dotaz na WDQS a vrátený výsledok vrátiť v Json formáte. Tento výsledok sa nepošle rovno klientovi. Ešte pred poslaním na SPA bude výsledok skonvertovaný do nového Json formátu, ktorý upravuje Json do podoby aká je vhodná pre našu aplikáciu.

4.1.4 SPA Komponent diagram

SPA obsahuje dva APIProxy komponenty. Každá z nich obsahuje funkcie, ktoré robia asynchónne volania na server pomocou HTTPS požiadavkou. Parametre sa konvertujú do Json formátu a sú poslane na server v tele požiadavky.

SPA ponúka používateľovi štyri hlavné stavby aplikácie. Medzi jednotlivými stavmi sa používateľ prepína v navigačnom panely. Tieto stavby sú v modely C4 reprezentované ako komponenty s ktorými používateľ interaguje. Sú to nasledujúce komponenty:

Prehľad kolekcii - v tomto stave aplikácia zobrazí zoznam kolekcii. Z toho dôvodu musí táto komponenta požiadať Databázovú API o dátu z databázy reprezentujúce kolekcie. Používateľ je schopný vybrať kolekciu a následne si pozrieť všetky mapové objekty v danej kolekcii. Z toho dôvodu sa zavolá komponenta "Interaktívna mapa", ktorá poskytuje interaktívnu mapu. Dáta svetovej mapy sa získavajú pomocou knižnice "Leaflet". Táto knižnica získava svetovú mapu od poskytovateľa OpenStreetMap. Do mapy je potrebne vykresliť body (marker) reprezentujúce mapové objekty. Na to slúži komponenta "Marker", ktorou úlohou je vykresliť bod na mape. Po tom čo používateľ klikne na konkrétny Bod, aplikácia zobrazí UI okienko kde umožní prezeranie informácií o mapovom objekte (základne informácie, obrázok objektu, možnosť získania detailov, návštěvnosť, poznámky), možnosť napísat poznámky, nastaviť obrázok ikonky bodu alebo uviesť návštěvnosť. Každá z týchto vecí je ako samostatná komponenta. Každá z týchto komponent komunikuje s Databázovou API. pretože je potrebne získavať údaje z databázy alebo vykonané zmeny ako zmena poznámok, zmena ikonky alebo zmena návštěvnosti zapísat do databázy. Výnimkou je komponenta "Detaily mapového objektu", ktorá poskytuje detaily o objekte a preto je potrebne aby tieto informácie získala z Wikidat za pomoci "Wikidata API".

Pridanie mapového objektu - v tomto stave aplikácia zobrazí interaktívnu mapu a možnosť vyhľadať konkrétny mapový objekt. Objekt je vyhľadávaní za pomoci volania "Wikidata API". Po tom ako používateľ vyberie nájdený objekt sa na interaktívnej mape zobrazí daný mapový objekt. Preto táto komponenta využíva komponenty "Interaktívna mapa" a "Marker". V tomto konkretnom prípade Marker nevyužíva všetky štyri komponenty iba "Detaily mapového objektu". To je z dôvodu, že tento mapový objekt ešte nie je uložený v databáze a preto aplikácia smie poskytnúť iba detaily o mapovom objekte. Vyhľadané a vybraté mapové ob-

jekty aplikácia umožní uložiť do kolekcie. Na to existuje komponenta "Ukladanie mapových objektov a tvorba kolekcií", ktorá tento mechanizmus zapuzdruje.

Vyhľadanie mapových objektov - v tomto stave aplikácia zobrazí postupne možnosť zadat parametre vyhľadávania. Najprv výber kategórie, následne vyber spôsobu definovania oblasti vyhľadávania, definovanie konkrétnej oblasti a nako-nieč vyber filtrov a zadanie hodnôt daných filtrov. Potom sa uskutoční volanie na "WikidataAPI", kde sa tieto parametre zabalia do Json formátu a odošlú na server ako náklad v tele požiadavky. Server z nich postaví konkrétny dotaz a pošle ho na Wikidata. Vrátené výsledky sa zobrazia buď ako tabuľka alebo ako body na interaktívnej mape.

Editácia - v tomto stave aplikácia zobrazí tabuľku s kolekciami a možnosťami dané kolekcie editovať. Každú kolekciu používateľ je schopný otvoriť a následne sa mu zobrazí editačná tabuľka v ktorej vie editovať mapové objekty v danej kolekcií. Komponenta získava dátu z komponenty "WorldCollectionAPI API" a vykresluje tabuľky pomocou komponenty "Tabuľka", ktorá poskytuje možnosť vykresliť tabuľku s pätičkou tabuľky. V tejto pätičke vie používateľ sa preklikávať na jednotlivé strany tabuľky, nastavovať počet riadkov na stranu v tabuľke. Po editácii a uložení komponenta urobí API volanie na server aby boli zmeny uložené v databáze.

4.2 Návrh databázy

V databáze je potrebne mať uložene existujúce kolekcie a aké mapové objekty do kolekcií patria. K mapovému objektu je potrebne udržiavať údaj o jeho QNumber identifikujúci mapový objekt na Wikidatách pre získavanie informácií o objekte, ale aj iné informácie ako napríklad poznámky alebo dátum návštěvnosti.

Pre každú kolekciu si v databáze budeme musieť ukladať dátové informácie ako identifikátor a názov. Názov musí byť unikátny. Nemôžeme dovoliť aby v databáze boli uložené dve kolekcie s rovnakým menom.

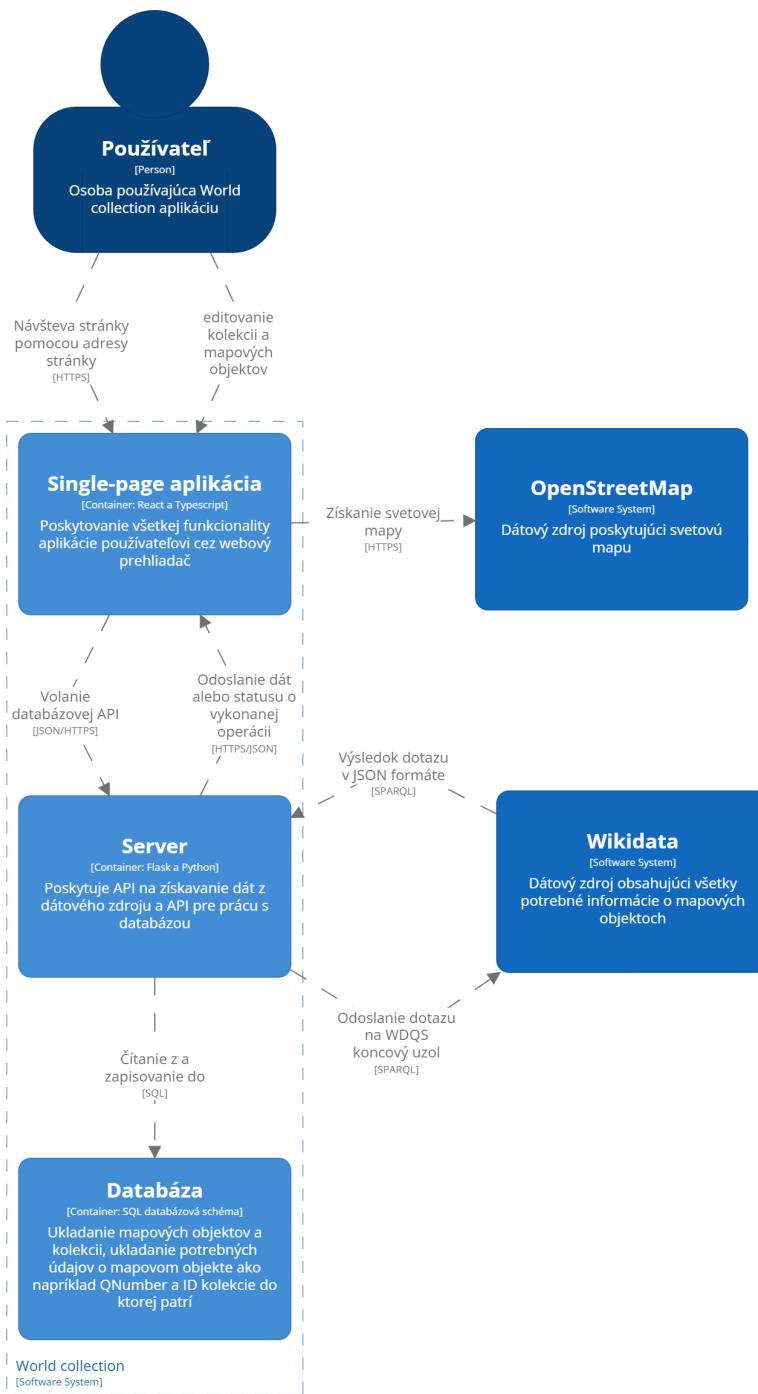
Pre každý mapový objekt si budeme ukladať nasledujúce informácie:

- QNumber objektu
- meno mapového objektu
- textový reťazec obsahujúci triedy, ktorými je daný objekt inštanciou na Wikidatách. Tento údaj by sa dal stále získavať z Wikidat, ale je rozumnejšie si ho raz uložiť a nezatažovať Wikidata. Reťazec obsahuje špeciálny znak, ktorý oddeluje jednotlivé triedy.
- údaj o zemepisnej šírke a výške. Tento údaj by sme mohli získavať cez Wikidata vďaka QNumber, ale je výhodnejšie si tieto informácie uložiť a nezatažovať neustále Wikidata.
- údaj či bol mapový objekt navštívený, pomocou boolean hodnoty
- dátum návštěvy. Kedže podporujeme možnosť zadat dátum návštěvy aj ako rozsah a v rôznych precíznostiach (dátum, mesiac a rok, rok), preto v databáze budeme mať uložene 3 údaje, ktoré spolu budú reprezentovať dátum návštěvy. Prvým bude dátum od a druhým dátum do. Tieto údaje

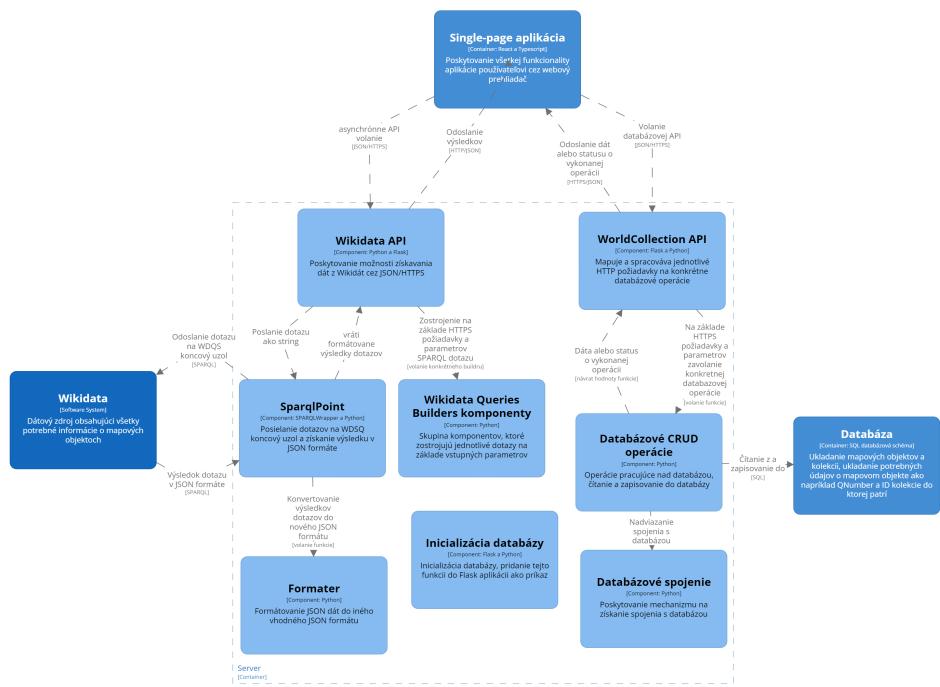
budú typu Date. Obsahovať budú cely dátum. V prípade zadania dátumu a nie rozsahu bude dátum do práznej hodnoty. Tretím údajom bude formát dátumu. V ňom je informácie ako reprezentovať dátum návštevy. Čí ako rozsah alebo ako samostatný dátum a v akej precíznosti je potrebne vyjadriť dátum.

- názov obrázka ikonky. SPA na strane používateľa na základe názvu tohto udajú zobrazí príslušný obrázok ako ikonku mapového objektu
- poznámky ako textový reťazec

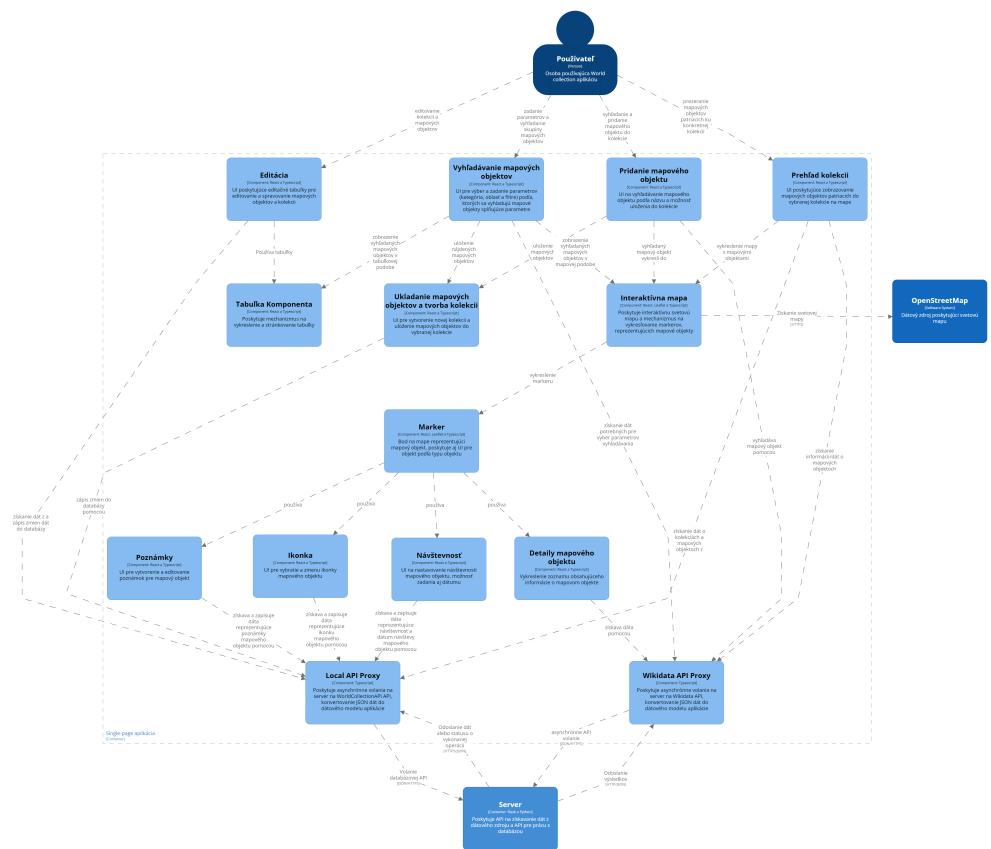
Zároveň je potrebné si v databáze držať vzťahy medzi kolekciami a mapovými objektami. Vďaka týmto vzťahom budeme schopní povedať do akej kolekcie patri daný mapový objekt.



Obrázek 4.2: Kontajner diagram



Obrázek 4.3: Server Komponent diagram



Obrázek 4.4: SPA Komponent diagram

5. Implementácia aplikácie

Vývoj samotnej aplikácie je rozdelený do dvoch častí backend a frontend. V tejto kapitole si ukážeme technológie zvolené pre vývoj, navrhнемe a vysvetlíme jednotlivé SPARQL dotazy, ktoré bude backend posielat na Wikidata, popíšeme implementáciu aplikácie a popíšeme implementáciu databázy.

5.1 Technológie použite pre vývoj

5.1.1 Backend

Volba jazyka a frameworku

Backend je možné vyvíjať v rôznych jazykoch, kde pre každý jeden z nich existujú frameworky vhodné na vývoj. Pred začiatkom návrhu a implementácie je potrebné zvoliť vhodný jazyk na základe dopytu a popularity programovacieho jazyka, alebo podľa požiadavkou potrebných pre vývoj aplikácie. Časte programovacie jazyky pre vývoj backendu sú napríklad Javascript, Python, PHP, Java, alebo Ruby. Pre vývoj backendu sme zvolili programovací jazyk Python. Výhoda tohto jazyka je jeho jednoduchosť a to, že existuje obrovské množstvo knižníc pre tento jazyk. Podľa mňa je jazyk Python jednoduchý na používanie a kód je ľahko čitateľný.

Frameworkov pre vývoj backendu v programovacom jazyku Python existuje viacero. Najpopulárnejšie sú asi najskôr Django a Flask. Z týchto dvoch sme zvolili Flask, pretože je jednoduchý na používanie, obsahuje málo požiadaviek na iné knižnice a umožňuje vyber preferovaných návrhových vzorov a voľbu databázy.

Použité knižnice

Python obsahuje obrovskú štandardnú knižnicu s množstvom modulov, ktoré sú požívané v tejto práci. Tieto moduly si popisovať nebudeme.

Ako sme už spomínali, hlavnou použitou knižnicou je Flask. Vďaka nemu vytvoríme webový server, s ktorým bude frondend komunikovať pomocou HTTP požiadavkou. Jednou z hlavných úloh serveru je na základe parametrov prichádzajúcich na server pomocou HTTP požiadavkou zakonštruovať SPARQL dotazy a tie následne poslať na koncový uzol. Odoslanie SPARQL dotazov a spracovanie vrátených dá ma na starosť knižnica SPARQLWrapper, ktorú na túto funkcionality použijeme.

Virtuálne prostredie

Pre vývoj našej aplikácie použijeme virtuálne prostredie na spravovanie závislosti. To nám vyrieši problém s používaním rôznych verzii knižníc, aby nainštalovalné balíky pre jeden projekt neovplyvňovali ostatné projekty. Python obsahuje modul venv, ktorý slúži na vytváranie virtuálnych prostredí.

5.1.2 Frontend

Framework a jazyk

Ako už bolo spomenuté v návrhu architektúry aplikácie, frontend implementujeme ako Single-page aplikáciu. To znamená, že budeme dynamicky prepisovať webovú stránku na základe získaných dát z backendu namiesto toho aby sme načítavali úplne nove webové stránky. Pre takýto vývoj aplikácie existuje viacero frameworkov a knižníc. Zvolili sme Javascriptovú knižnicu React. Základom Reactu je stavanie užívateľských rozhraní pomocou takzvaných “komponentov”. Tieto komponenty sú typický napísane použitím JSX (JavaScript Syntax Extension). JSX pripomína jazyk HTML zmiešaní s Javascriptom a z toho dôvodu je implementácia komponent jednoduchšia pre developerov poznajúcich HTML. Komponenty môžu byť, ale implementované čisto použitím Javascriptu bez použitia JSX. V našej implementácii využijeme JSX.

Javascript nie je typovaný jazyk a z toho dôvodu použijeme Typescript. Ten sa za behu preloží do Javascriptu, ale počas vývoja nám poskytne typovanie premenných. To nám pomôže aby sme sa vyhli častým chybám s dynamickými jazykmi ako Javascript.

Bootstrap

Aplikáciu v podobe webovej stránky je potrebné vhodne nadizajnovať, aby jednotlivé elementy vyzerali používateľsky rozumne. Klasický spôsob je použitím jazyka CSS. Počas vývoja aplikácie sa budeme skôr sústrediť na funkcionality ako na výzor stránky. Z toho dôvodu a ďalších ako napríklad ušetrenie si práce, sme sa rozhodli pre použitie nástroja Bootstrap. Jedná sa o nástroj, ktorý pridáva preddefinovaný výzor elementom na webových stránkach. Jedná sa o nahradu CSS. Vďaka Bootstrapu nám stačí vhodne pomenovať triedu elementu a daný element sa už zobrazí tak ako ho Bootstrap preddefinoval.

Použité knižnice

Na vývoj frontendu použijeme viacero knižníc. Všetky si uvádzat nebude, iba tie najdôležitejšie. Pre možnosť aby bola aplikácia vhodne responzívna použijeme knižnicu react-responsive, ktorá nám pomôže pri vyváraní aplikácie. Ako už bolo spomenuté v návrhu, pre implementáciu interaktívnej svetovej mapy využijeme javascriptovú knižnicu Leaflet. Zároveň použijeme knižnicu React Leaflet, ktorá poskytuje väzbu medzi knižnicou Leaflet a samotným Reactom. Tato knižnica využíva Leaflet k tomu aby vyjadriala vrstvy Leafletu ako komponenty Reactu.

5.2 Implementácia SPARQL dotazov

V tejto sekcií si postupne prejdeme a popíšeme všetky SPARQL dotazy, ktoré naša aplikácia bude posielat na Wikidata WDQS SPARQL koncový uzol. Pre každý dotaz uvedieme jeho účel, navrhнемe a implementujeme riešenie a popíšeme problémy, ktoré je potrebné počas návrhu a implementácie dotazov riešiť.

Jedná sa hlavne problémy spojené s optimalizáciou, pretože niektoré dotazy trvajú príliš dlho alebo ani neskončia do 60 sekúnd. V takom prípade WDSQ presuší proces dotazu a vráti chybu.

5.2.1 Vyhľadávanie mapových objektov

Funkčná požiadavka : Používateľ je schopný vyhľadať konkrétny mapový objekt podľa zadania názvu objektu, pretože si ho môže chcieť uložiť do svojej kolekcie.

Jedným z prvých a základných dotaz je vyhľadávanie mapových objektov. Používateľ zadá klúčový textový reťazec reprezentujúci celý alebo časť názvu mapového objektu. Podľa poskytnutého reťazca musí aplikácia na strane serveru skonštruovať SPARQL dotaz, ktorý vyhľadá mapové objekty na základe názvu zhodujúcim sa s textovým reťazcom.

Pre každý nájdený mapový objekt musíme dotazom nielen získať názov, ale aj základne informácie o mapovom objekte potrebne pre aplikáciu. Zakladám je získať QNumber a názov objektu. Používateľovi je potrebné poskytnúť aj popis objektu aby vedel, čo daný objekt reprezentuje. Z toho dôvodu musí dotaz pre nájdene objekty získať aj ich popis. Nájdené objekty aplikácia zobrazí používateľovi v podobe zoznamu. Keď používateľ vyberie niektorý z objektov, ten ktorý hľadal, tak aplikácia zobrazí tento objekt bodom na mape. Pre túto funkciu je potrebne získať aj súradnice nájdených mapových objektov.

Pri vyhľadávaní podľa názvu je potrebné vyhľadávať iba objekty, ktoré vieme reprezentovať ako mapové objekty v našej aplikácii. Z toho vyplýva, že dotaz musí obsahovať statement, ktorý z množiny všetkých objektov zhodujúcich sa v názve zahodí tie objekty, ktoré nie sú mapovými objektami. Riešením je definovať, že množina objektov musí byť inštanciou triedy, ktorá ma nad sebou v hierarchii rozumného rodiča. Tato trieda musí ručiť, že inštancie všetkých pod-tried tejto triedy sú mapové objekty. Navrhнемe preto triedu "geografická lokácia"(Q2221906). Popis tejto triedy na Wikidatach je nasledovný : "bod alebo územie na povrchu Zeme alebo aj inde". Tato trieda dobre definuje mapový objekt.

Teraz si ukážme ako by mohol vyzerať dotaz. Najprv sa zaoberajme iba vyhľadávaním objektov na základe názvu a získavanie informácií o mapovom objekte pridáme až na koniec.

Prvý dotaz, kde vyhľadávame mapové objekty zhodujúce sa s názvom "Tokyo", vyzera nasledovné:

```
SELECT DISTINCT ?item ?name
WHERE
{
# objekt je "geografická lokácia"
?item wdt:P31/wdt:P279* wd:Q2221906.
# získanie názvu
?item rdfs:label ?name.
# filtričia, názov objektu iba v anglickom jazyku
FILTER (lang(?name) = "en")
# filtričia, názov objektu musí obsahovať text "Tokyo"
FILTER(CONTAINS(?name, "Tokyo")).
```

```

SERVICE wikibase:label {
  bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
}

```

Takýto dotaz ale nezbehne do 60 sekúnd a WDQS ho prerusí. Problém je, že WDQS prechádzajú všetky objekty, ktoré sú v hierarchii inštanciou triedy spadajúcej pod geografickú lokáciu. Následne sa zahadzujú objekty nezhodujúce sa so zadaným názvom. Objektov "geografická lokácia" existuje obrovské množstvo a WDQS nestihne prejsť za daný čas všetky.

To nie je ale jediný problém. Nahradením triedy "geografická lokácia" za inú triedu napríklad "mesto" vyrieši iba to, že dotaz teraz dobehne, ale trvá to okolo 15 až 20 sekúnd. Je nepriyatelné aby používateľ pri vyhľadávaní objektov podľa zadania názvu musel čakať až 20 sekúnd, kým mu aplikácia vyhľadá objekty.

Tento problém a každý podobný dotaz, kde dochádza k vyhľadávaniu objektov na základe zadania názvu objektu vieme vyriešiť použitím MWAPI. V MWAPI si zvolíme podporovanú službu "EntitySearch". Tá vyhľadáva Wikibase entity, teda objekty podľa názvu entity, objektu. Ako vstupné parametre nastavíme jazyk na anglicky. Služba vyhľadáva na základe anglických názvov objektu, a do parametru "search" zadáme textový reťazec poskytnutý používateľom, na základe ktorého bude služba vyhľadávať objekty. Ako výstupný parameter vyberieme "item", kde do premennej služba priradí všetky nájdené objekty.

Dotaz s použitím služby vyzerá nasledovne:

```

SERVICE wikibase:mwapi {
  # povinný parameter - koncový uzol
  bd:serviceParam wikibase:endpoint "www.wikidata.org";
  # povinný parameter -
  # služba na vyhľadávanie objektov podľa názvu
  wikibase:api "EntitySearch";
  # vyhľadávanie objektov s názvom "tokyo"
  mwapi:search "tokyo";
  # nastavenie anglického jazyka
  mwapi:language "en" .
  # priradenie nájdených objektov do premennej
  ?item wikibase:apiOutputItem mwapi:item.
  ?num wikibase:apiOrdinal true. }

```

S využitím služby MWAPI dotaz zbehne veľmi rýchlo. Priemer je okolo pol sekundy. Tým spĺňame nefunkčnú požiadavku:

Aplikácia je dostatočne vykoná. Vyhladávanie dát na základe zadania názvu v niektorom z vyhľadávačov. Netrvá viac ako sekundu.

Pre kompletnejší dotaz je potrebné pridať statement, ktorý sa postará aby nájdené objekty boli "geografickými lokáciami" a pridať trojice na získanie informácií o mapovom objekte. To je názov, opis a súradnice objektu.

Finálny dotaz, pre vyhľadávanie s klúčom "tokyo" vyzerá nasledovne:

```

SELECT DISTINCT
?description
(?lat AS ?lati)

```

```

(?item AS ?QNumber)
(?lon AS ?long)
(?itemLabel AS ?name)
WHERE {
# služba na vyhľadávanie objektov podľa názvu
SERVICE wikibase:mwapi {
bd:serviceParam wikibase:endpoint "www.wikidata.org";
wikibase:api "EntitySearch";
mwapi:search "tokyo";
mwapi:language "en" .
?item wikibase:apiOutputItem mwapi:item.
?num wikibase:apiOrdinal true. }
# objekt je "geografickou lokáciou"
?item wdt:P31/wdt:P279* wd:Q2221906 .
# pridanie nápovedi pre optimalizáciu
hint:Prior hint:gearing "forward".
# získanie súradníc
?item p:P625 ?coord .
?coord psv:P625 ?coord_node .
?coord_node wikibase:geoLongitude ?lon .
?coord_node wikibase:geoLatitude ?lat .
# získanie opisu
?item schema:description ?description .
# zahodenie opisov, ktoré nie sú v anglickom jazyku
FILTER (lang(?description) = "en")
# pridanie služby pre získanie názvov
SERVICE wikibase:label {
bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
?item rdfs:label ?itemLabel .
}
# zoradiť výsledky podľa poradia aké boli vrátene MWAPI službou
} ORDER BY ASC(?num)

```

Napovedá "hint:Prior hint:gearing "forward"" sa môže použiť za trojicou obsahujúcou cestu. Je to optimalizačná stratégia, kde povieme dotazu aby prechádzal v opačnom poradí ako je default. V dotaze rozhodujeme či daný objekt je "geografickou lokáciou". Default-ne by dotaz začal s triedou "geografická lokácia" a prechádzal rekurzívne jej podtriedy. My ale použijeme túto stratégiu aby dotaz prechádzal cestu opačne. Začne sa s jednou triedou, ktorou je objekt inštanciou, a pokračuje sa v prechádzaní jej nad-tried smerom dopredu.

Bez použitia tejto nápovedi je čas dobehnutia dotazu trvá viac ako 1 sekundu a aplikácia teda nesplňuje nefunkčnú požiadavku.

Tento dotaz zároveň použijeme pri vyhľadávaní objektov, ktoré používateľ je schopný definovať ako stred kruhu vo funkčnom požiadavku: Pre spomínanú definíciu kruhu, v ktorom sa budú objekty vyhľadávať, používateľ je schopný stred tohto kruhu nastaviť na mape posúvaním bodu, alebo vyhľadaním konkrétneho miesta pomocou vyhľadávania. V takom prípade sa stred kruhu presunie na súradnice nájdeného a vybratého objektu. Pretože používateľ môže chcieť vyhľadávať objekty v blízkom okolí okolo konkrétneho miesta.

5.2.2 Popis a obrázok

Funkčná požiadavka: Používateľ potrebuje od aplikácie aby mu pre vybraný objekt získala obrázok objektu, aby používateľ mal predstavu ako objekt vyzerá.

Pre každý uložený mapový objekt, ktorý si používateľ uložil do kolekcie, si v databáze nebudeme udržiavať hodnotu popisu a obrázku objektu. Je potrebne navrhnúť SPARQL dotaz, ktorý pre zvolený objekt získá obrázok a popis. Tento dotaz sa bude volať, zakaždým keď aplikácia potrebuje prezentovať uložený mapový objekt používateľovi. Návrh takéhoto dotazu je celkom jednoduchý. Obrázok získame z vlastnosti objektu "obrázok"s identifikačným číslom vlastnosti P18. Popis na druhú stranu získame použitím prefixu schéma s hodnotou "description", ktorý získá popis objektu. Popis existuje v rôznych jazykoch a preto je ešte potrebne zahodiť v dotaze iné ako ten v anglickom jazyku.

Vlastnosť "obrázok" môže nadobúdať viacero hodnôt, inak povedane existuje viacero obrázkov daného objektu. Pre potreby aplikácie postačí získať iba jeden z nich. Tento problém vyriešime použitím agregovanej funkcie SAMPLE. Tá vyberie jeden ľubovoľný obrázok z množiny všetkých obrázkov objektu. Aby sa dala tato funkcia v dotaze použiť je potrebne zoskupiť výsledky a v tomto prípade je to cez premennú držiacu popis.

Dotaz pre získanie obrázka a popisu pre mesto Tokio (Q1490) vyzerá nasledovne:

```
SELECT
# vybranie ľubovoľného obrázoku
(SAMPLE(?img) AS ?image)
?desc
WHERE {
# získanie obrázka
wd:Q1490 wdt:P18 ?img .
# získanie popisu
wd:Q1490 schema:description ?desc .
# filtričia, zahodenie popisov,
# ktoré nie sú v anglickom jazyku
FILTER (LANG(?desc) = "en")
SERVICE wikibase:label {
bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
}
# zoskupenie podľa popisu
} GROUP BY ?desc
```

5.2.3 Odkaz na Wikipédia článok

Funkčná požiadavka: Používateľ je schopný požiadať aplikáciu aby mu pre vybraný objekt vyhľadala odkaz na článok, sídliaci na Wikipédii, ak článok existuje, aby používateľ mal možnosť sa o objekte dozvedieť viac informácií vo forme článku.

Aplikácia podporuje iba jeden jazyk a to anglicky. Z toho dôvodu nájdený odkaz na článok o objekte musí sídliť na anglickej Wikipédii.

Dotaz pre získanie odkazu na článok o objekte na anglickej Wikipédii vyzerá nasledovne:

```

SELECT DISTINCT ?article
WHERE {
  # získanie článku o objekte
  ?article schema:about wd:Q1490 .
  # článok sídli na anglickej Wikipédií
  ?article schema:isPartOf <https://en.wikipedia.org/> .
  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
  }
}

```

5.2.4 Detaily mapového objektu

Funkčná požiadavka : Používateľ je schopný požiadať aplikáciu aby mu získala podrobnejšie informácie o objekte(budeme označovať ako detaily objektu), medzi ktorými vie používateľ filtrovať podľa názvu informácie, aby používateľ získal konkrétnu informáciu o objekte.

Aplikácia získava zdroje jedine z Wikidat pomocou SPARQL dotazov posielajúcich sa na WDQS koncový uzol. Aplikácia by mohla získavať informácie o mapovom objekte aj z iných zdrojov, ale rozhodli sme sa používať ako zdroj jedine Wikidata.

Na Wikidátoch ma každý objekt nejaké vlastnosti obsahujúce jedno alebo viac hodnôt. Každá z týchto vlastnosti poskytuje informáciu o objekte. Informáciu o objekte označujme ako detail objektu a v tejto pod-sekcii sa zaoberejme ako získavať detaily objektu.

Pôvodný návrh bol statické získavanie detaily. Analyzovali by sme rozumne detaily, spísali ich a vložili do dotazu, ktorý by bol rovnaký a pre každý mapový objekt by získaval zadané detaily. Každé získavanie vlastnosti by bolo zabalené do bloku OPTIONAL, pretože nie každú vlastnosť by musel objekt o sebe mať.

Dotaz pre získanie detailov by vyzeral nasledovne:

```

SELECT ?population ?elevation
WHERE
{
  OPTIONAL{
    # Získanie populácie
    wd:Q1490 wdt:P1082 ?population.
    # Získanie nadmorskej výšky
    wd:Q1490 wdt:P2044 ?elevation.
    # ... ďalšie trojice
    # pre získanie hodnôt vlastnosti
  }
  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
  }
}

```

Problém je ale analyzovať rozumne detaily. Možné kategórie mapových objektov existuje obrovské množstvo a objekty v týchto kategóriách môžu obsahovať

zaujímavé detaily, ktoré by sme nezanalyzovali. Je preto potrebne vymyslieť dynamické riešenie.

Ďalší návrh je získavať všetky detaily o objekte. Získať hodnoty všetkých vlastnostiach a tie následne prezentovať používateľovi. Jedna sa o lepsiu variantu, ale do množiny všetkých detailov pripadnú aj vlastnosti, ktoré nie sú vhodne na prezentovanie používateľovi. Príkladom môžu byť takzvane "identifikátory". To sú vlastnosti obsahujúce hodnotu ako identifikátor v inom systéme. Napríklad "OpenStreetMap relation ID", čo je ID relácia daného objektu v OpenStreetMap. Tento návrh je vhodný na použitie, ale je potrebne vlastnosti rozumne filtrovať.

Každá hodnota vlastnosti je určitého dátového typu. Pri vytváraní novej vlastnosti na Wikidatach sa určí jej dátový typ a ten sa nemení. Ak ma objekt tuto vlastnosť, tak v nej bude vždy hodnota v tomto dátovom type. Dátový typ vlastnosti zároveň definuje správanie a aké ďalšie dáta môže vlastnosť obsahovať. Napríklad zoberme si dátový typ "Quantity". Hodnota takéhoto typu vlastnosti môže obsahovať atribút "unit", ktorý hovorí o v akých jednotkách je kvantitatívna hodnota vyjadrená. Riešením problému je vymedziť vlastnosti na základe dátového typu.

Existuje viacero dátových typov, ale vymenujme iba tie, ktoré sme po analýze vybrali, že budú podporované. Pojmom podporované myslime, že dotaz bude filtrovať iba tie vlastnosti, ktorých dátový typ patrí do množiny vybratých dátových typov. Vybrane dátové typy sú:

1. Monolingualtext - textový retazec, ktorý nie je preložený do iných jazykov
2. Quantity - kvantitatívna hodnota reprezentovaná desatinným číslom
3. Time - dátum vyjadrený v Gregoriánskom alebo Juliánskom kalendári
4. WikibaseItem - hodnota je iná entita, objekt existujúci na Wikidáta
5. Url = URL, ktorá identifikuje externý zdroj

Týmto spôsobom vyriešime spomínaný problém s "OpenStreetMap relation ID". Tato vlastnosť nadobúda hodnoty dátového typu "External identifier", čo je textový retazec reprezentujúci identifikátor použitý v inom externom systéme. Keďže v dotaze filtrujueme vlastnosti na základe dátových typov, tak vlastnosti vyjadrene ako External identifier budú zahodene.

Rozhodnút v dotaze, či vlastnosť je podporovaného dátového typu dokážeme nasledujúcimi trojicami:

```
# uloženie podporovaných dátových typov do premennej  
VALUES ?supportedDataTypes {  
    wikibase:Monolingualtext  
    wikibase:Quantity  
    wikibase:Time  
    wikibase:WikibaseItem  
    wikibase:Url }  
# zahodenie vlastnosti dátového typu, ktorý nie je podporovaný  
?wd wikibase:propertyType ?supportedDataTypes .
```

Poznámka: v premennej ?wd sú uložene vlastnosti.

Vlastnosť môže nadobúdať viacero hodnôt. Tieto hodnoty sú označene "rankom". V dotaze preto zadáme, aby vrátený výsledok dotazu obsahoval pre vlastnosti hodnoty s najlepším, najvhodnejším rankom. To spravíme nasledovne:

```
# získanie hodnoty s najlepším rankom  
?statement rdf:type wikibase:BestRank .
```

Po analýze tohto dotazu sme ale našli zopár vlastnosti s podporovaným dátovým typom, ktoré nie sú v kontexte detailov vhodne. Napríklad vlastnosť "je inštanciou"(P131). Hodnotu tejto vlastnosti získavame v inom dotaze a následne ju ukladáme do databázy s mapovým objektom. Tieto hodnoty definujú mapový objekt a často krát ich aplikáciu prezentuje používateľovi. Preto je vhodné si tento údaj uložiť do databázy. Z toho dôvodu tuto vlastnosť nie je potrebné znova získať v kontexte získavania detailov.

Nežiadúce vlastnosti definujeme ako zoznam a v dotaze ich vyfiltrujeme nasledovne:

```
FILTER (?wd not in ( wd:P190 , wd:P31 , wd:P150 , wd:P355 , wd:P131 , wd:P1381 )
```

Poznamka: v premennej ?wd sú uložené vlastnosti.

Posledným problém je, že hodnoty niektorých dátových typov obsahujú ďalšie doležité informácie. Konkrétnie dátový typ "Time"obsahuje "timePrecision", v ktorej je vyjadrená precíznosť času. To znamená, či dátum ako hodnota v tejto vlastnosti ma byť interpretovaná ako cely dátum, rok alebo storočie. Tato informácia je potrebná pre interpretovanie a zobrazovanie časového detailu používateľovi.

Pre vlastnosti s dátovým typom "Quantity"je potrebné získať informáciu v "quantityUnit", ktorá obsahuje jednotku v ktorej je kvantitatívna hodnota vyjadrená.

Tieto informácie sú pre aplikáciu nevyhnutné aby dokázala správne prezentovať detaily.

Aby sme nemuseli pre tieto dátové typy vytvárať rozdielne dotazy, tak získavanie týchto dvoch informácií obalíme do OPTIONAL bloku.

Kompletný dotaz vyzerá nasledovne:

```
SELECT DISTINCT  
?unit  
?property  
(GROUP_CONCAT( DISTINCT ?valueLabel ;separator=<space>) AS ?values)  
?dataType  
?timePrecision  
WHERE {  
# získanie všetkých vlastností  
wd:Q1490 ?p ?statement .  
# získanie hodnôt s najlepším rankom  
?statement rdf:type wikibase:BestRank .  
?statement ?ps ?ps_ .  
?wd wikibase:claim ?p .  
?wd wikibase:statementProperty ?ps .  
?wd wikibase:statementValue ?psv .
```

```

# definovanie podporovaných dátových typov
VALUES ?supportedDataTypes {
wikibase:Monolingualtext
wikibase:Quantity
wikibase:Time
wikibase:WikibaseItem
wikibase:Url  }
# zahodenie vlastnosti s iným dátovým typom
?wd wikibase:propertyType ?supportedDataTypes .
# zahodenie konkrétnych nežiadúcich vlastností
FILTER (?wd not in ( wd:P190 , wd:P31 , wd:P150 , wd:P355 , wd:P131 , wd:P1383 ))
# získanie dátového typu
?wd wikibase:propertyType ?dataType .
# získanie hodnoty vlastnosti
?statement ?ps ?value .
# získanie časovej precíznosti
# ak je vlastnosť dátového typu "Time"
OPTIONAL {?statement ?psv [wikibase:timePrecision ?timePrecision].}
# získanie jednotky
# ak je vlastnosť dátového typu "Quantity"
OPTIONAL {?statement ?psv [wikibase:quantityUnit ?unitValue].}
SERVICE wikibase:label {
bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
?wd rdfs:label ?property .
?value rdfs:label ?valueLabel.
?unitValue rdfs:label ?unit. }
} GROUP BY ?dataType ?timePrecision ?property ?unit

```

Ak vlastnosť nadobúda viacero hodnôt, tak tieto hodnoty zoskupíme do jednej hodnoty obsahujúcu decimeter aby následne vedela aplikácia rozdeliť a interpretovať samotné hodnoty. Z toho dôvodu zoskupíme vlastnosti a použijeme agregovanú funkciu GROUP CONCAT.

5.2.5 Vyhľadávanie kategórii

Funkčná požiadavka: Používateľ potrebuje od aplikácie možnosť vyhľadať možne a rozumné triedy definujúce objekty podľa názvu triedy, aby sa následne trieda dala použiť ako parameter.

Počas vyhľadávania skupiny mapových objektov používateľ môže zadať parametre, ktoré ovplyvnia nájdenú skupinu mapových objektov. Prvým týmto parametrom je výber kategórie. Kategória je vyjadrená ako trieda na Wikidatách definujúca určitú skupinu objektov. Používateľ potrebuje vyhľadať a následne vybrať kategóriu na základe názvu. Navrhнемe preto dotaz vyhľadávací rozumne triedy, ktoré je možne použiť ako kategóriu v aplikácii. Vyhľadávanie na základe názvu je rovnaké ako u vyhľadaní mapových objektov. Použijeme MWAPI službu "EntitySearch". Zároveň získame názov a popis triedy aby používateľ bol schopný lepšie pochopiť danú triedu, aký skupinu objektov reprezentuje. QNumber získame aby aplikácia vedela neskôr použiť tuto kategóriu v dotaze pre vyhľadávanie skupiny objektov ako parameter definujúci kategóriu.

Nájdene výsledky musia byť vhodne triedy. Po analýze hierarchie Wikidát navrhne, že vhodné triedy musia patriť v hierarchii pod triedou "geografická lokácia" (Q2221906).

Dotaz vychladajúci vhodné kategórie s klúčovou hodnotou "jaskyne" vyzerá nasledovný:

```
SELECT DISTINCT
  (?item AS ?QNumber)
  ?description
  (?itemLabel AS ?name)
WHERE {
  # vyhľadanie objektov na základe názvu
  SERVICE wikibase:mwapi {
    bd:serviceParam wikibase:endpoint "www.wikidata.org";
    wikibase:api "EntitySearch";
    mwapi:search "castle";
    mwapi:language "en" .
    ?item wikibase:apiOutputItem mwapi:item.
    ?num wikibase:apiOrdinal true. }
  # priradenie hodnôt do premennej
  VALUES ?classRestrictions {wd:Q2221906 } 
  # objekt musí spadať pod triedu
  # geografická lokácia
  ?item wdt:P279* ?classRestrictions .
  # napovedá na optimalizáciu
  hint:Prior hint:gearing "forward".
  # získanie popisu objektu
  ?item schema:description ?description .
  # zahodenie iných popisov ako tie v anglickom jazyku
  FILTER (lang(?description) = "en")
  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
    ?item rdfs:label ?itemLabel . }
} ORDER BY ASC(?num)
```

Od aplikácie očakávame možnosť definovať aj výnimky. Tieto výnimky sú triedy spadajúce pod triedu vybranej kategórie. Dotaz vychladajúci pod-kategórie je rovnaký ako vyššie uvedený, ale do premennej ?classRestrictions namiesto "geografickej lokácie" priradíme triedu, ktorá identifikuje vybratú kategóriu.

Ked si používateľ vyberie kategóriu a vyberie aj nejakú pod-kategóriu, a následne chce ďalej vyhľadávať ďalšie možné pod-kategórie, tak je očakávane, že dotaz zahodí pod-triedy spadajúce pod už zvolené pod-kategórie. Tento problém vyriešime pridaním MINUS bloku, kde rovnakým princípom, ako sme vyjadrovali, že trieda musí spadať pod triedu "geografická lokácia", vyriešime zahodenie tých tried, ktoré spadajú pod triedy zvolených pod-kategórií ako výnimky.

Pridaná časť dotazu vyzerá nasledovný:

```
MINUS{
VALUES ?classExceptionRestrictions {wd:Q3480180 } 
?item wdt:P279* ?classExceptionRestrictions .
```

```
    hint:Prior hint:gearing "forward".  
}
```

Poznámka: do premennej ?classExceptionRestrictions priradíme triedy, ktoré sú vybraté ako výnimky.

5.2.6 Vyhľadávanie administratívneho územia a regiónov

Funkčná požiadavka: Používateľ je schopný vyhľadať podľa názvu a definovať parameter územia ako súčasne existujúce územie, aby používateľ vedel vyhľadávať objekty na území kontinentu, krajiny alebo administratívneho územia.

Vyhľadávanie skupiny mapových objektov podporuje definovanie oblasti, respektíve územia, v ktorom sa musia vyskytovať nájdene mapové objekty. Územie môže byť definované ako celý svet, na ktorý nepotrebjeme dotaz, región , krajina a administratívne územie. Krajina je administratívne územie, teda spadá nám do dotazu na vyhľadávanie administratívneho územia.

Navrhнемe dva rozdielne dotazy. Jeden na vyhľadávanie administratívneho územia a druhý na vyhľadávanie regiónov.

Vyhľadávanie administratívneho územia

Dotaz na vyhľadávanie administratívneho územia navrhнемe rovnakým princípom ako vyhľadávanie kategórii. Triedu "geografickej lokácie" nahradíme za "administratívny územný celok"(Q56061). Pri vyhľadávaní kategórii bolo potrebne aby nájdene objekty boli triedy. V tomto dotaze potrebujeme vyhľadávať inštancie, preto musíme pozmeniť predikát z wdt:P279* na wdt:P31/wdt:P279*, vďaka ktorému dotaz vyhľadá inštancie.

Po analýze Wikidat je ešte potrebné do dotazu pridať výnimky, teda triedy, ktorých nájdene objekty nesmú byť inštanciou. Sú to "nepolitická administratívna jednotka"(Q15642566) a "historické administratívne územie"(Q19832712), ktoré definuje, že objekt je historické administratívne územie. Dôvod je časť z funkčného požiadavku: "súčasne existujúce územie".

Rovnako ako u možnosti zvoliť pod-kategórie vybratej kategórii ako výnimku, tak aj u administratívneho územia aplikácia umožňuje definovať výnimky ako administratívne územia, ktoré spadajú do vybranej administratívnej oblasti.

Do dotazu pridáme trojice, ktoré sa postarajú o to aby nájdene administratívne oblasti, ktoré používateľ vyhľadáva pre definovanie výnimiek, boli súčasťou vybranej konkrétnej administrovanej oblasti. Na to poslúži predikát "nachádza sa v administratívnej jednotke"(P131).

Zároveň rovnakým princípom pridáme možnosť zahadzovať tie administratívne územia, ktoré spadajú pod administratívne územia, ktoré sú vybraté ako výnimky.

Pridaná časť dotazu vyzerá nasledovne:

```
# zahodenie území, ktoré nespadajú pod  
# administratívne územie Q214  
VALUES ?locatedInAreas {wd:Q214 }  
?item wdt:P131* ?locatedInAreas .  
hint:Prior hint:gearing "forward".
```

```

# zahodenie území, ktoré spadajú pod
# administratívne územie Q181342
MINUS{
VALUES ?notLocatedInAreas {wd:Q181342 }
?item wdt:P131* ?notLocatedInAreas .
hint:Prior hint:gearing "forward".
}

```

Vyhľadávanie regiónu

Dotaz pre vyhľadávanie regiónu začína rovnako použitím MWAPI na vyhľadanie objektov podľa názvu.

Následne musíme v dotaze obmedziť nájdene objekty na to aby boli objektami reprezentujúce región. Regiónom myslíme oblasť do ktorej niektorá krajina patrí. Preto navrhнемe toto obmedzenie nasledovne. Zoberieme všetky objekty, ktoré sú inštanciou "nezávislý štát"(Q3624078) a vyjadrimo, že nájdene objekty musia:

1. patriť do množiny objektov obsahujúcich územia. Tieto územia získame z vlastnosti "patriť"(P361). Objekt reprezentujúci krajinu má vlastnosť "patriť" kde sú ako hodnoty objekty reprezentujúce územie, región, do ktorého daná krajina patrí.
2. objekt je inštanciou "geografický región"(Q82794)
3. alebo objekt je inštanciou "kontinent"(Q5107).

Na spojenie týchto vyjadrení použijeme UNION blok.

Pri analýze dospejeme k záveru, že dotaz vracia aj objekt "planétu Zem"(Q2). Tento objekt nie je vhodný ako región a preto ho v dotaze zahodíme.

Dotaz na vyhľadanie regiónov s klúčom "europe"vyzerá nasledovne:

```

SELECT DISTINCT
?description
(?itemLabel AS ?name)
(?item AS ?QNumber)
WHERE {
# vyhľadanie objektov na základe názvu
SERVICE wikibase:mwapi {
bd:serviceParam wikibase:endpoint "www.wikidata.org";
wikibase:api "EntitySearch";
mwapi:search "europe";
mwapi:language "en" .
?item wikibase:apiOutputItem mwapi:item.
?num wikibase:apiOrdinal true. }
# získanie nezávislých krajín sveta
?country wdt:P31 wd:Q3624078 .
# obmedzenie
# objekt je geograficky región
# a patrí do množiny
# získanej ako hodnoty z vlastnosti
# "súčasť" všetkých nezávislých krajín
}

```

```

{ ?country wdt:P361 ?item.
?item wdt:P31 wd:Q82794.}
UNION
# objekt je kontinent
{ ?item wdt:P31 wd:Q5107.}
# zahodenie objektu "planét Zem"
FILTER (?item not in (wd:Q2))
# získanie popisu regiónu
?item schema:description ?description .
# zahodenie popisov, ktoré nie sú
# v anglickom jazyku
FILTER (lang(?description) = "en")
SERVICE wikibase:label {
bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
?item rdfs:label ?itemLabel . }
} ORDER BY ASC(?num)

```

5.2.7 Vlastnosti ako filtre

Funkčná požiadavka: Používateľ je schopný v procese vyhľadávania objektov definovať nejaké vlastnosti objektu, ktoré musia nájdene objekty splňovať, aby používateľ mohol zadat vlastnosti, ktoré očakáva, že nájdené objekty budú splňovať.

V procese vyhľadávania skupiny objektov od aplikácie je očakávaná možnosť ako parameter definovať vlastnosti s hodnotou, ktoré musia hľadané objekty splňovať.

Každá vlastnosť je nejakého dátového typu a ku každému dátovému typu sa viažu obmedzenia. V návrhu podporujeme iba vlastnosti troch dátových typov. Sú to Quantity, Time a WikibaseItem.

Vyber možných vlastností

Prvý návrh bol aby aplikácia podporovala možnosť vybrať ako filter iba z množiny zopár vlastností, ktoré zanalyzujeme a pridáme do dotazu pre vyhľadávanie skupiny objektov. Tento návrh nie je prakticky a preto sme prišli s dynamickým návrhom. Navrhнемe dotaz, ktorý pre konkrétnu kategóriu získa zoznam vlastností. Vlastnosti v zozname dávajú zmysel pre konkrétnu kategóriu. Napríklad vlastnosť "populácia" sa viaže ku kategórii ako mesto alebo krajina a nie ku hradom. Zoznam nazveme ako "odporúčane vlastnosti" pre danú zvolenú kategóriu.

V tomto probléme nám dosť pomôže vlastnosť "vlastnosti tohto typu" (P1963). Trieda reprezentujúca kategóriu môže mať tuto konkrétnu vlastnosť. V nej nájdeme zoznam vlastností, ktoré sa často spájajú s danou triedou. Urobíme to ale rekurzívne. Zoberieme konkrétnu kategóriu, respektívne triedu, ktorú si používateľ vybral a prejdeme všetky nad-triedy tejto triedy rekurzívne. Pre každú prechádzanú triedu skúsime zistíť ci ma vlastnosť P1963 obsahujúcu vlastnosť, ktoré si uložíme do premennej. Nebudeme však prechádzať všetky triedy. Do úvahy zoberieme iba triedy, ktoré v hierarchii spadajú pod triedu "geografická lokalita". Vďaka tomu dostaneme iba vlastnosti, ktoré sa logicky spájajú ku geografickým lokalitám, čo sú naše kategórie.

V dotaze ešte zahodíme vlastnosti "krajina"(P17) a "nachádza sa v administratívnej jednotke"(P131), pretože tieto vlastnosti používateľ definuje parametrom vyber oblasti, kde sa musia nájdene mapové objekty vyskytovať.

Na koniec je ešte potrebne zahodiť vlastnosti , ktoré nie sú podporovaného dátového typu.

Pre každú vlastnosť získame názov, PNumber ako identifikáciu vlastnosti na použitie v iných dotazov, popis, aby používateľ vedel, čo daná vlastnosť reprezentuje, a dátový typ vlastnosti.

Dotaz na získanie odporúčaných vlastností kategórie "hrady"vyzerá nasledovne:

```
SELECT DISTINCT
?description
(?property AS ?PNumber)
(?propertyLabel AS ?name)
?dataType
WHERE {
# získanie všetkých nad-tried triedy hrad
wd:Q23413 wdt:P279* ?classes .
# triedy musia byť pod-triedou "geografická lokácia"
VALUES ?superClasses {wd:Q2221906 }
?classes wdt:P279* ?superClasses .
hint:Prior hint:gearing "forward".
# získanie všetkých vlastností
# ktoré sa viažu k nejakej z tried
?classes wdt:P1963 ?property .
# zahodenie konkrétnych tried
FILTER (?property not in ( wd:P131 , wd:P17 ))
# zahodenie vlastnosti, ktoré
# nie sú podporovaného dátového typu
VALUES ?supportedDataTypes {
wikibase:Time
wikibase:Quantity
wikibase:WikibaseItem }
?property wikibase:propertyType ?supportedDataTypes .
# získanie dátového typu
?property wikibase:propertyType ?dataType .
# získanie popisu
?property schema:description ?description .
FILTER (lang(?description) = "en")
SERVICE wikibase:label {
bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
?property rdfs:label ?propertyLabel . }
}
```

Problém ale je, že niektoré vlastnosti nedostaneme pre danú kategóriu ako odporúčanú vlastnosť. Napríklad pre kategóriu hrad nedostaneme vlastnosť "pamiatkový status"(P1435), ktorou vieme vyhľadávať hrady patriace k UNESCO pamiatkam.

Musíme preto umožniť používateľovi zvolať danú vlastnosť nielen zo zoznamu odporúčaných filtrov ale aj zo zoznamu, ktorý označíme ako "všetky vlastnosti".

Tento zoznam samozrejme neobsahuje úplne všetky existujúce vlastnosti na Wikidatach.

Zoberme všetky triedy spadajúce pod triedu "geografická lokalita". Pre každú z tried priradíme všetky hodnoty z vlastnosti "vlastnosti tohto typu"(P1963) do premennej. Nasleduje rovnaký postup ako pri "odporúčaných filtrov". V zozname "všetky vlastnosti" nájdeme už aj napríklad vlastnosť "pamiatkový status".

Tento dotaz je ale veľmi náročný. Trvá veľmi dlho kým skončí, dokonca ak je WDQS zafažene nemusí nedobe hnúť do stanoveného limitu. Kedže ale tento zoznam je rovnaký pre každú zvolenú kategóriu, tak výsledok tohto dotazu si uložíme na server a keď aplikácia požiada server o tento zoznam, tak server vráti rovno výsledok z uloženého súboru.

5.2.8 Obmedzenia vlastnosti

Každá vlastnosť obsahuje svoje špecifické obmedzenia, ktoré je potrebné získať aby aplikácia umožnila používateľovi zadávať validnú hodnotu vlastnosti. Tieto obmedzenia sú vyjadrené ako hodnoty vlastnosti "obmedzenie"(P2302) a sú nevyhnutné pre správne vyhľadávanie a zadávanie hodnôt. Obmedzenia sú zároveň pre každý dátový typ rozdielne. Preto je potrebné získavať rozdielne obmedzenia na základe dátového typu.

Aplikácia podporuje tri dátové typy.

1. Time - pre tento dátový typ nie je potrebne získavať obmedzenia, pretože používateľ zadáva dátum
2. Quantity - používateľ zadáva desatinne číslo a preto je potrebne získavať povolený rozsah tohto čísla, teda minimálnu a maximálnu povolenú hodnotu čísla. Číslo môže byt vyjadrené používateľom v rôznych jednotkách. Z toho dôvodu je potrebné získavať zoznam povolených jednotiek, ktorými môže byt desatinne číslo vyjadrené v konkrétnej vlastnosti.
3. WikibaseItem - používateľ ako hodnotu zadá iný objekt z Wikidat, ten ale musí si vyhľadať podla názvu. V tomto vyhľadávaní nechceme umožniť vyhľadávať všetky objekty. Je vhodné vyhľadávať iba tie, ktoré môžu byt použité ako hodnota pre tuto vlastnosť. Pre to je potrebné získavať z obmedzení informácie o obmedzeniach definujúce množinu objektov.

Quantity obmedzenia

Pre kvantitatívne vlastnosti aplikácia potrebuje dva dotazy na obmedzenia. Jeden pre získanie obmedzení reprezentujúce maximálnu a minimálnu povolenú hodnotu a druhý pre získanie zoznamu povolených jednotiek.

Maximum a minimum, danej hodnoty vlastnosti, sú vyjadrené ako kvantifikátory "minimum value"(P2313) a "maximum value"(P2312) v hodnote "range constraint"(Q21510860).

Dotaz pre získanie rozsahu pre vlastnosť "nadmorská výška"(P2044) vyzerá nasledovne:

```
SELECT ?min ?max  
WHERE {
```

```

# získanie uzlu
# daný uzol bude použitý ako predmet
wd:P2044 p:P2302 ?node .
# získanie objektu obsahujúci povolený rozsah
?node ps:P2302 wd:Q21510860 .
# získanie hodnoty kvantifikátoru minimálna hodnota
?node pq:P2313 ?min .
# získanie hodnoty kvantifikátoru maximálna hodnota
?node pq:P2312 ?max .
SERVICE wikibase:label {
  bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
}
}

```

Zoznam povolených jednotiek nájdeme ako hodnoty v kvantifikátore "item of property constraint"(P2305) v hodnote "allowed units constraint"(Q21514353).

Dotaz pre získanie povolených jednotiek pre vlastnosť "nadmorská výška"(P2044) vyzera nasledovne:

```

SELECT (?item AS ?QNumber) (?itemLabel AS ?name)
WHERE {
  # získanie uzlu
  # daný uzol bude použitý ako predmet
  wd:P2044 p:P2302 ?node .
  # získanie objektu obsahujúci povolené jednotky
  ?node ps:P2302 wd:Q21514353 .
  # získanie hodnoty kvantifikátoru
  ?node pq:P2305 ?item .
  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
    ?item rdfs:label ?itemLabel .
  }
}

```

WikibaseItem obmedzenia

Pre vlastnosti tohto dátového typu bude aplikácia potrebovať štyri dotazy. V každom jednom z nich získame iný druh obmedzenia, ktoré obmedzuje množinu objektov možných použiť ako hodnotu pre danú vlastnosť.

Nasleduje výpis týchto obmedzení:

1. "one-of constraint"(Q21510859) - obsahuje zoznam konkrétnych dovolených objektov, ktoré môže hodnota danej vlastnosti nadobúdať
2. "none-of constraint"(Q52558054) - obsahuje zoznam konkrétnych objektov, ktoré by nemali byť použité s danou vlastnosťou
3. "conflicts-with constraint"(Q21502838) - obsahuje zoznam tried a vzťahov, ktoré definujú, že objekt ako hodnota s danou vlastnosťou by nemal byť vo vzťahu s triedami so zoznamom. Vzťah môže byť buď "je inštanciou", "je pod-triedou" alebo "je inštanciou alebo pod-triedou".

4. "value-type constraint"(Q21510865) - obsahuje zoznam tried a vzťahov, ktoré definujú to, že objekt ako hodnota s danou vlastnosťou by mal byť vo vzťahu s triedami zo zoznamu. Vzťahy sú v tomto kontexte rovnaké ako vzťahy u "conflicts-with constraint".

Dotazy vyzerajú podobne ako dotazy obmedzení u kvantitatívnych vlastností a preto ich neuvedieme. Poznamenajme len, že u obmedzení kde je potrebne získať ešte vzťah, ho získame z kvantifikátoru "relation"(P2309)

5.2.9 Vyhľadávanie hodnôt u filtrov

Pre vlastnosti dátových typov Time a Quantity používateľ zadá dátum alebo desatinne číslo. Ale pri vlastnosti typu WikibaseItem je potrebne aby používateľ zadal objekt, ktorý existuje na Wikidatách. Tento objekt je potrebne vyhľadať podľa názvu a zároveň na toto vyhľadávanie aplikovať obmedzenia vlastnosti.

Výnimkou je ak z obmedzení zistime, že "one-of constraint"obsahuje objekty. V takom prípade aplikácia neumožní používateľovi vyhľadávať objekt, ale umožní vybrať jeden z týchto objektov ako hodnotu vlastnosti.

Ak "one-of constraint"je prázdna množina aplikácia umožní vyhľadávať objekt pomocou dotazu. Dotaz má nasledujúcu štruktúru. Najprv použijeme MWAPI službu pre vyhľadanie objektov na základe poskytnutého textového reťazca reprezentujúceho časť alebo celok názvu. Následne použijeme obmedzenia, ktoré aplikácia získa z iných dotazov.

Pre "value-type constraint"(Q21510865) použijeme nasledujúcu časť dotazu:

```
VALUES ?classRestrictions {<objekty, reprezentujúce triedy zo zoznamu>}  
?item <wdt:vzťah> ?classRestrictions .  
hint:Prior hint:gearing "forward".
```

Za vzťah dosadíme predikát, ktorý získame so zoznamom "value-type constraint". Ten definuje vzťah objektov ku triedam so zoznamu.

Pre "conflicts-with constraint"(Q21502838) použijeme rovnakú časť dotazu ako je uvedená vyššie, ale celu časť zabalíme do MINUS bloku, pretože objekty splňujúce tieto trojice je nutné zahodiť. Pretože hodnota vlastnosti nemôže obsahovať objekty, ktoré sú vo danom vzťahu s týmito triedami. Ukážka časti dotazu:

```
MINUS{  
VALUES ?classExceptionRestrictions {<objekty, reprezentujúce triedy zo zoznamu>}  
?item <vzťah> ?classExceptionRestrictions .  
hint:Prior hint:gearing "forward".  
}
```

Pre "none-of constraint"(Q52558054) stačí použiť nasledujúci filter v dotaze:

```
FILTER (?item not in (<objekty>))
```

Objekty sú zo zoznamu "none-of constraint".

5.2.10 Vyhľadávanie skupiny mapových objektov na základe parametrov

Funkčná požiadavka: Používateľ je schopný vyhľadať skupinu objektov, ktoré splňajú všetky parametre zadané používateľom, aby si používateľ mohol vyhľadať skupinu objektov na základe svojich preferencii, ktoré si môže následne uložiť do svojej kolekcie.

V predošlých sekciách sme navrhli dotazy, ktoré umožnia používateľovi aplikácie vyhľadať rôzne parametre pre vyhľadávanie skupiny objektov. Teraz navrhнемe ako ma vyzerá dotaz na získavanie skupiny objektov na základe zvolených parametrov.

Na začiatku je potrebné definovať ako údaje je potrebne o každom objekte získať. Pre každý objekt získame nasledovné údaje:

1. QNumber - identifikátor na Wikidatach a v našej databáze aplikácií. Potrebný pre ďalšie funkcie aplikácie, ktoré komunikujú s Wikidatami ako získavanie detailov o mapovom objekte.
2. name - Názov mapového objektu
3. long - zemepisná výška, pre potrebu vykresliť objekt na mape
4. lati - zemepisná šírka, pre potrebu vykresliť objekt na mape
5. subTypeOf - zoznam tried. Dany mapový objekt je inštanciou týchto tried. Používateľ vďaka tomu lepšie pochopí čo je daný mapový objekt. Pre potrebu mať tieto triedy ako jednu hodnotu, v dotaze použijeme agregačnú funkciu GROUP CONCAT a jednotlivé hodnoty oddelíme separátorom.

Základný dotaz pre získanie mapových objektov vyzerá nasledovne:

```
SELECT DISTINCT (?itemLabel AS ?name)
(GROUP_CONCAT(
DISTINCT ?instancesLabel;separator="/""
) AS ?subTypeOf)
(SAMPLE(?lon) AS ?long)
(?item AS ?QNumber)
(SAMPLE(?lat) AS ?lati)
WHERE {
?item wdt:P31/wdt:P279* wd:Q2221906 .
hint:Prior hint:gearing "forward".
# získanie long a lati
?item p:P625 ?coord .
?coord psv:P625 ?coord_node .
?coord_node wikibase:geoLongitude ?lon .
?coord_node wikibase:geoLatitude ?lat .
# získanie subTypeOf
?item wdt:P31 ?instances .
SERVICE wikibase:label {
bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
# získanie názvu objektu
?item rdfs:label ?itemLabel .
```

```
# získanie názvu jednotlivých tried
?instances rdfs:label ?instancesLabel . }
} GROUP BY ?itemLabel ?item
```

Tento dotaz by mal nájsť všetky mapové objekty. Samozrejme nedobehne do stanoveného limitu, pretože všetkých mapových objektov existuje obrovské množstvo a služba nestihne prejsť všetky na to aby získala potrebné údaje.

Kým nie je zvolená kategória, zároveň aplikácia umožní používateľovi zadať aj "vyhľadať všetko", tak objekt musí byt potomkom "geografická lokalita". Táto dedičnosť definuje mapový objekt.

V nasledujúcich sekciách navrhнемe ako jednotlivé parametre pridať do dotazu.

Kategória

Kategóriu pridáme do dotazu tak, že v pôvodnom dotazu nahradíme za triedu "geografická lokalita" triedu, ktorá reprezentuje vybranú kategóriu ako parameter.

```
?item wdt:P31/wdt:P279* wd:<QNumber kategórie > .
hint:Prior hint:gearing "forward".
```

Táto trojica nám zaručí, že objekty spadajú do vybranej kategórie.

Ako už sme spomenuli, aplikácia musí umožniť používateľovi zadať aj kategóriu ako výnimku. Nájdene objekty nesmú spadať do týchto kategórii reprezentujúcich výnimky.

Tento problém vyriešime rovnako ako u kategórii. Tuto časť dotazu, ale obalíme MINUS blokom. Ten zahodí všetky objekty, ktoré sú potomkami kategórii zvolených ako výnimky.

Ukážka časti dotazu, kde za "výnimky" nahradíme zoznam všetkých tried reprezentujúcich výnimky kategórii.

```
MINUS{
VALUES ?exceptionClasses {<výnimky> }
?item wdt:P31/wdt:P279* ?exceptionClasses .
}
```

Oblast' vyhľadávania

Naša aplikácia umožňuje štyri možnosti ako definovať oblasť vyhľadávania. Sú to : celý svet, región, administratívne územie a okolie okolo vybraného bodu.

Pre celý svet netreba do dotazu nič pridávať, pretože tým pádom sú vyhľadávané všetky objekty. Pre ostatné prípady navrhнемe jednotlivé časti dotazu v nasledujúcich sekciách, ktoré obmedzia vyhľadávanie.

Oblast' ako administratívne územie

Každý objekt, ktorý sa dá považovať za "mapový", respektíve je potomkom "geografickej lokácie", sa musí nachádzať na planéte Zem. Inak povedané musí ležať v niektorom z dnešných administratívnych území. Z toho dôvodu majú mapové

objekty vlastnosť "nachádza sa v administratívnej jednotke"(P131), v ktorej je ako hodnota objekt reprezentujúci nejaké administratívne územie. Toto administratívne územie sa nachádza v, spadá pod nejaké vyššie administratívne územie a preto sa na neho dá znova použiť vlastnosť (P131). Takto sa vieme dostať až po územie reprezentujúce štát.

Zistíť, či objekt patri do konkrétneho administratívneho územia sa dá princípom postupnom prechádzaní z jedného objektu na druhý objektov cez vlastnosť P131. Časť dotazu vyzerá nasledovne:

```
?item wdt:P131* wd:<QNumber administratívneho územia> .  
hint:Prior hint:gearing "forward".
```

Rovnako ako u kategórii aj u administratívneho územia aplikácia umožňuje používateľovi zadat územia ako výnimky. To vyriešime úplne rovnako ako u výnimiek kategórii. Stačí nahradíť predikát P31/wdt:P279* za P131*.

Ukážka, kde za nahradíme administratívne územia reprezentujúce výnimky:

```
MINUS{  
VALUES ?areaExceptions {<výnimky>}  
?item wdt:P131* ?areaExceptions .  
hint:Prior hint:gearing "forward".  
}
```

Oblast' ako región

Každý mapový objekt leží na území nejakej krajiny. Tuto krajinu vieme získať postupne cez vlastnosť P131 až dokým nedôjdeme na objekt, ktorý je krajinou. To je ale zbytočný spôsob a pridáva na čase skončenia dotazu. Mapový objekt má vlastnosť "krajina"(P17), kde už je ako hodnota rovno nezávislý štát.

Štát je následne súčasťou niečoho väčšieho ako napríklad Európy. Pojmom "väčším"myslime región. Ten vieme získať cez vlastnosť "súčasť"(P361) z objektu reprezentujúci nezávislý štát. Zistit či mapový objekt sa nadchádza v Európe by sa dalo nasledujúcim dotazom:

```
# Q46 je Európa  
?item wdt:P17/wdt:P361* wd:Q46 .  
hint:Prior hint:gearing "forward".
```

Toto riešenie je korektné, ale je príliš pomalé. Pre každý objekt je potrebne rekurzívne prejst objekty a rozhodnúť, či je niektorý z nich konkrétny zadaný región ako parameter. Toto riešenie často krát ani nedobeňe do stanoveného limitu, pretože je príliš náročne.

Preto sme rozmyšľali tuto možnosť zahodiť, no nakoniec sme navrhli rýchlejšiu možnosť. Zoberme tento problém z opačnej strany. Nájdime rekurzívne množinu všetkých objektov, ktoré sú súčasťou P361 zvoleného regiónu.

Tuto množinu obmedzme iba na objekty, ktoré sú zároveň inštanciami triedy "krajina"(Q6256). Tuto náročnú operáciu stačí počas behu dotazu urobiť iba raz. Teraz pre každý mapový objekt stačí rozhodnúť,či krajina získaná z vlastnosti

P17 objektu patri do množiny štátov patriacich do konkrétneho regiónu získanú ako je uvedené vyššie.

Ukážka časti dotazu:

```
# objekty, ktoré sú súčasťou Európy  
?countries wdt:P361* wd:Q46 .  
# objekt je zároveň krajina  
?countries wdt:P31 wd:Q6256 .  
# získanie krajiny kde sa objekt nachádza  
?item wdt:P17 ?temp  
# zahodenie objektov, ktorých  
# získaná krajina nepatri do množiny  
FILTER (?temp in ( ?countries ))
```

Tento spôsob je oveľa efektívnejší a dotaz má šancu dobehnuť.

Oblast okolo bodu

Pre vyhľadávanie mapových objektov v určitom okruhu okolo konkrétneho bodu použijeme službu "Around".

Ukážka použitia služby :

```
SERVICE wikibase:around {  
?item wdt:P625 ?geo .  
bd:serviceParam wikibase:radius "1" ;  
wikibase:center "Point(-0.09 51.505)"^^geo:wktLiteral . }
```

Služba do premennej ?item priradí všetky objekty obsahujúce vlastnosť "geografické súradnice"(P625), ktoré sa zároveň nachádzajú v okruhu okolo zadaného bodu. Parameter "radius" berie číslo vždy iba vyjadrené v jednotke kilometer. Aplikácia podporuje možnosť zadat rádius od 1 km až po 250 km. Je to z dôvodu náročnosti služby. Čím väčšia oblasť tým dlhšie trvá službe aby našla všetky výsledky. Rozhodli sme sa preto pre nastavenie maximálneho limitu na 250 km.

Služba podporuje dva možné spôsoby ako vyjadriť stredový bod. Bud' poskytnutím objektu, ktorý ma vlastnosť "geografické súradnice" a tie budú požíte ako stredový bod, alebo zadanie textového reťazca vo formáte Point(<geografická šírka> <geografická výška>). V dotaze pre zadanie stredového bodu okruhu použijeme druhý spôsob. Aplikácia poskytne dotazu súradnice, ktoré dotaz dosadí do textového reťazca. Tieto súradnice získa aplikácia od používateľa, ktorý vyhľadá konkrétny bod, tento dotaz vráti aj súradnice, alebo posunie mapový bod na mape. Tento bod je na interaktívnej mape posuvný. Knižnica, ktorá bude poskytovať mapu , umožňuje získať súradnice vykresleného bodu a zároveň s týmto bodom aj pohybovať.

Aplikovanie filtrov

V nasledujúcich sekciách je pre každý dátový typ vlastnosti navrhnuté ako do dotazu tento filter aplikovať. Aplikované filtre obmedzujú množinu nájdených mapových objektov, pretože nájdene objekty musia každý jeden aplikovaný filter v dotaze splňať.

Aplikácia poskytuje rozhranie pre vyber konkrétneho filtra a zadanie hodnoty daného filtra. Aplikácia si pre aplikované filtre zapamäta získane údaje identifikujúce vlastnosť jej a hodnotu, ktorú používateľ zadá.

WikibaseItem filter

Pre tuto možnosť je potrebný identifikátor vlastnosti "PNumber" a identifikátor objektu "QNumber", ktorý reprezentuje vybranú hodnotu. Samotný filter je reprezentovaný v dotaze ako jedna trojica, kde predmet je premenná obsahujúca mapové objekty, predikát je "PNumber" vlastnosti a objekt je "QNumber" vybranej hodnoty existujúcej na Wikidatách.

Ukážka časti dotazu s filtrom:

```
# mapový objekt je v gotickom architektonickom štýle  
?item wdt:P149 wd:Q176483 .
```

Používateľ môže aplikovať viacero filtrov tohto dátového typu. Pre každý jeden z nich sa do dotazu zapíše jedna trojica, tak ako je vyššie ukázané.

Time filter

Pre vlastnosti dátového typu obsahujúceho čas údaj je potrebný identifikátor vlastnosti "PNumber" a čas vyjadrený ako textový reťazec vo formáte "yyyy-mm-dd" (rok-mesiak-deň). Textový reťazec môže obsahovať prefix "-", ktorý reprezentuje mínus, teda čas pred Kristom. Zároveň v dotaze môže byť mesiac alebo deň nahradený "00". Časť nahradená "00" je ignorovaná. Napríklad "1000-00-00" sa interpretuje ako 1000 rokov alebo 10 storočí. Bez udania mesiaca a dňa v tom roku.

Ukážka časti dotazu filtrejúca mapové objekty vniknuté až po 10. storočí:

```
# "dátum vzniku" (P571)  
?item wdt:P571 ?timeFilterTemp .  
FILTER ("1000-00-00"^^xsd:date < ?timeFilterTemp )
```

Najprv získame časovú hodnotu z vlastnosti mapového objektu a tu následne porovnáme vo Filter bloku. Za textom reprezentujúcim čas nasleduje postfix xsd:date. To je nutné z toho dôvodu aby dotaz tento text reprezentoval ako dátum. Bez použitia tohto postfixu dotaz interpretuje text ako čistý text nereprezentujúci čas.

Quantity filter

Pre vlastnosti kvantitatívneho dátového typu je potrebný identifikátor vlastnosti "PNumber", hodnotu ako desatinné číslo a "Qnumber" jednotky, v ktorej je hodnota vyjadrená.

Prvým krokom je získanie prevodu danej jednotky do základnej jednotky za pomoci predikátu "prevod na jednotku SI" (P2370).

Následne získame hodnotu vlastnosti mapových objektov, ale je potrebné danú hodnotu mať vyjadrenú v základnej jednotke. Na to slúži prefix psn:, ktorý vráti

uzol normalizovanej hodnoty vlastnosti. Inak povedané vráti kvantitatívnu hodnotu vyjadrenú v základných jednotkách. Konkrétnu číselnú hodnotu získame zavolaním wikibase:quantityAmount na tento uzol.

Ako poslednú vec je potrebne vytvoriť blok Filter obsahujúci rovnici s porovnávacím operátom v strede, kde na ľavej strane máme zadanú hodnotu používateľom vynásobenú hodnotou prevodu a na pravej strane máme konkrétnu číselnú hodnotu vlastnosti vyjadrenú v základných jednotkách. Aplikácia poskytne používateľovi vyber porovnávacieho operátora, ktorý sa dosadí do dotazu.

Ukážka časti dotazu ktorá filtriuje objekty, ktoré majú nadmorskú výšku "nadmorská výška" (P2044) vyššiu ako 8 km:

```
# kilometer, predikát na prevod, premenná  
wd:Q828224 wdt:P2370 ?quantityTempconversion .  
# získanie číselnej hodnoty vlastnosti  
# "nadmorská výška" v základných jednotkách  
?item p:P2044/psn:P2044/wikibase:quantityAmount ?quantityTemp .  
# samotná filtrácia  
FILTER ((8 * ?quantityTempconversion) < ?quantityTemp)
```

Nie každá vlastnosť obsahuje hodnotu, ktorá sa vyjadruje v jednotkách. Napríklad hodnoty vlastnosti "populácia" nie sú vyjadrene v žiadnych jednotkách. Sú vyjadrené iba ako číslo udávajúce počet. V takom prípade nemusíme zistovať žiadnen prevod, hodnotu konvertovať pomocou prefixu do normalizovanej formy a v Filter bloku násobiť uvedenú hodnotu prevodom.

5.3 Štruktúra projektu

V root adresári projektu nájdeme podadresáre backend a world-collection-app. V prvom adresári je implementovaná časť aplikácie bežiaca na serveri, teda samotný backend. V druhom adresári nájdeme zdrojové a konfiguračné súbory pre implementáciu frontendu.

Zároveň tu nájdeme Powershell dokumenty initServer a startServer. V initServer je napísaná postupnosť príkazov, ktoré vytvoria virtuálne prostredie, nainštalujú jednotlivé závislosti a nakoniec vytvoria databázu. Na druhej strane startServer ulahčí spúšťanie serveru, pretože svojimi príkazmi sa prepne do virtuálneho prostredia a v ňom zavolá príkaz na spustenie backendu.

V nasledujúcich dvoch sekciách si popíšeme implementáciu backendu a frontendu.

5.4 Backend

Všetky adresáre a dokumenty spomínane v tejto sekcií sa nachádzajú v adresári backend/server.

Súbor __init__.py je vstupným bodom časti aplikácie bežiacej na serveri. V tomto súbore je aplikácia inicializovaná, pomocou takzvaných "blueprints" sa k nej pripoja časti reprezentujúce REST API a zároveň sa do aplikácie pridá

konzolový príkaz na inicializáciu databázy. Tento príkaz sa zavolá spustením dokumentu initServer.ps1. Po tom ako je všetko inicializované a databáza je vytvorená sa zavolaním dokumentu startServer.ps1 spustí aplikácia, ktorá pri lokálnom spustení pobeží na porte s číslom 5000.

Okrem spomínaného vstupného bodu v tomto adresári nájdeme ďalšie dva podadresáre.

V podadresári API nájdeme súbory WorldCollectionAPI.py a WikidataAPI.py, reprezentujúce dokopy REST API. Tie používajú ďalšie súbory, ktoré ale popíšeme neskôr pri opisovaní jednotlivých časti REST API.

V podadresári Data nájdeme dokument AllFilters v Json formáte, ktoré v určitom prípade pošle server na užívateľskú stranu aplikácie. Tento dokument existuje z dôvodu, že dátu ktoré obsahuje sa nemusia zakaždým získať z Wikidata. SPARQL Dotaz na získanie týchto dát je časovo náročný na zbehnutie a vždy vráti rovnaký výsledok. Preto sme tento SPARQL dotaz raz spustili a jeho výsledok si uložili do tohto dokumentu.

5.4.1 Vytváranie SPARQL dotazov

Jednotlivé dotazy by bolo možné implementovať nasledujúcim spôsobom. Pre každý jeden dotaz vytvoríme jeden textový reťazec ako dátový typ string, do ktorého na určite miesta dosadíme vstupné parametre zadane používateľom. To je ale nepraktické z dôvodu, že nevieme následne SPARQL dotazy rozšíriť a pre každý ďalší novy dotaz je potrebné písat cely text dotazu odznova. Z toho dôvodu vytvoríme hierarchiu buildorov, ktoré budú od seba dedit a každý jeden z nich pridá novu funkčnosť. Týmto docielime možnosť rozšírenia a zbytočnej duplicity textu dotazov. Všetky uvedené buildre na tvorbu SPARQL dotazov nájdeme v adresári API/WikidataQueries. Každý builder je definovaný v samostatnom súbore s rovnakým názvom ako nesie daný builder.

Na samotnom vrchole hierarchii stoji základný builder "QueryBuilder", ktorý je schopný generovať základy SPARQL dotazu. Táto trieda je abstraktná a obsahuje abstraktnú metódu potrebnú implementovať v jednotlivých potomkoch. Táto abstraktná metóda reprezentuje vnútro WHERE bloku, ktoré ostatné buildre prepíšu pre svoje vlastné použitie. Trieda obsahuje funkčnosť na zostrojenie bloku SELECT, do ktorého sa pridajú premenne pridané konkrétnou metódou triedy. Ďalej obsahuje možnosť pridať trojice, obalif vyraz, respektívne časť trojíc, do blokov ako napríklad MINUS, FILTER alebo SERVICE, priradiť konkrétné hodnoty do konkrétnej premennej a ďalšie užitočné metódy pre prácu so SPARQL. Je doležité zmieniť, že tento builder zároveň obsahuje možnosť pridať "label službu" na získavanie názvov. Ak nie je možné použiť túto službu v automatickom režime, tak táto trieda obsahuje mechanizmu pre automatické pridanie manuálneho režimu. Manuálny režim sa pridá automaticky a programátor ho nemusí riešiť. Metódou build sa zástoji daný dotaz.

Builder-e dedičace od základného buildru je možné rozdeliť do nasledujúcich skupín:

- Search - vyhľadávanie na základe názvu objektu
- CollectibleDetails - dotazy, ktoré získajú o mapovom objekte ďalšie informácie

- CollectibleSearch - dotazy, ktoré vyhľadajú množinu mapových objektov na základe zadaných parametrov
- Filter - dotaz na získanie odporúčaných filtrov pre danú kategóriu a dotaz na získanie dát o obmedzení vlastnosti

Search skupina obsahuje ako základ “SearchQueryBuilder”. Ten pridáva schopnosť do dotazu aplikovať MWAPI EntitySearch službu pre možnosť vyhľadávania objektov na základe ich názvu. Od tejto triedy dedia ďalšie triedy pre konkrétnu vyhľadávanie, či už vyhľadávanie kategórii, regiónov alebo objektov reprezentujúce hodnoty filtru. Tieto triedy pridávajú do dotazu schopnosť vytvoriť dotaz na konkrétnu vyhľadávanie, inak povedane pridanie konkrétnych trojíc do dotazu aby dotaz vyhľadával tu správnu množinu objektov.

CollectibleDetails pozostáva z troch builderov. WikipediaLinkQuery na získanie odkazu na Wikipédiu, CollectibleDetailsQuery na získanie obrázku a popisu mapového objektu a CollectibleBasicInfoQuery na získanie detailov, informácií o mapovom objekte. Tieto tri buildre berú ako parameter iba QNumber objektu, pre ktorý vytvoria dotaz na získanie potrebných dát.

Základom skupiny CollectibleSearch je “CollectiblesSearchQuery”, ktorý obsahuje funkcionality pridania do dotazu trojice na získanie mapových objektov a možnosť pridávať obmedzenia vlastnosti. Inými slovami pridávať filtre, ktoré obmedzujú nájdenú skupinu objektov. Od tohto builderu dedia štyri buildre, ktoré sa delia na základe definovania oblasti kde sa mapové objekty vyhľadávajú. Každý z nich pridá vlastnú časť dotazu, ktorá obsahuje trojice na definovanie a obmedzenie oblasti.

5.4.2 Posielanie a získavanie dát z Wikidata

Všetky vyššie spomínané buildere zakonštruuju iba text dotazu. Ten je následne potrebné poslať na WDQS koncový uzol. Na to použijeme spomínanú knižnicu SPARQLWrapper. Tá nám odošle dotaz na koncový uzol WDQS a zároveň nám spracuje vrátený výsledok, ktorý nastavíme aby bol vrátený vo formáte JSON. V súbore SparqlPoint je funkcia get_query_results, ktorá berie dva vstupné parametre a vráti dátu v JSON formáte. Prvým vstupným parametrom je URL koncového uzlu a druhý je text SPARQL dotazu, ktorý bude odoslaný na daný koncový uzol.

Ukažka funkcie:

```
def get_query_results(endpoint_url: str, query: str):
    # adjust user agent; see https://w.wiki/CX6
    user_agent = "WorldCollection/%s.%s" % (
        sys.version_info[0], sys.version_info[1])
    sparql = SPARQLWrapper(endpoint_url, agent=user_agent)
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    result = sparql.query().convert()
    return result
```

Výsledok v JSON formáte je pre naše účely príliš komplexný a obsahuje niektoré dátu v podobe, z ktorej je potrebne dátu ešte pred používaním našou aplikáciou vyčistiť. Tým myslime, že sú dátu uvedene ako celá URL a naša aplikácia

pre svoje potreby potrebuje iba časť tejto URL. Z toho dôvodu ešte pred tým ako sa dáta pošlú užívateľovi na frontend, tieto dáta konvertujeme do nášho JSON formátu. Funkcia na prevod do nášho JSON formátu je definovaná v súbore Formatter.py.

Ukážka funkcie:

```
def toJson(input):
    data = input['results']['bindings']
    formatted_data = []
    for item in data:
        d = collections.OrderedDict()
        for bindings_in_item in item:
            type = item[bindings_in_item]['type']
            value: str = item[bindings_in_item]['value']
            match type:
                case "literal":
                    d[bindings_in_item] = value
                case "uri":
                    if bindings_in_item == "image" or
                    bindings_in_item == "article":
                        d[bindings_in_item] = value
                        break
                    temp: str = value.split('/')[-1]
                    if temp.startswith("ontology#"):
                        temp = temp.lstrip("ontology#")
                    d[bindings_in_item] = temp
        formatted_data.append(d)
    return json.dumps(formatted_data)
```

5.4.3 Databáza

Na serveri je potrebné mať databázu, v ktorej budú uložené mapové objekty patriace do kolekcie a samotné kolekcie vytvorené používateľom. Pre nás projekt použijeme databázový systém SQLite, ktorý je pre naše účely dostačujúci. Tento systém je obsiahnutý v Pythone. Samotné dáta reprezentujúce databázu sú uložená na serveri v adresári backend/instance. Čítanie a zapisovanie do databázy prebieha pomocou jazyka SQL.

V kapitole návrh v sekcií návrh databázy sú popísané údaje potrebné pre ukladanie do databázy. Z toho dôvodu si ich nebudeme tu už popisovať.

Z návrhu vyplýva, že v databáze budú dva entity. Jedna bude reprezentovať kolekciu a druhá mapový objekt. Najjednoduchšie je vytvoriť tri tabuľky, kde v prvej budú uložené dátové prvky reprezentujúce kolekcie, v druhej jednotlivé mapové objekty a v tretej vzťahy medzi kolekciami a mapovými objektami. Inak povedané tretia tabuľka bude obsahovať údaje o tom, že daný mapový objekt patri do danej kolekcie. Mapový objekt môže patriť do viacerých kolekcií a teda v tejto tabuľke by sme našli aj viacero riadkov obsahujúcich rovnaký mapový objekt. Dôležité je obmedziť aby žiadnen konkrétny mapový objekt nepatril dvakrát do rovnakej kolekcie. Pre toto riešenie je následne pri čítaní a zapisovaní databázy potrebne urobiť join na tabuľky cez tabuľku obsahujúcu vzťahy, aby sa nám informácie spojili a vedeli sme povedať, do akej kolekcie patri mapový objekt.

V našej implementácii však urobíme iba dve tabuľky a tým nebudeme musieť používať join na tabuľky.

Schémy

Jedna tabuľka je pre kolekcie, ktorá si drží meno a ID kolekcie. Druhá tabuľka je pre mapové objekty patriace do konkrétnnej kolekcie. Každý riadok v druhej tabuľke drží nielen dátu potrebné pre reprezentáciu mapového objektu, ale aj ID kolekcie do ktorej patri.

Každá kolekcia ma unikátny identifikátor, čo reprezentuje collection_id. Pre tento identifikátor nastavíme aby sa automaticky inkrementoval pri vytváraní novej kolekcie. Tým docielime unikátnosť a tuto hodnotu označíme ako primárny kľúč. Každá kolekcia má zároveň aj svoj názov, respektíve meno. Je potrebne aby každá kolekcia mala unikátné meno a žiadne dve kolekcie sa nevolali rovnako. Preto hodnotu name obsahujúcu názov kolekcie nastavíme aby bola unikátna.

Schéma pre vytvorenie tabuľky obsahujúcej kolekcie vyzerá nasledovne:

```
DROP TABLE IF EXISTS Collections;

-- Schema for creating a new table, which will
-- contain collections data
CREATE TABLE Collections(
--a unique ID of collection
collection_id INTEGER PRIMARY KEY AUTOINCREMENT,
-- a unique name of collection
name TEXT UNIQUE NOT NULL
);
```

Každý mapový objekt musí obsahovať tiež svoj unikátny identifikátor. Ako už sme spomínali v návrhu, použijeme ako identifikátor QNumber z Wikidata. Každý objekt na Wikidatách ma svoj unikátny identifikátor vyjadrený ako QNumber. Zároveň ak budeme potrebovať získať ďalšie informácie o mapovom objekte z Wikidata, tak nám tento QNumber uľahčí prácu, pretože vďaka QNumber vieme na Wikidatách rovno získať objekt na základe jeho QNumber. . Ďalším dôležitým údajom je ID kolekcie, do ktorej mapový objekt patri. Tato hodnota pochádza z tabuľky s kolekciami a preto je v tejto tabuľke ako cudzí kľúč. Primárnym kľúčom v tabuľke s mapovými objektami je dvojica q_Number a collection_id. Tým sme docieli, že konkrétny mapový objekt patriaci do danej kolekcie sa môže nachádzať v tabuľke iba raz. Avšak jeden konkrétny mapový objekt s unikátnym QNumber môže patrili do viacerých kolekcií. V takom prípade sú v tabuľke viacero riadkov, ktoré reprezentujú tento mapový objekt, ale každý z týchto riadkov obsahuje iné id_collection. Ak zmažeme mapový objekt patriaci do danej kolekcie, ale objekt patril zároveň aj do inej kolekcie, tak sa nevymaže z databázy mapový objekt iba riadok reprezentujúci, že patril do danej kolekcie. Aby sa z databázy mapový objekt úplne vymazal, tak je potrebné vymazať všetky riadky s konkrétnym q_number. V operáciach nad databázou môžeme definovať, či pracujeme s mapovým objektom, v takom prípade používame iba q_number a pracujeme so všetkými výskytmi v databáze. Druhou možnosť je pracovať iba s mapovým objektom patriacim do danej kolekcie. Vtedy potrebujeme používať

dvojicu q_number a collection_id. Ak používateľ vymaže mapový objekt z kolekcie, použijeme druhu možnosť a teda sa vymaže iba udaj o tom, že daný objekt patril do danej kolekcie. V prípade ,že používateľ zmení návštevnosť objektu, alebo aktualizuje poznámky, tak v takom prípade sa použije prvá možnosť a tieto dátia sa zmenia všetkým riadkom v databáze, ktoré reprezentuje daný mapový objekt.

Toto riešenie má ale nevýhodu. V riešení s troma tabuľkami máme pre každý mapový objekt iba jeden dátový prvok. V riešení s dvoma tabuľkami môžeme mať konkrétny mapový objekt priradený do N kolekciách a tým pádom tabuľka obsahuje N dátových prvkov, kde každý obsahuje rovnaké údaje až na výnimku identifikátora kolekcie. Táto nevýhoda by sa stratila keby operácie nad databázou používali stále druhu možnosť pristupovania k mapovému objektu a k objektu by pristupovali vždy ako k dvojici QNumber a ID kolekcie. Vtedy by používateľ mohol mať rovnaký mapový objekt vo viacerých kolekciách a pre každý z nich by mohol mať poznačené iné poznámky, nastavenú inú ikonu a nastavenú návštevnosť rozlične. V našej aplikácii toto docielí nechceme a preto použitie iba dvoch tabuľiek nesie so sebou aj túto nevýhodu.

Schéma pre vytvorenie tabuľky obsahujúcej mapové objekty patriace do niektornej kolekcie vyzerá nasledovne:

```
DROP TABLE IF EXISTS Collectibles;

-- Schema for creating a new table, which will contain
-- collectibles content
CREATE TABLE Collectibles(
    q_number INTEGER NOT NULL,
    collection_id INTEGER NOT NULL,
    name TEXT NOT NULL,
    instance_of TEXT NOT NULL,
    latitude REAL NOT NULL,
    longitude REAL NOT NULL,
    is_visited BOOL DEFAULT false,
    visit_date_from DATE DEFAULT NULL,
    visit_date_to DATE DEFAULT NULL,
    visit_date_format TEXT DEFAULT NULL,
    icon TEXT NOT NULL DEFAULT 'default',
    notes TEXT DEFAULT NULL,

    CONSTRAINT PK_Collectible PRIMARY KEY (q_number,collection_id),
    FOREIGN KEY (collection_id) REFERENCES Collections(collection_id)
    ON DELETE CASCADE
);
```

Vďaka nastaveniu ON DELETE CASCADE pre definíciu cudzieho kľúča, docielime to, že v prípade ak sa zmaže kolekcia z jednej tabuľky, tak všetky mapové objekty v druhej tabuľke patriacej do tejto zmazanej kolekcie sa tiež vymazú. To nastane automaticky a nie je potrebné nič implementovať navyše.

Prístup a inicializácia databázy

Pred tým ako sa prvý krát spustí backend aplikácia na serveri je potrebné inicializovať databázu. Inak povedané vytvoriť tabuľky na základe schém. Ako už bolo spomenuté , príkaz na inicializáciu databázy sa zavolá pri spustení dokumentu initServer.ps1. Ten v sebe obsahuje zavolanie nasledujúceho príkazu:

```
flask --app ./backend/server/ init-db
```

Flask aplikácia v sebe totiž obsahuje konzolový príkaz init-db. Ten je pridaný do aplikácia vďaka Python modulu click. V adresári database/Database_operations v subore db_initiazition.py je funkcia init_db, ktorá vytvorí databázu tým , že spustí prikaz na vytvorenie tabuliek z databázových schém. Túto funkciu zavoláme z funkcie init_db_command, ktorú označíme deklaratórom command aby bola zavolaná príkazom "init-db".

Ukážka funkcie s deklaratórom:

```
@click.command('init-db')
def init_db_command():
    """Clear the existing data and create new tables."""
    init_db()
    click.echo('Initialized the database.')
```

Je potrebne ešte do aplikácie pridať tento príkaz. To spravíme vo funkciu init_app, ktorá vyzerá nasledovne:

```
def init_app(app):
    """Add database init command into app."""
    app.teardown_appcontext(close_db)
    app.cli.add_command(init_db_command)
```

Po tom ako už existuje databáza je potrebné aby s ňou aplikácia naviazala spojenie. To ma nastarosti funkcia get_db nachádzajúca sa v súbore db_access.py.

```
import sqlite3
from flask import current_app, g

def get_db():
    """Return connection to database."""
    if 'db' not in g:
        g.db = sqlite3.connect(
            current_app.config['DATABASE'],
            detect_types=sqlite3.PARSE_DECLTYPES
        )
        g.db.row_factory = sqlite3.Row
    return g.db
```

CRUD operácie

CRUD je akronym pre CREATE, READ, UPDATE and DELETE. Termíny opisujú štyri základne operácie pre vytváranie a spravovanie dátových prvokov v databáze.

V adresári atabase/Database_operations v súbore db_CRUD.py sú implementované všetky potrebné operácie nad databázou, ktoré aplikácia používa pre spravovanie a udržiavanie databázy. Každá z týchto operácií spadá pod jeden termín z CRUD.

- CREATE - operácie vytvárajúce nové dátové prvky. V našej aplikácii sú to operácie ako vytvorenie novej kolekcie a mapového objektu.
- READ - operácie, ktorých úlohou je len čítať dátové prvky z databázy. Sú to operácie získavajúce dátu z databázy. V našej aplikácii sú to operácie na získavanie zoznamu všetkých kolekcií, alebo zoznamu všetkých mapových objektov patriacich do danej kolekcie.
- UPDATE - operácie modifikujúce dátu už existujúcim dátovým prvkom. Sú to operácie, ktoré zmenia dátum návštevy, poznámky alebo ikonku mapového objektu.
- DELETE - operácie, ktoré vymažú dátové prvky. Sú to operácie ako mazanie mapového objektu patriaceho do danej kolekcie alebo zmazanie celej kolekcie.

Aby sme v každej CRUD operácii predišli k neustálemu používaniu toho istého kódu ako je execetovanie SQL príkazov a fetchovanie záznamov z databázy, tak vytvoríme pomocné funkcie, ktoré definujeme v tom istom súbore.

READ operácie vracajú dátu získane z databázy. V prípade, že nastane nejaká chyba počas čítania databázy, tak operácie vrátia výsledok ako dátový typ None. Vďaka tomu budeme môcť ošetriť chybu. Tu ošetríme vo API funkcií, ktorá vola danú CRUD operáciu. Ostatne CRUD operácie vracajú iba status dátové typu boolean. Ak operácia úspešne vykoná daný SQL príkaz funkcia vráti hodnotu true, v opačnom prípade false. Vďaka tomu dokážeme ošetriť prípadné chyby podobne ako u READ operáciách.

CRUD operácií nad databázou v našej aplikácii je celkom dosť a preto si ich nebudeme uvádzat.

5.4.4 REST API

Úlohou REST API je spracovať jednotlivé HTTP požiadavky prichádzajúce zo strany užívateľa, vykonať činnosť daného požiadavku a vrátiť výsledok. Každá požiadavka smeruje na iný koncový uzol serveru. Vďaka knižnici Flask dokážeme jednoducho nasmerovať jednotlivé prichádzajúce požiadavky na konkrétnu funkciu. Stačí nad definíciu funkcie definovať nasledujúci deklarátor:

```
@app.route('<URL, reprezentujúce koncový uzol>')
```

Webové aplikácie používajú rozdielne HTTP metódy pri pristupovaní na URL. Bez udania druhu metódy, každý koncový uzol odpovedá iba na požiadavky s GET metódou. V prípade, že je potrebné aby koncový uzol odpovedal na iný druh požiadavku, tak do deklaratoru route() použijeme argument "methods".

Na koncový uzol môže, ale písť aj požiadavka s parametrami. Tie sú pre našu aplikáciu veľmi doležité, keďže na základe nich sa vykoná práca danej funkcie ako je zstrojenie SPARQL dotazu. Ak príde požiadavka na to aby server

vrátil zoznam všetkých mapových objektov v danej kolekcii, tak je potrebné aby s požiadavkou prišiel parameter definujúci danú kolekciu v podobe čísla reprezentujúceho identifikátor kolekcie. Parametre nájdeme v globálnom objekte request.

Všetky parametre prichádzajúce na koncový uzol chceme mať vyjadrene v Json formáte. Tie získame z request objektu a uložíme si ich do premennej data.

Nasledujúca ukážka kódu získa dátu ako Json prichádzajúce na koncový uzol.

```
data = request.get_json()
```

Tento spôsob získania dát funguje iba pre HTTP požiadavky POST, PUT a DELETE. Pretože je potrebné v hlavičke požiadavky uviesť nasledujúcu hlavičku:

```
headers: {  
    'Content-Type': 'application/json',  
},
```

Bohužiaľ pre metódu GET nevieme takto hlavičku uviesť.

Pre GET je potrebné na strane klienta parametre zakódovať do textového retazca reprezentujúceho data v Json formáte a ten následne poslať ako hodnotu priradenú do nejakého parametra. Na zakódovanie Json dát ako textový retazec použijeme metódu JSON.stringify().

URL na koncové uzly požívajúce GET metódou s parametrami, na konci za URL dodávajú za "?" parametre vo formáte kľuč=hodnota. Preto URL na server s GET metódou vyzerajú nasledovne :

```
<URL koncového uzlu>?data=<Json dátu zakódovane ako string >
```

Na serveri vo funkciu ktorá spracuje požiadavku s Get metódou Json dátu získame z URL nasledovne:

```
data = json.loads(request.args.get("data"))
```

Každý koncový uzol našej aplikácie vracia dátu v Json formáte.

REST API sa v našej aplikácii delí na dve časti podľa funkcionality. WikidataAPI a WorldCollectionAPI. Smerovacie funkcie definovane v týchto dvoch API sú pridane do blueprintov , ktoré sú registrované do aplikácie. Každý blueprint udáva prefix, ktorý je potrebné uviesť pred definíciu koncového uzla. Tento prefix udáva, že pristupujeme k smerovacím funkciám patriacim ku konkrétnnej časti API reprezentujúcej významovo nejaký celok.

WikidataAPI

Smerovacie funkcie a definíciu blueprintu nájdeme v súbore WikidataAPI.py nachádzajúcim sa v adresári API.

Blueprint je definovaný úplne hore v súbore nasledovne:

```
API = Blueprint('WikidataAPI', __name__, url_prefix='/WikidataAPI')
```

Pre každú funkciu pridáme deklarátor @API.route('<koncovy uzol>', methods=['GET']).

Funkcie spadajúce do tohto blueprintu a súboru spolu reprezentujú časť API komunikujúcu s Wikidatami. Každá funkcia odpovedá na metódu GET. Najprv si funkcia získá z request objektu Json dátu, následne overí či obsahujú všetky potrebné údaje pre vykonanie činnosti smerovacej funkcie, zostaví sa konkrétny SPARQL dotaz na základe parametrov za pomoci spomínaných buildorov. Dotaz sa následne odošle na koncový uzol, výsledok sa spracuje, sformátuje do našej reprezentácie v Json formáte a vráti na frontend. Odosielanie, a spracovanie výsledkov je definované vo funkcií process_query(), ktorá berie ako argument textový retazec reprezentujúci SPARQL dotaz a vracia získané dátu z Wikidata.

Ukážka smerovacej funkcie, ktorá sa pokusy získať odkaz na Wikipédia článok daného mapového objektu použitím WikiPediaLinkQuery buildra:

```
@API.route('/get/collectible_wikipedia_link', methods=['GET'])
def get_collectible_wikipedia_link():
    """
    API route function for obtaining collectible's link
    to wikipedia if exists.

    Required json parameters
    -----
    collectible_QNumber : str
        QNumber of collectible, for which it will search link.

    Return
    -----
    JSON formatted str
    Data about found wikipedia link.
    """

    data = json.loads(request.args.get("data"))
    collectible: str = data['collectible_QNumber']
    if collectible is None:
        return "Invalid request, 'collectible_QNumber' parameter must be provided"
    builder = WikiPediaLinkQuery(collectible)
    return process_query(builder.build())
```

Každý koncový uzol s prefixom /WikidataAPI teda vracia nejaké dátá získané z Wikidata. Jedná sa o REST API komunikujúce s Wikidata. Po prefixe /WikidataAPI nasleduje časť URL buď /search alebo /get. To nám rozdeľuje smerovacie funkcie na tie, ktoré využívajú vyhľadávanie na základe zadанého názvu od funkcií získavajúcej informácie na zakladá rôznych parametrov, nie však na základe názvu.

Nasleduje výpis všetkých WikidataAPI koncových uzlov, bez udania potrebných parametrov, s krátkym popisom, aké dátá z Wikidata vrátia.

- /search/collectible_allowed_types - vyhľadá množinu objektov reprezentujúcich kategórie mapových objektov. Používateľ potrebuje vybrať kategóriu pri vyhľadávaní mapových objektov patriacich do konkrétnnej kategórie.

- /search/placesOrCollectibles - vyhľadá konkrétné mapové objekty na základe poskytnutého názvu.
- /search/regions - vyhľadá objekty reprezentujúce "region", ktoré sa následne aplikujú ako hodnota v dotaze pri vyhľadaní objektov nachádzajúcich sa v tomto regióne.
- /search/administrative_areas - vyhľadá objekty reprezentujúce administratívne územie, ktoré sa následne aplikuje ako hodnota v dotaze pri vyhľadaní objektov nachádzajúcich sa v tomto administratívnom území. Zároveň sa tento koncový uzol používa aj pre definovanie územia ako výnimka.
- /search/wikibase_item - vyhľadá objekty na Wikidatach, ktoré sa môžu použiť ako hodnota pre daný filter typu WikibaseItem.
- /search/collectibles - vyhľadá množinu mapových objektov splňujúcich všetky aplikované filtre, patriace do danej kategórii a nachádzajúce sa v definovanej oblasti.
- /get/collectible_wikipedia_link - získa odkaz na Wikipédia článok o danom mapovom objekte.
- /get/collectible_details - získa detaily mapového objektu.
- /get/collectible_basic_info - získa obrázok a popis mapového objektu.
- /get/filter_data - získa dátá reprezentujúce obmedzenia konkrétneho filtra, respektívne "property".
- /get/recomended_filters - získa zoznam odporúčaných filtrov, vhodných na použitie pre zvolenú kategóriu
- /get/collectible_data - získa potrebné dátá o mapovom objekte, v prípade, že poznáme iba QNumber mapového objektu, ale nemáme ostatné dátá potrebné pre reprezentovanie mapového objektu.

WorldCollectionAPI

Smerovacie funkcie a definíciu blueprintu nájdeme v súbore WorldCollectionAPI.py v adresári API.

Blueprint je definovaný v súbore úplne hore nasledovne:

```
API = Blueprint('WorldCollectionAPI', __name__, url_prefix='/WorldCollectionAPI')
```

Pre každú funkciu pridáme deklarator @API.route('<koncovy uzol>', methods=['<metoda: POST , GET , PUT alebo DELETE>']).

Funkcie spadajúce do tohto blueprintu a súboru spolu reprezentujú časť REST API spravujúcu našu aplikáciu. Inak povedané každá smerovacia funkcia obaľuje konkrétnu CRUD operáciu nad databázou aplikačnej vrstvou. Každá funkcia najprv spracuje parametre z HTTP požiadavku, zavolá CRUD operáciu a predá jej parametre. Následne vráti výsledok alebo v prípade vyskytnutia chyby, ošetrí tuto chybu tým, že pošle správu o chybe na frontend.

Po prefixe /WorldCollectionAPI nasleduje časť URL bud' /get, /post , /put alebo /delete. To nám rozdeľuje smerovacie funkcie, podľa druhu HTTP metódy na ktoré , smerovacia funkcia odpovedá a zároveň aký typ CRUD operácie je v nej volaní. Poznámka :pre /put sa volá operácia spadajúca pod pojmom UPDATE.

Ukážka smerovacej funkcie vracajúcej mapové objekty v danej kolekcii:

```
@API.route('/get/collectibles', methods=['GET'])
def get_collectibles_in_collection():
    """
    Obtain data of all collectibles in specific collection.
    Necessary Parameters in request
    -----
    collectionID : str
        ID of collection, from which we want to obtain collectibles.
    Returns
    -----
    JSON formatted str
    Json with array of data containing collectibles.
    None
    If fetching was not successful.
    """
    data = json.loads(request.args.get("data"))
    collection_id = data["collectionID"]
    if collection_id is None:
        return "Invalid request, collectionID=< existing collectionID> must be provided"
    result = db_CRUD.get_all_collectibles_in_collection(collection_id)
    if result is None:
        return "Error, there occurs some problem with getting collectibles from database"
    return result
```

Najprv získame dátu z požiadavku v Json formáte. Overíme či dátu obsahujú udaj o kolekcii. V prípade, že tomu tak nie je, tak vrátíme chybu. Následne zavoláme CRUD operáciu na získanie dátových prvkov reprezentujúcich mapové objekty patriace do danej kolekcie. Na záver overíme, či operácia nad databázou vrátila výsledky a v prípade, že výsledky sú None, tak vrátíme správu o chybe.

Nasleduje výpis všetkých WorldCollectionAPI koncových uzlov, bez udania potrebných parametrov, s krátkym popisom čo je ich úloha:

- /get/collections - získa zoznam všetkých kolekcií v databáze.
- /get/collectibles - získa zoznam všetkých mapových objektov patriacich do danej kolekcie.
- /get/exists_collections - zistí či v databáze existuje kolekcia s daným názvom. Potrebné pri overovaní, či je možné vytvoriť, premenovať kolekciu s použitím takéhoto názvu.
- /post/create_collection - vytvorí záznam reprezentujúci kolekciu

- /post/collectibles - vytvorí záznamy reprezentujúce mapové objekty patriace do danej kolekcie.
- /update/set_visit - aktualizuje dátá reprezentujúce návštevnosť mapového objektu.
- /update/collection_name - aktualizuje dátá reprezentuje meno kolekcie.
- /update/collection_merge - aktualizuje hodnotu collection_id v zázname. Používa sa keď používateľ chce všetky mapové objekty z jednej kolekcie presunúť do inej. V takom prípade stačí aktualizovať iba hodnotu reprezentujúcu kolekciu, do ktorej objekt patri.
- /update/collectible_name - aktualizuje dátá reprezentuje meno mapového objektu.
- /update/collectible_icon - aktualizuje dátá reprezentuje názov obrázka ikonky mapového objektu.
- /update/collectibles_in_collection_icons - aktualizuje dátá reprezentujúce názov obrázka ikonky všetkých mapových objektu patriacich do danej kolekcie. Používanie v prípade, keď používateľ chce nastaviť jedným kliknutím obrázok ikonky pre všetky mapové objekty v danej kolekcie.
- /update/collectible_notes - aktualizuje dátá reprezentujúce poznámky mapového objektu.
- /delete/collection - zmaže záznam reprezentujúci danú kolekciu a všetky mapové objekty patriace do nej.
- /delete/collectible - zmaže záznam reprezentujúci konkrétny mapový objekt patriaci do konkrétnej kolekcie

5.5 Frontend

5.5.1 Funkcionálne komponenty

Ako už bolo spomenuté, aplikácia s použitím Reactu sa zakladá z komponent. Každá komponenta predstavuje časť užívateľského rozhrania, ktorá ma svoju vlastnú logiku a vzhľad. Komponenty by mali byť usporiadane hierarchicky. Teda užívateľské rozhranie pozostáva z viacerých komponent a tie sa delia na ďalšie pod-komponenty. Komponentu môže byť implementovaná dvoma spôsobmi.

Najjednoduchší spôsob je komponentu definovať ako javascriptovú, v našom prípade typescriptovú funkciu. Táto funkcia berie jeden argument "props", označovaný ako vlastnosti. V nich sa nahadzujú všetky dátá vstupujúce do funkcie. V implementácii využívajúcej jazyk Typescript je potrebné tieto props definovať ako rozhranie. Funkcia vracia základný element knižnice Reactu, čo je JSX.Element využívajúci spomínanú syntax JSX. Syntax budeme používať pre každú návratovú hodnotu funkcie reprezentujúcu react komponentu.

Druhou možnosťou je komponentu implementovať ako triedu. Každá takáto trieda musí dediť od triedy React.Component a obsahovať metódu render() vracajúcu JSX.Element.

Pre obe možnosti platí, že každá komponenta by mala byť definovaná v samostatnom súbore. Ten kvôli použitiu jazyka Typescript bude mať koncovku tsx.

Ukážka komponenty vyjadrenej ako funkcionálna komponenta v jazyku Typescript:

```
export interface MyButtonProps {
    buttonText: string
}

export function MyButton({ buttonText: string }: MyButtonProps) {
    return (
        <button>{buttonText}</button>
    );
}
```

Klúčové slovo "export" sa používa z dôvodu aby bolo možné volať tuto funkciu z iných súborov. Do syntaxe JSX môžeme pomocou zátvoriek "" pridať aj typescriptový kód. Ten vďaka týmto zátvorkám je braný pri preklade ako kód. V ukážke sme to použili aby sme mohli do textu tlačidla zadať hodnotu nachádzajúcu sa v premennej. Bez použitia týchto zátvoriek by tlačidlo obsahovalo text "buttonText".

V našej implementácii aplikácie budeme komponenty definovať pomocou funkcií. V týchto funkciách budeme taktiež používať funkcie označované ako "Hooks".

Hooks

Hooks je označenie pre funkciu začínajúcu prefixom use. Tieto funkcie sú zabudované a poskytované samotným Reactom. Účelom týchto funkcií je spravovať stav, životný cyklus komponenty bez toho aby bolo potrebne používať triedy. Dve najpoužívanejšie hooks funkcie sú useState a useEffect.

Hook funkcia useState sa používa na zapamätyvanie si nejakého stavu komponenty, respektíve nejakej hodnoty reprezentujúcej stav. Hook funkcia useEffect umožňuje komponentu pripojiť a synchronizovať s externými systémami. Tento druh Hooks budeme používať napríklad pre fetchovanie dát z beckendu, potom ako sa prvý krát vykreslí komponenta.

Hooks sa dajú používať iba vo funkcionálnej komponente, nejde ich použiť v triedovej komponente.

Príklad použitia useState pre pamäťanie si aký index je zvolený v komponente:

```
function ImageGallery() {
    const [index, setIndex] = useState(0);
    // ...
}
```

5.5.2 Vytvorenie React aplikácie

Pre vytvorenie novej aplikácie je doporučene použiť prostredie Create React App. Projekt reprezentujúci frontend časť našej aplikácie vytvoríme a nakonfigurujeme zavolaním príkazu:

```
npx create-react-app world-collection-app  
--template typescript --use-npm
```

Tento príkaz nám vytvorí všetky potrebné adresáre a dokumenty, ktoré následne nájdeme v adresári world-collection-app. Po vytvorení aplikácie ju môžeme spustiť z adresáru world-collection-app príkazom npm start. Po zavolaní tohto príkazu bude aplikácia bežať na adrese localhost:3000.

Knížnice pridáme do projektu nasledovným formátom príkazu: npm install <názov knižnice> –save.

5.5.3 Implementácia World collection

V tejto sekcii si prejdeme hierarchiu a štruktúru komponent, ktoré spolu tvoria celu frontend časť aplikácie.

Podadresár Public

V podadresári public nájdeme koreňový súbor index.html. Tento súbor nebudeme skoro vôbec nie ako upravovať, jedine do neho pridáme nasledujúci odkaz:

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.2/dist/leaflet.css"  
integrity="sha256-sA+zWATbFVeLLNqW02gtiw3HL/lh1giY/Inf1BJ0z14="  
crossorigin="" />
```

Tento odkaz je potrebné pridať do koreňového HTML súboru aby sa zobrazovala interaktívna mapa sveta vytvorená pomocou knižnice leaflet.

Zároveň ešte v tomto súbore modifikujeme title aby pri zobrazovaní vo webovom prehliadací písalo na karte text "World collection" a nastavíme aby ikonka vyzerala ako obrázok zemegule. Obrázok tejto ikonky nájdeme v tom isto podadresári ako dokument s názvom favicon.ico.

V tomto podadresári nájdeme aj adresár Images obsahujúci obrázky, ktoré sa budú vykreslovať v aplikácii.

Hlavný bod aplikácie

Hlavným bodom aplikácie je súbor index.tsx nachádzajúci sa v podadresári src. V tomto súbore je vytvorený koreň stromovej štruktúry elementu React.ReactNode. Metoda render() sa stará o renderovanie všetkých našich komponent. V nej sa zavolá iba komponenta App , ktorá sa už stará o renderovanie ostatných komponent na základe stavu aplikácie.

```
const root = ReactDOM.createRoot(  
    document.getElementById('root') as HTMLElement  
) ;
```

```
root.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>  
)
```

Všetky funkčné komponenty sa nachádzajú v podadresári src. Z toho dôvodu od teraz budeme dalsie adresáre uvádzať vzhľadom k adresáru src, ktorý bude braný ako koreňový.

App komponenta

Táto komponenta ma na starosť renderovať daný stav aplikácie, ktorý si používateľ zvolí v navigačnom menu.

Aplikácia má štyri stavy a pre každý z nich vytvoríme vnútornú funkciu vracajúcu JSX element, v ktorom sa volá komponenta reprezentujúca daný stav.

Ukážka jednej funkcie reprezentujúcej určitý stav aplikácie:

```
const renderCollectionsOverviewState = () => {  
  return (  
    <CollectionsOverview />  
  )  
}
```

Použijeme hooks funkciu useState na to aby sme si zapamätali momentálny stav aplikácie. Hodnota stavu je typu JSX.Element a priradzovať da nej budeme spomínané jednotlivé funkcie. Ako počiatočný stav nastavíme stav renderujúci komponentu CollectionsOverview.

```
const [currentState, setCurrentState] = useState<JSX.Element>(renderCollection
```

Táto komponenta ma taktiež za úlohu vykresliť navigačné menu aplikácie, po-zostávajúce zo štyroch tlačidiel reprezentujúcich jednotlivé stavy aplikácie. Toto menu sa renderuje na základe šírky obrazovky. Pre mobilné zariadenie ho zobrazíme inak ako pre zariadenia s obrovskou šírkou obrazu. Pomocou knižnice react-responsive dokážeme do premennej priradiť boolean hodnotu, ktorá bude reprezentovať či sa aplikácia zobrazuje pre mobilné zariadenie alebo nie. Nasledujúci riadok kódu priradí do premennej hodnotu true ak šírka obrazovky je aspoň minimálne 1024 pixelov a tým vieme, že použitá obrazovka ma dostatočný rozmer.

```
const isBigScreen = useMediaQuery({ query: '(min-width: 1024px)' })
```

Ako návratovú hodnotu komponenta App vracia JSX.Element, v ktorom sa zavolá funkcia na základe premennej isBigScreen, ktorá vykresli navigačné menu s tlačidlami podľa šírky obrazovky a momentálne zvolený stav reprezentujúci zavolaním funkcie vykreslujúcu komponentu stavu.

```

return (
  <>
    {isBigScreen && renderNavbarForDesktop()}
    {!isBigScreen && renderNavbarForMobile()}
    <div >
      {currentState}
    </div>
  </>
);

```

Pre jednotlivé tlačidla nastavíme aby po kliknutí na nich sa do premennej reprezentujúcej momentálny stav aplikácie uložila hodnota ako funkcia, ktorá bude reprezentovať iný stav a tým pádom sa vykresli zavolaním novej funkcie iná komponenta.

Dátové modely

Ked' získame dátá z backendu, či už sa jedná o dátá z Wikidata , alebo vytiahnuté z databázy našej aplikácie, je potrebné vhodne tieto dátá na frontende vhodne reprezentovať. Nejde len o dátá získane z backendu. Existujú dátá, s ktorými sa pracuje iba na užívateľskej strane. Z toho dôvodu vytvoríme jednotlivé dátové modely. Každý dátový model reprezentuje nejaké dátá. Všetky modely nájdeme v adresári Data. Modely sú implementované ako trieda, kde jej atribúty reprezentujú dátá. Každá trieda ma konštruktor. Ten buď berie argumenty, ktoré sa jednotlivo v konštruktore dosadia do atribútov, alebo niektoré berú jeden argument "initializer" datového typu any. To znamená, že môžeme dosadiť úplne hoci aké dátá. V konštruktore sú definované if podmienky, ktoré skúsia z initializer dostat tie správne dátá a priradiť ich do atributov. Tento prístup využívajú hlavne dátové modely, ktoré vzniknú zakonštruovaním z dát získaných z backendu.

Nebudeme si prechádzať jednotlivé dátové modely, ukážeme ako príklad iba jeden z nich.

```

export class CollectibleBasicInfo {
  imageURL: string | null = null
  description: string | null = null

  constructor(initializer?: any) {
    if (!initializer) return;
    if (initializer.image) this.imageURL = initializer.image;
    if (initializer.desc) this.description = initializer.desc;
  }
}

```

Trieda CollectibleBasicInfo reprezentuje dátá, ktoré sú cez WikidataAPI získané z Wikidata. Jedná sa o informáciu obrázku mapového objektu a popisu popisujúceho tento objekt. WikidataAPI vracia všetky dátá vo formate Json. Z toho dôvodu konštruktor berie iba argument initializer a z neho sa do dátového modelu priradia dátá spôsobom aký je ukázaný v kóde.

API Proxy

Frontend musí nejako komunikovať s backendom. Na stane backendu nájdeme REST API v podobe dvoch časti. WikidataAPI a WorldCollectionAPI. API ponúkajú na jednotlivých koncových uzlov danú funkciaľitu. Našou úlohou na strane užívateľa je vytvoriť proxy smerujúce na toto API.

V adresári world-collection-app nájdeme dokument package.json. Ten obsahuje konfiguráciu projektu. Do dokumente pridáme novú dvojicu, kde kľúč je "proxy" a ako hodnotu mu dáme URL kde sídli backend. Kedže server pri spustení pobeží lokálne na našom zariadení, z ktorého je spustení aj frontend, tak vieme že nám vdaka knižnici Flask pobeží na IP adresu zariadenia na porte 5000.

```
"proxy": "http://127.0.0.1:5000/"
```

Teraz nám stačí uviesť iba zvyšnú časť URL na konkrétny koncový uzol.

V adresári src nájdeme podadresár API. V ňom nájdeme dokumenty LocalAPIProxy a WikidataAPIProxy. Obe obsahujú triedu s metódami, kde každá z nich posiela dátu na určitý koncový uzol a následne získava dátu z backendu. V týchto dokumentoch úplne hore nájdeme aj výpis všetkých koncových uzlov, ktoré REST API backendu ponuka. Úlohou každej metódy je z argumentov zstrojiť javascriptovy objekt, ktorý sa zakóduje do Json formátu a odošle sa na konkrétny koncový uzol. Všetky metódy sú asynchronné a teda vracajú Promise. Po tom ako backend vráti nejaké dátu, sú tieto dátu pomocou privátnych metód konvertované do určitého dátového modelu. Inak povedane tieto metódy nevracajú dátu ako Json, ale ako triedy dátového modelu reprezentujúce konkrétné dátu.

Ukážka metódy, ktorá zstrojí objekt s argumentu reprezentujúceho zadanú klúčovú hodnotu. Následne sa objekt pomocou Fetch triedy odošle na koncový uzol. Na backende sa zstrojí SPARQL dotaz, ktorý sa odošle na Wikidata nato aby vyhľadal všetky mapové objekty zhodujúce sa v nazve s uvedenou hodnotou v argumente. Potom sa dátu vrátia na frontend, kde sa pomocou privátnej metódy skonvertujú do daného dátového modelu:

```
private static convertToListOfSearchDataModel(data: any[]): SearchData[] {
    let results: SearchData[] = data.map((d: any) => new SearchData(d));
    return results;
}

static async searchForCollectible(searchWord: string) {
    let params = {
        search_word: searchWord
    }
    const data = await Fetch.Get(searchPlacesOrCollectibles, params);
    return this.convertToListOfSearchDataModel(data);
}
```

Obe APIProxy využívajú triedu Fetch, ktorej implementácia sa nachládza v tom istom adresári. Úlohou Fetch je odosielanie a získavanie dát z koncových uzlov. Fetch obsahuje verejné metódy reprezentujúce konkrétnu HTTP metódu, ktorou sa požiadavka na backend pošle a samotný mechanizmus na odosielanie dát.

Interaktívna mapa

Ako už bolo ukázane v kapitole návrhu je potrebne vytvoriť komponentu vytvárajúcu a spravujúcu interaktívnu mapu sveta. Na to využijeme spomínanú knižnicu leaflet a jej pomocnú knižnicu pre react react-leaflet. Mapu sa bude zobrazovať pre rôzne účely. Aby sme predišli používaniu toho istého kódu niekoľko krát, tak vytvoríme komponentu MapWrapper. Tuto komponentu a všetky ďalšie komponenty spojene s mapou nájdeme v adresári Map. MapWrapper obsahuje mechanizmus vykreslenia interaktívnej mapy a jej konfigurácie. Mapu vykreslime pomocou MapContainer z knižnice react-leaflet. Do vnútra tohto kontajnera je potrebné vložiť komponentu TileLayer, ktorou úlohou je získať mapové dátu. Jej argumenty nastavíme podľa leaflet dokumentácie. Aby sa mapa vykreslila je potrebne nastaviť tomuto kontajneru šírku a výšku. Tu nastavíme v dokumente Map.css.

Ukažká CSS dokumentu:

```
.leaflet-container {  
    width: inherit;  
    height: calc(100vh - 70px);  
}
```

Do MapContainer teraz vieme pridať ďalšie komponenty ponúknuté z knižnice react-leaflet ako komponentu Marker vykreslujúci bod na mape. Tieto komponenty, ale už závisia od konkrétneho použitia interaktívnej mapy. MapWrapper preto bude brat iba jeden argument. To bude funkcia vracajúca JSX.Element. Do tejto funkcie vložíme implementáciu konkrétnych vecí, čo sa majú do mapy vykresliť.

Naša aplikácia do interaktívnej mapy bude hlavne vykreslovať mapové objekty. Bud ako mapové objekty, ktoré používateľ vyhľadáva a ešte nie sú uložené v databáze, alebo už uložené mapové objekty. Aby sme odlišili tieto rôzne dátu od seba tak mapové objekty už uložené v databáze označíme ako Collectible a tie neuložené ako RawCollectible. Hlavným rozdielom je, že Collectible obsahujú navyše dátá reprezentujúce návštěvnosť, poznámky a obrázok ikonky. Preto potrebujeme dva rozdielne interaktívne mapy. Jedna nám zobrazí RawCollectible a druhá Collectible. Tieto mapy nájdeme v komponentoch MapShowingCollectibles a MapShowingRawCollectibles. Tieto komponenty v sebe zavolajú komponentu MapWrapper a predajú jej ako argument funkciu, kde prejdú všetky mapové objekty, a pre každý z nich zavolajú komponentu na vykreslenie vhodného bodu, Markeru.

V našej aplikácii budeme používať tri rôzne Markery. Jeden pre Collectible, druhý pre RawCollectible a tretí bude nadobúdať schopnosť byť posúvaným na mape. Možnosť hýbať s bodom na mape je potrebne v prípade, keď používateľ vyberá na mape stred kruhu v ktorom sa majú mapové objekty vyhľadávať. Všetky tri spomínane komponenty nájdeme v podadresári Markers.

Ukážka implementácie posuvného markeru:

```
function DraggableMarker({ position, handleChangeOfPosition }: DraggableMarkerProps)  
const markerRef = useRef<any>(null)  
const eventHandlers = useMemo(  
    () => ({ dragend()
```

```

    {
        const marker = markerRef.current
        if (marker != null) {
            handleChangeOfPosition(marker.getLatLng())
        }
    },
),
[handleChangeOfPosition])

return (
    <Marker
        draggable={true}
        eventHandlers={eventHandlers}
        position={position}
        ref={markerRef}>
    </Marker>
)
}

```

Každý z markerov ako návratovú hodnotu vracia element <Marker> <Marker/>. Tento element poskytuje knižnica react-leaflet. Do kontajneru mapy môžeme vkladať iba tieto <Marker> elementy. Naše jednotlivé komponenty markerov majú za úlohu vložiť do elementu marker element Popup, ktorý správne naplnia obsahom. Element Popup je taktiež element z knižnice react-leaflet a reprezentuje okienko, ktoré sa zobrazí po tom ako sa na mape klikne na daný element Markeru.

RawCollectibleMarker ma za úlohu vložiť do elementu Popup obsah, ktorý do okienka pridá mechanizmus na editovanie mapového objektu pred uložením do aplikácie. Na druhu stranu CollectibleMarker do Popup pridá mechanizmus na vykreslenie infomácií o mapovom objekte. Informácie ako obrázok a popis sú získeane z WikidataAPI pomocou triedy WikidataAPIProxy. V tomto "okienku"ktorý element Popup zobrazí musíme používateľovi umožniť aj možnosť modifikovať návštevnosť, ikonku, poznámky a zobraziť detaily mapového objektu. Na všetko toto sa do okienka zavolá iná komponenta, ktorá pridá do okienka možnosť modifikácie alebo zobrazovania. Komponenty na modifikáciu návštevnosti, tvorenia poznámok a nastavenia ikonky nájdeme v adresári src/CollectibleEditableComponents. Komponentu na získanie a vykreslenie detailov mapového objektu nájdeme v adresári src/CollectibleDetails.

Ikonka markeru

Element Marker berie viacej argumentov. Jeden z nich je argument "icon". Do neho je potrebné uviesť ikonku, ktorá sa ma zobraziť ako bod na mape. Na získavanie ikoniek si vytvoríme pomocnú funkciu nachádzajúcu sa v dokumente GetIcon.ts v adresári src/Map/MapFunc.

Ukážka funkcie:

```

export function getIcon(nameOfIcon: string, grayFlag: boolean) {
    return L.icon({
        iconUrl: require(' ../../static/Icons/' + nameOfIcon + '.png'),
        iconSize: [64, 64],

```

```

        className: (grayFlag) ? "" : "gray"
    })
}

```

Tato funkcia berie dva argumenty. Jeden je názov ikonky, ktorý je uložený v databáze. Druhy je boolean, ktorý nastaví, či sa ma obrázok ikonky zobraziť čisto sivo alebo plne farebná. Obrázky všetkých ikoniek máme uložené v adresári src/static/Icons. Druhy argument predáme informáciu, či bol daný mapový objekt navštívený. Ak ešte neboli tak sa do triedy ikonky na základe argumentu pridá trieda "gray". V Marker.css nachádzajúci sa v adresári src/Map/Marker nájdeme definíciu nasledujúceho štýlu:

```

gray {
    filter: grayscale(1);
}

```

Teda na elementy, ktoré sú triedou "grey" sa aplikuje filter, ktorý sa postará aby boli zobrazene bez farby iba čisto sivé. Tento filter sa dá aplikovať na obrázky.

Mapové možnosti

Po tom ako sa na interaktívnej mape vykreslia všetky body pomocou elementu Marker je užitočné aby sa viditeľná oblasť mapy zmenila. My sme tým, že sa pohľad na mapu zmení tak aby bolo na nej možné vidieť všetky práve vykreslene body. To docielime vytvorením komponenty MapBoundsOption. Tá ako parameter zoberie pole súradníc všetkých bodov, ktoré sa majú zobraziť na mape. A na základe súradníc pomocou leaflet funkcie fitBounds() sa nastaví pohľad mapy. Tuto komponentu je možné pridať do kontaineru MapContainer.

Ďalšou vhodnou možnosť je to, aby pohľad mapy bol prenesený animáciou na konkrétny bod. To docielime pomocou leaflet funkcie flyTo(), ktorá ako argument berie dvojicu reprezentujúcu súradnice bodu.

```

export interface MapFlyToOptionProps {
    pointOfTheEarth: { lat: number, lng: number }
}

function MapFlyToOption({ pointOfTheEarth: point }: MapFlyToOptionProps) {
    const map = useMap();

    useEffect(() => {
        map.flyTo(point);
    }, [map, point])

    return (
        <></>
    );
}

```

Implementáciu mapových možností nájdeme v dokumentoch nachádzajúcich sa v adresári src/Map/MapOptions.

Vyhľadávacie pole

V aplikácii na viacerých miestach je potrebne umožniť používateľovi vyhľadať objekty na Wikidatách podľa zadania názvu. Na to vytvoríme komponentu SearchBar nachádzajúcu sa v adresári src/SearchBar. Úlohou tejto komponenty je vykresliť input element, do ktorého môže používateľ zadat názov hľadaného objektu. Zároveň je potrebne spracovať zadaný textový reťazec, poslat na správny koncový uzol uvedenú hodnotu a vrátene výsledky vhodne vykresliť. Výsledky vykreslime ako zoznam, ktorý sa zobrazí pod input elementom. Keďže existuje viacero vyhľadávaní tak implementuje komponentu tak aby bola použitá na všetky účely. Z toho dôvodu potrebuje tato komponenta brať nasledujúce tri argumenty:

- placeHolderText – text, ktorý sa zobrazí v input elemente.
- handleClickedResult – funkcia, ktorá sa zavolá po tom ako používateľ klikne na nejakú vyhľadanú možnosť. Aby sme mohli v danom stave aplikácie vhodne spracovať vybranie hľadanej možnosti.
- dataGetter - funkcia, ktorá ako argument berie text, ktorý je zadaný v input elemente a vracia Promise, obsahujúci vyhľadane dátá. Do tejto funkcie vložíme korektné volanie metódy z WikidataAPIProxy podľa jednotlivých účelov.

Stavy aplikácie

Funkčné komponenty reprezentujúce všetky štyri stavy nájdeme v podadresári APPStates. Stavy odpovedajú komponentom z návrhu architektúry aplikácie. Pre pripomienutie sú to:

- Prehľad kolekcii
- Vyhľadanie mapových objektov
- Pridanie mapového objektu
- Editácia

Každá z týchto komponent volá v sebe iné pod-komponenty, ktoré spolu tvoria a vykresľujú daný stav aplikácie. Podme si prejst tieto stavy a ukážeme aké pod-komponenty tieto komponenty volajú.

Prehľad kolekcii

V tomto stave má aplikácia poskytnúť užívateľské rozhranie, ktoré umožní používateľovi si prezerať mapové objekty patriace do danej kolekcie pomocou interaktívnej mapy sveta. Tato komponenta, keď sa prvý krát vykreslí, tak ma za úlohu získať z backendu dátá všetkých kolekcii aby ich mohla následne zobraziť. Toto chovanie implementujeme pomocou hooks metódy useEffect. Ta zavolá metódu z LocalAPIProxy, ktorá získa z backendu tieto dátá. Po tom ako ich úspešne získame ich uložíme do vhodných premenných aby sme dátá mohli ďalej používať v kóde. Tieto premenne sú definované cez hooks funkcie useState a teda reprezentujú stav aplikácie.

```

useEffect(() => {
    setCollectionLoading(true);
    LocalAPIProxy.getAllCollections().then((collections) => {
        setCollectionLoading(false);
        setCollections(collections);
        setCollectionsToShow(collections)
    })
);
}, [])

```

Poznámka: useEffect() berie dva argumenty, prvý je funkcia, ktorá sa zavolá ak sa zmení hodnota uložená v premennej, ktorú uvedieme do poľa ako druhý argument. Ak pole je prázdne ako v ukážke vyššie, tak to znamená, že sa funkcia zavolá iba raz a to keď sa komponenta prvý krát vykreslí.

Komponenta ma za úlohu vykresliť bočný panel, v ktorom je zoznam kolekcii s informáciami. Tento bočný panel je najprv zobrazení ako úzky panel s jedným tlačidlom. Po tom ako je tlačidlo stlačené sa zobrazí širší panel zobrazujúci zoznam kolekcii. Širší panel sa vykreslí zavolaním funkcie renderCollectionsMenu() a užší panel je implementovaný samostatne. Podľa premennej showingCollectionsMenu rozhodneme aký panel vykreslime. Hodnotu tejto premennej nastavujeme na základe stlačených tlačidiel. Jeden na otvorenie panelu a druhý na zavretie.

```

{showingCollectionsMenu ? (
    <>
    {renderCollectionsMenu()}
    </>
) : (
    <>
        <div className='side-menu'>
            <button data-testid="buttonToOpenCollectionList" type="button" class="button-align-left">
                <img className="align" src={require('../static/Icons/menu.svg')} alt="menu icon" />
            </button>
        </div>
    </>
)}

```

V paneli sa následne zobrazí zoznam kolekcii. Každý prvok v zozname reprezentuje jednu kolekciu, vykresluje informácie o kolekcii. Tento zoznam vykreslime Tým, že pre-iterujeme premennú collectionsToShow, v ktorej sú uložené kolekcie. Pre každý prvok nastavíme, aby v prípade kliknutia sa zavolala funkcia handleClickOnCollection(). Tá ako parameter berie dátový model reprezentujúci kolekciu. Tu získame počas iterácie všetkých kolekcii. Tato funkcia ma za úlohu spracovať kliknutie na kolekciu. Počas tohto procesu sa zavolá aj funkcia handleCollectionSelect(), ktorá získa z backendu zoznam všetkých mapových objektov v danej kliknutej kolekcií. Získane dátu sa priradia do premennej CollectiblesToShow definovej pomocou useState.

Komponenta zároveň volá pod-komponentu MapShowingCollectibles, ktorá ma za úlohu vykresliť interaktívnu mapu s bodmi reprezentujúcimi mapové objekty a ako props berie mapové objekty. Ako props predáme premennú collecti-

blesToShow, v ktorej sú uložene získane mapové objekty z backendu. Tie sú následne pre-iterované a pre každý z nich je zavolaná komponenta CollectibleMarker, ktorej predáme dátový model reprezentujúci daný mapový objekt.

Pridanie mapového objektu

V tomto stave má aplikácia poskytnúť užívateľské rozhranie, ktoré umožní používateľovi vyhľadať mapové objekty podľa názvu. Následne zvolené mapové objekty zobrazí ako body na interaktívnej mape. Po tom ako používateľ vyhľadá a zvolí všetky mapové objekty, aplikácia mu umožní pridať zvolené mapové objekty do niektoréj z kolekcii. Inak povedané uložiť mapové objekty do databázy. Aj v tomto stave použijeme mechanizmus bočného úzkeho a širšieho panelu, ktorý implementuje rovnakým princípom ako v predošлом stave.

Pre vykreslenie interaktívnej mapy použijeme komponentu MapShowingRawCollectibles, pretože budeme do nej vykreslovať mapové objekty, ktoré nie sú uložene v databáze. Mapové objekty bude možné vyhľadať pomocou komponenty SearchBar, ktorej predáme argument na získavanie dát ako nasledujúcu funkciu:

```
const dataGetter = (searchWord: string) => {
    return WikiDataAPIProxy.searchForCollectible(searchWord);
}
```

Ta podľa zadaného názvu nájde mapové objekty zhodujúce sa v názve so zadanou hodnotou.

Po tom ako používateľ zvolí nájdený mapový objekt v komponente SearchBar sa na interaktívnej mape zobrazí daný mapový objekt. V tomto konkrétnom prípade Marker nevyužíva všetky štyri komponenty iba "Detaily mapového objektu". To je z dôvodu, že tento mapový objekt ešte nie je uložený v databáze a preto aplikácia smie poskytnúť iba detaily o mapovom objekte. Vyhľadané a vybraté mapové objekty aplikácia umožní uložiť do kolekcie. Na to existuje komponenta pre ukladanie mapových objektov a komponenta pre tvorbu nových kolekcii, ktorá tento mechanizmus zapuzdruje. Implementáciu týchto komponent nájdeme v adresári src/DataSaving.

Editácia

V tomto stave aplikácia zobrazí tabuľku s kolekciami a možnosťami dané kolekcie editovať. Každú kolekciu používateľ je schopný otvoriť a následne sa mu zobrazí editačná tabuľka v ktorej vie editovať mapové objekty v danej kolekcií.

Pomocou useEffect() sa z backendu získajú dátá o kolekciách, potom ako je komponentná prvý krát vykreslená.

Tabuľka ponúkajúca možnosť editovať kolekcie sa vykresli pomocou komponenty CollectionsEditingTable, ktorú nájdeme v adresári src/Tables/EditationTable. Tato komponenty berie ako argumenty kolekcie, z ktorých zostrojí tabuľku, a ďalšie funkcie, ktoré sú zavolane vo volanej komponente, potom ako prebehne nejaká editácia kolekcií. Pomocou týchto funkcií dôkaze podkomponenta upozorniť jej rodičovskú komponentu a zavolať jej funkcie. Mechanizmus na ukladanie dát, editáciu a podobne je implementovaný vo funkciách v komponente Editation a nie CollectionsEditingTable.

Komponenta CollectionsEditingTable v sebe volá komponentu Table. Keďže v aplikácii vykresľujeme tabuľku pre tri rôzne udalosti, tak sme vytvorili komponentu Table. Ta v sebe ako návratovú hodnotu obsahuje elementy na vykreslenie tabuľky. Ako argumenty berie funkcie, ktoré v sebe obsahujú konkrétnie implementácie hlavičky a tela tabuľky. Tieto funkcie sú zavolane vo vnútri elementu table tak aby reprezentovali hlavičku a telo tabuľky.

Komponentu Table nájdeme v adresári src/Tables/Table. Okrem toho tam nájdeme aj komponentu TableFooter. Jedná sa o pomocnú komponentu, ktorá vykresluje a spracováva pätičku tabuľky. Pomocou pätičky používateľ nastavuje počet záznamov na stranu v tabuľke, prepína si medzi stranami a podobne.

CollectionsEditingTable predá komponente Table funkcie, v ktorých je konkrétna implementácia hlavičky a tela tabuľky. V tabuľke chceme vykresliť informácie o kolekcii a tlačidla, ktoré umožnia editáciu, mazanie kolekcie a podobne. Taktiež chceme umožniť vybrať kolekciu pomocou tlačidla, ktoré vykresli novu editačnú tabuľku. Tuto novu tabuľku vykreslime pomocou komponenty CollectiblesEditingTable. Tato komponenta funguje na rovnakom princípe ako CollectionsEditingTable, ale predáva komponente Table funkcie s rozdielnou implementáciou. Tabuľka obsahuje informácie o mapových objektoch vo vybranej kolekcii a možnosť ich editovať a mazat.

Pre všetky zmeny v tabuľkách sa zavolajú funkcie predané ako argumenty do týchto komponent. Funkcionalitu týchto zmien ma na starosti spracovať komponenta Editation. Zároveň tato komponenta obsahuje API volanie na server aby boli uvedene zmeny uložené v databáze.

Vyhľadanie mapových objektov

V tomto stave aplikácia zobrazí postupne možnosť zadať parametre vyhľadávania. Najprv výber kategórie, následne vyber spôsobu definovania oblasti vyhľadávania, definovanie konkrétnej oblasti a nakoniec vyber filtrov a zadanie hodnôt daných filtrov. Potom sa uskutoční volanie na WikidataAPI , kde sa tieto parametre zabalia do Json formátu a odošlú na server ako náklad v tele požiadavky. Server z nich postaví konkrétny dotaz a pošle ho na Wikidata. Vrátené výsledky sa zobrazia buď ako tabuľka alebo ako body na interaktívnej mape.

Tento stav je tvorení z pomerne dost komponent. Všetky komponenty nájdeme v adresári src/AppStates/CollectiblesSearchingStates. Hlavnou komponentnou je CollectiblesSearching. Tá manažuje všetky stavy, v ktorých používateľ zadáva parametre vyhľadávania. Každý z týchto stavov je reprezentovaný inou komponentov. Tieto komponenty nájdeme v dokumentoch v podadresári Collectibles-SearchingStates. CollectiblesSearching komponenta si za pomoci useState hooks pamäta momentálny stav, v ktorom sa používateľ vo zadávaní parametrov nachádza a objekt CollectiblesSearchQueryData. Tento objekt slúži na zapamätyvanie si parametrov, ktoré používateľ zadal. Na konci je tento objekt konvertovaný do Json formátu, ktorý je odoslaný na backend pomocou vhodnej metódy z triedy WikidataAPIProxy.

Podľa momentálneho nastavaného stavu, komponenta vo svojej návratovej hodnote vracia volanie funkcie, ktorá v sebe vola komponentu renderujúcu daný stav. Komponenta každého stavu ako jeden z argumentov berie funkciu, ktorá je zavolaná potom ako sú parametre zadane a používateľ zvolí, že chce pokračovať

na ďalší stav. Podkomponenta v takom príde zavolá tuto funkciu, ktorá je implementovaná v hlavnej komponente CollectiblesSearching. Každá z týchto funkcií berie iné argumenty na základe, ktorých vo funkcii do objektu CollectiblesSearchQueryData zapíše zvolene argumenty a zvolí nový stav.

Jednotlivé stavy si nebudeme podrobnejšie rozoberať. Spomenieme, ale že využívajú komponenty ako SearchBar, alebo MapWrapper pre vyhľadávanie jednotlivých hodnôt parametrov a vykreslenie interaktívnej mapy sveta. v tomto adresári nájdeme aj podadresár Filters. V ňom sú komponenty, ktoré vykresľujú užívateľské rozhranie na vyber hodnôt jednotlivých filtrov.

Po tom ako používateľ zvolí všetky parametre a pokračuje na posledný stav, tak sa nastaví stav, ktorý zavolá komponentu CollectiblesPresenter. Tato komponenta ako props berie objekt CollectiblesSearchQueryData. Potom ako sa zmení tento objekt a zobrazí sa komponenta, tak sa zavolá nasledujúci useEffect hooks:

```
useEffect(() => {
    setLoading(true);
    WikiDataAPIProxy.searchForCollectibles(dataForWikibaseAPI).then((data) => {
        setLoading(false)
        setCollectibles(data);
    }).catch(() =>
        setError(true)
    )
}, [dataForWikibaseAPI])
```

Poznámka : v premennej dataForWikibaseAPI je uložený objekt typu CollectiblesSearchQueryData obsahujúci všetky zvolene parametre vyhľadávania.

Potom ako dobehne SPARQL dotaz a backend vráti výsledky, tak tato komponenta zavolá podkomponentu FoundResultsHandler. Jej predá získane dátu reprezentujúce vyhľadané mapové objekty. FoundResultsHandler ma za úlohu zobraziť a vhodne prezentovať nájdene dátu pomocou interaktívnej mapy, inak povedane komponentnou MapShowingRawCollectible, alebo zobrazí dátu v tabuľke pomocou komponenty RawCollectiblesTable. Tuto komponentu nájdeme v adresári src/Tables/RawCollectiblesTable.

Pre uloženie nájdených mapových objektov sa zavolajú pomocné komponenty implementované v adresári src/DataSaving.

No ešte pred uložením aplikácia umožní používateľovi mapové objekty modifikovať. Tým myslíme meniť názov objektu, alebo ho zmazať z množiny mapových objektov, ktorú následne si chce používateľ uložiť. Z dôvodu umožnenia používateľovi vrátiť zmenu implementujeme mechanizmus "undo". Inak povedane budeme si pamätať par predošlých stavov množiny nájdených mapových objektov. V prípade, že používateľ chce vrátiť úpravu naspäť, tak zo zapamätaných stavov získame stav aký bol pred úpravou. Tento mechanizmus implementuje pomocou návrhového vzoru "State". Implementácia nájdeme v adresári src/DataSearcing/Undo.

6. Užívateľská dokumentácia

Webová aplikácia umožňuje používateľovi vyhľadávať a spravovať mapové objekty. Vyhľadávať konkrétny mapový objekt je možné na základe zadania názvu. Aplikácia zároveň poskytuje možnosť vyhľadávať množinu mapových objektov, ktorá je obmedzená na základe zadaných parametrov. Používateľ je schopný si mapové objekty uložiť do nejakej kolekcie, poprípade vytvoriť novu, a následne svoje kolekcie s mapovými objektami spravovať. Spravovaním je myšlená možnosť si ku každému mapovému objektu urobiť textové poznámky, alebo označiť, že daný mapový objekt používateľ navštívil. V aplikácii používateľ vždy úplne hore nájde navigačné menu, kde pomocou tlačidiel idúcich za sebou v jednom rade, môže meniť stav aplikácie.

Aplikácia je responzívna, z toho dôvodu sa na mobilných zariadeniach navigačné menu zobrazuje inak. Na malých obrazovkách v menu používateľ nájde iba jedno tlačidlo, ktoré po stlačení zobrazí panel s tlačidlami na vyber stavu aplikácie.

Aplikácia má štyri stavy. V nasledujúcich sekciách sú popísané funkcionality všetkých stavov.

6.1 Prehľad kolekcii

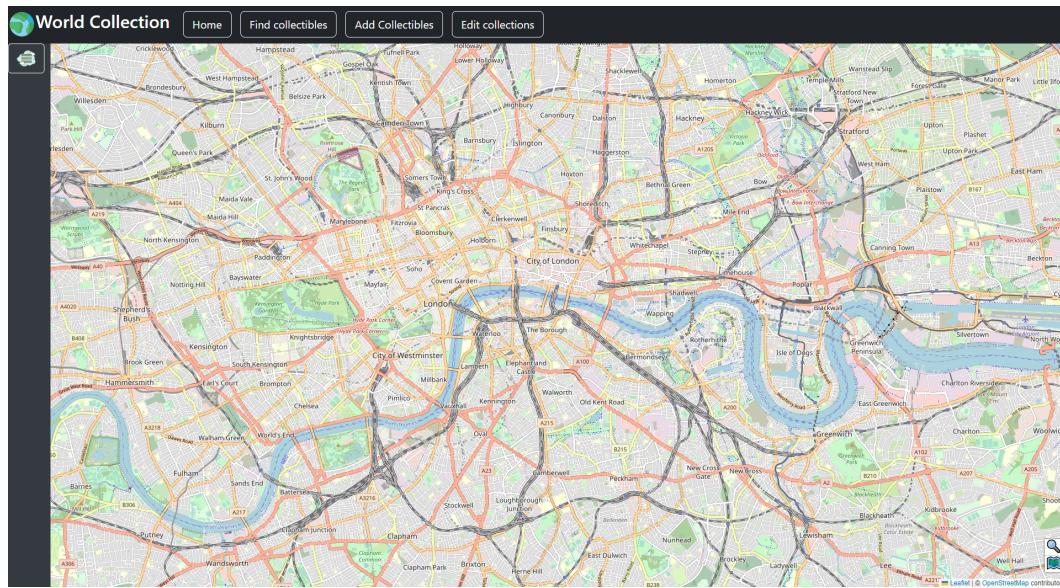
Tento stav je nastavený ako default. Používateľ po navštívení webovej stránky uvidí aplikáciu v zobrazujúcu tento stav. Po prejdený do iného stavu sa používateľ môže vrátiť do tohto stavu pomocou tlačidla "Home" v hornom navigačnom menu.

V tomto stave aplikácia zobrazí interaktívnu mapu sveta. Používateľ je schopný si mapu prezerat, viditeľnú oblasť mapy priblížovať alebo oddiaľovať pomocou kolieska na myši alebo tlačidlami nachládajúcimi sa v pravom dolnom rohu mapy. Tlačidlo s "lupou" priblížuje a tlačidlo s "mapou" na druhú stranu oddiaľuje. Takéto správanie majú všetky mapy v tejto aplikácii.

Dokým používateľ nevyberie nejakú kolekciu tak mapa je čistá. To znamená, že nezobrazuje žiadnen bod na mape, ktorý by reprezentoval uložený mapový objekt.

Na ľavej strane od mapy nájdeme úzky panel s jedným tlačidlom. Po stlačení tlačidla sa zobrazí namiesto úzkeho panelu širší panel, v ktorom používateľ nájde svoje vytvorené kolekcie v podobe zoznamu, a šírka mapy sa zmenší. Na zariadeniach s malou obrazovkou je tento panel zobrazený na celu šírku obrazovky a interaktívna mapa nie je zobrazená. Používateľ je schopný zatvoriť tento panel stlačením červenejho tlačidla "X" nachládajúcim sa v pravom hornom rohu panelu. V panely hore nájdeme textové pole, kde môže používateľ zadať text na základe ktorého sa zobrazujú príslušné kolekcie. Inými slovami používateľ dokáže pomocou tohto vyhľadávať svoje kolekcie.

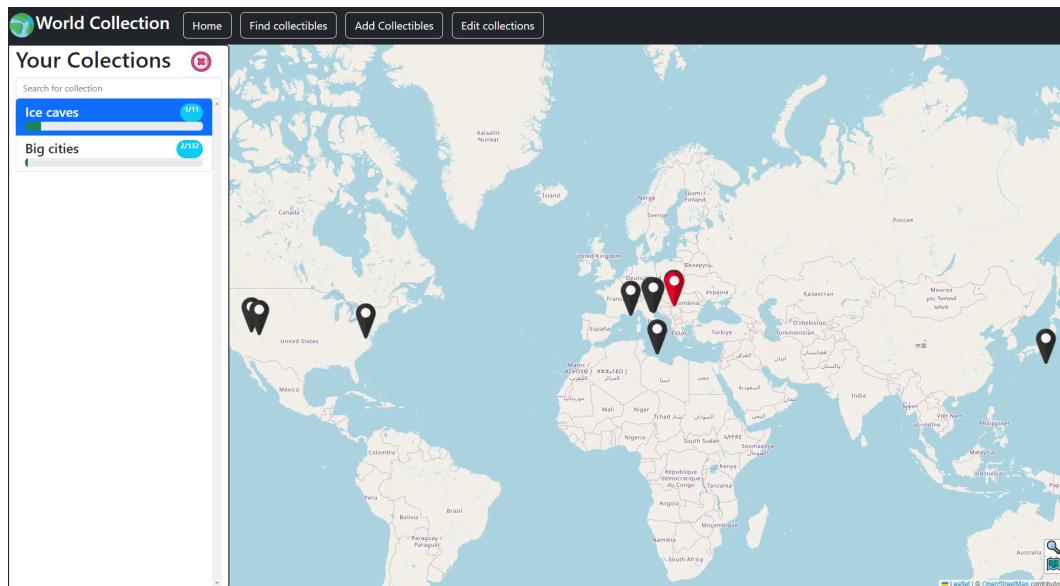
Každá kolekcia v zozname je reprezentovaná ako informatívny obdĺžnik, kde hore v ľavo je názov kolekcie, a na pravo sa nachádza informácia vyjadrená ako počet navštívených mapových objektov v danej kolekcie ku počtu všetkých mapových objektov v kolekcií. V dolnej časti je "bar", ktorý sa vyfarbuje do zelena na základe percenta navštívených objektov. Teda ak používateľ v danej kolekcií nemá navštívený žiadnen mapový objekt tak celý bar je sivý. Ak sú všetky mapové



Obrázek 6.1: Vzhľad aplikácia po navštívený stránky

objekty v kolekcii navštívené tak bar je celý naplnený zelenou farbou.

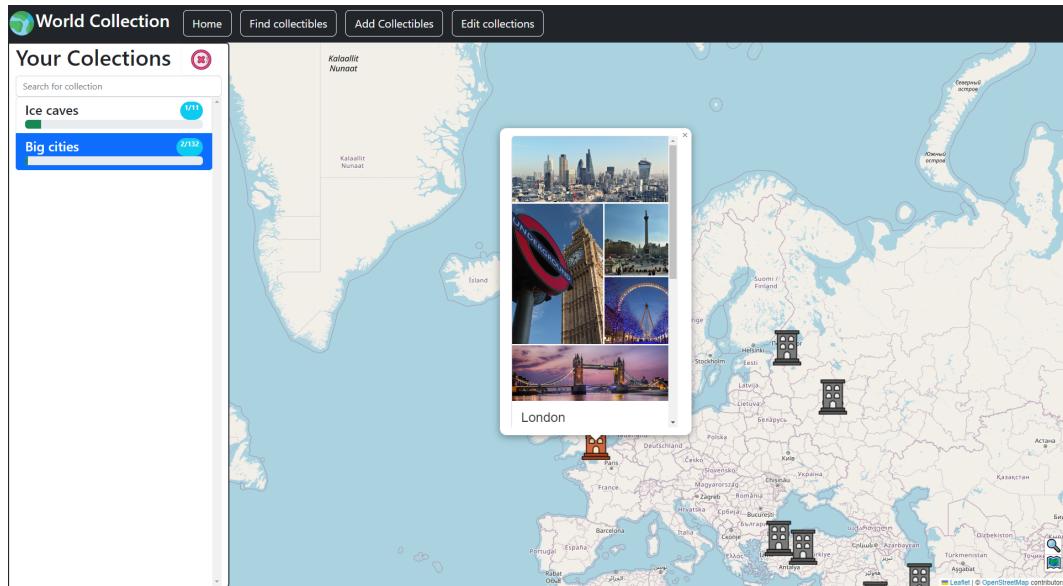
Používateľ môže na obdĺžnik kolekcie kliknúť. Po kliknutí sa na interaktívnej mape zobrazia body reprezentujúce všetky mapové objekty v danej kolekcii. Pohľad mapu sa automaticky nastaví tak aby boli viditeľné všetky body naraz. Po kliknutí na inú kolekciu sa na mape zobrazia body novo kliknutej kolekcií. Body sú vyfarbené na základe navštívenia daného mapového objektu. Sivé, nevyfarbené body znamenajú, že používateľ ešte nenavštívil daný objekt, kým plne vyfarbené body reprezentuje, že objekt už bol navštívený.



Obrázek 6.2: Prezentovanie mapových objektov danej kolekcie

Používateľ môže kliknúť na jednotlivý bod na mape. Po tom ako na neho klikne sa na mape tesne nad bodom zobrazí obdĺžnik, v ktorom sú informácie o danom mapovom objekte. Tento obdĺžnik ma danú veľkosť, z toho dôvodu sa

obsah nezmestí cely do obdĺžniku. Používateľ pre prezretie celého obsahu musí v obdĺžniku skrolovať.



Obrázek 6.3: Zobrazenie informácií pre daný bod

V obdĺžniku na úplnom začiatku je zobrazený obrázok mapového objektu. Môže nastať situácia, že obrázok neboli najdený a v takom prípade ho aplikácia nezobrazí. Po obrázku nasledujú informácie ako názov objektu, popis objektu a zoznam jednotlivých "tagov" reprezentujúcich kategórie, pod ktoré mapový objekt patrí. Napríklad pre mesto "Instanbul" nájdeme tag s informáciou "million city", čo znamená, že objekt je mesto s viac než miliónom obyvateľov. Ďalej nasleduje status o návštive objektu. V prípade ak objekt nie je navštívený tak v statuse nájdeme červený obdĺžnik s textom "Not visited yet", v opačnom prípade je to zelený obdĺžnik s textom "Visited". Pod statusom návštevnosti nájdeme priestor pre poznámky. Ak si používateľ zapíše k objektu poznámky, tak pravé na tomto mieste ich aplikácia zobrazí používateľovi.

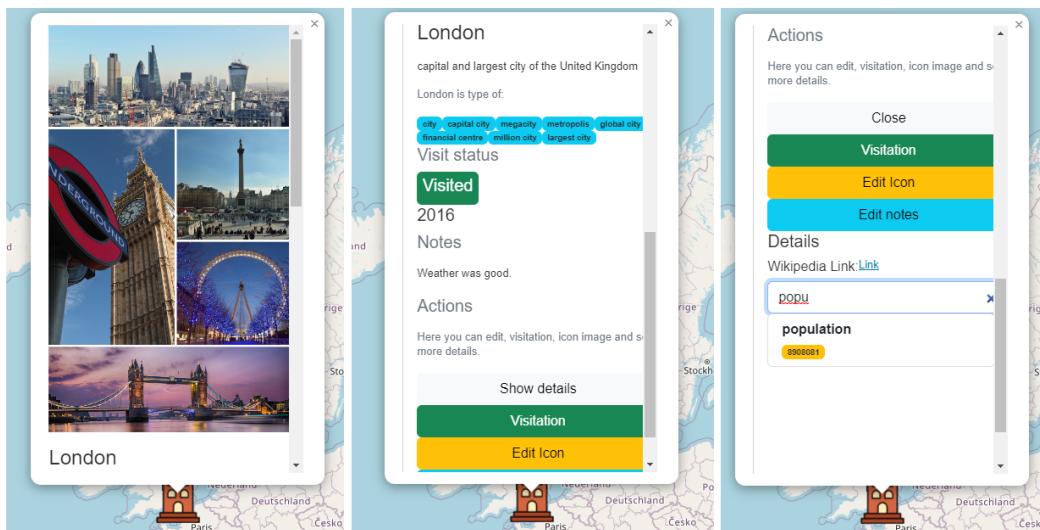
Na konci nájdeme akcie mapového objektu. V nich nájdeme štyri pod sebou idúce tlačidla, kde každé z nich po stlačení do obdĺžnika pridá nové časti. Po stlačení daného akčného tlačidla sa text tlačidla zmení na "Close" a po znova kliknutí na dane tlačidlo, pridaná časť zmizne.

Nasleduje výpis týchto akcii:

- "Show details"- zobrazí zoznam s detailmi o mapovom objekte a zároveň odkaz na článok o danom mapovom objekte vedúci na anglickú Wikipédiu, ak článok existuje. Detaily sú zobrazené ako zoznam s obdĺžnikmi, kde každý jeden z nich reprezentuje jeden detail. V obdĺžniku je v hornej časti názov detailu a v dolnej časti sú hodnoty daného detailu zobrazené ako "tagy". Tagy majú rôznu farbu na základe dátového typu detailu. Zelená je pre hodnoty reprezentujúce čas, oranžova pre kvantitatívne hodnoty a modra pre hodnoty reprezentujúce nejakú existujúcu entitu, objekt.
- "Visitation"- zobrazí "check box", ktorý používateľ zaškrtnie ak daný objekt navštívil. Po zaškrtnutí je zobrazená možnosť zadať dátum návštevy. Ten

sa da zadať ako dátum, mesiac s rokom alebo iba ako rok. Aplikácia zároveň umožňuje zadať dátum návštevy aj ako časové obdobie definované dvoma dátumami alebo mesiacmi v roku. Spôsob zadania návštevy si používateľ vyberie z výberu možnosti, ktoré mu aplikácia ponúkne a následne zadá návštevnosť. Podľa výberu spôsobu zadania návštevnosti sa zobrazia dané vstupné polia pre uvedenie návštevy. Možnosť uviesť aj čas návštevy je voliteľný. Potom ako používateľ zadá potrebné informácie musí pre uloženie kliknúť na zelené tlačidlo s textom "Save".

- "edit icon"- zobrazí jednotlivé obrázky, ktoré môžu byť zvolené ako obrázok bodu na mape reprezentujúci daný mapový objekt. Pre nastavenie je postačujúce kliknutie na daný obrázok.
- "edit notes"- zobrazí textové pole, do ktorého si môže používateľ zapísat poznámky. V prípade , že k danému objektu sú napísane poznámky, tak sa aktuálny obsah poznámok zobrazí v tomto textovom poli. Po napísaní alebo editovaný poznámok musí používateľ pre uloženie poznámok stlačiť zelené tlačidlo s textom "Save notes"



Obrázek 6.4: Ukážka výzoru časti obdĺžnika zobrazujúcej : obrázok, informácie o objekte, zobrazenie detailov s použitím filtrejacie

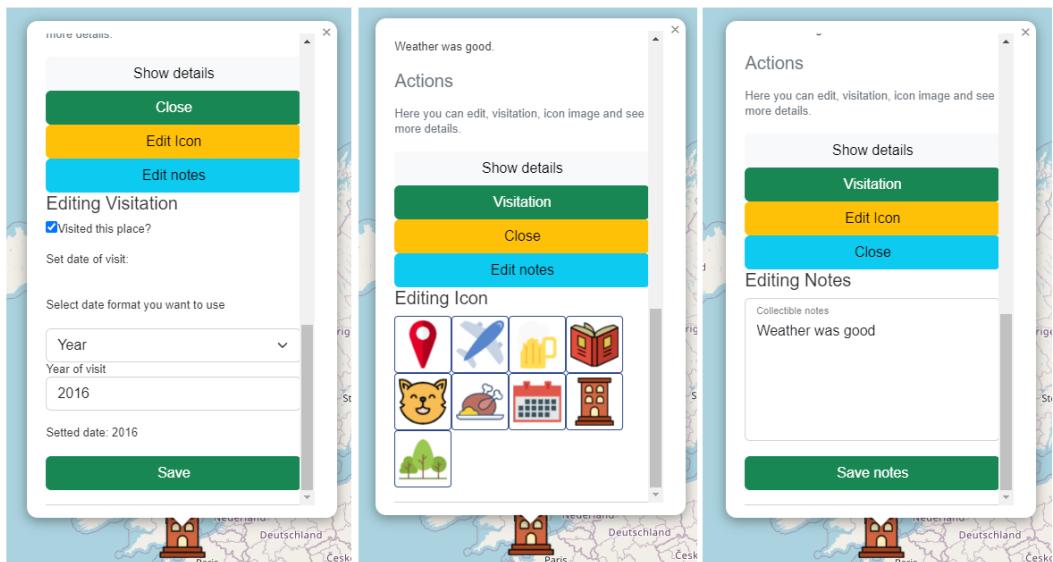
6.2 Vyhľadávanie mapových objektov

Tento stav sa zobrazí používateľovi po kliknutí tlačidla s textom "Find collectibles" v hornom navigačnom menu.

V tomto stave používateľ môže zadať jednotlivé parametre, na základe ktorých aplikácia vyhľadá množinu mapových objektov splňujúcich všetky poskytnuté parametre.

Parametre sa zadávajú postupne v jednotlivých krokoch. V každom kroku používateľ je schopný sa vrátiť o krok naspäť a zadať znova predchádzajúci parameter.

Jednotlivé kroky sú v nasledujúcich sekciách :



Obrázek 6.5: Ukážka výzoru časti obdĺžnika zobrazujúcej : nastavovanie návštěvnosti, zmena obrázka bodu, editovanie poznámok

Výber kategórii

Tento krok zobrazí vyhľadávacie pole, ktoré po zadaní textu začne dynamicky vyhľadávať možné kategórie na základe poskytnutého textu. Nájdene možnosti sú zobrazené v zozname pod týmto vyhľadávacím polom. Vždy je zobrazený názov kategórie a pod ním nasleduje sivou farbou popis danej kategórie. Kategória je nastavená potom ako používateľ klikne na danú nájdenú možnosť. Je možné nezvoliť žiadnu kategóriu. To používateľ docieli stlačením Žiadne návrhy tlačidla "Anything". Po tom ako je kategória vybraná môže používateľ pokračovať na ďalší krok stlačením zeleného tlačidla s textom "Continue" alebo vybrať inú kategóriu stlačením modrého tlačidla "Choose other". Ešte pred tým ako používateľ klikne na tlačidlo "Continue", si môže vybrať jednu alebo viac pod-kategórii, ktoré reprezentujú to, že nájdene objekty nesmú byť tejto kategórie. Vyhľadávanie funguje rovnako ako pri kategórii až na to, že v tomto prípade vo vyhľadávacom poli na začiatku je aj tlačidlo "Show all". Po stlačení tohto tlačidla aplikácia vyhľadá úplne všetky možné pod-kategórie zvonnej kategórii.

Vyber spôsobu zadania oblasti a definovanie oblasti

Tento krok zobrazí najprv štyri možnosti na výber spôsobu definovania oblasti, v ktoré sa musia nachádzať hľadané objekty. Po tom ako používateľ klikne na danú možnosť pomocou tlačidla "Use" aplikácia zobrazí potrebné prostriedky pre zadanie oblasti. Nasledujú výpis týchto možností:

- Radius - zobrazí interaktívnu mapu s jedným bodom na mape a modrým okruhom okolo bodu. Modry kruh reprezentuje oblasť, v ktorej sa bude vyhľadávať. Bod je možné presúvať chytením a posúvaním po mape. Na ľave od mapy je zobrazený panel, v ktorom sú tri tlačidla. Jedno vráti používateľa na vyber spôsobu zadania oblasti, druhé pokračuje na ďalší krok a tretí zobrazí širší panel, v ktorom môže používateľ nastaviť polomer

Type of collectibles choosing

Choosed type "cave" [Choose other](#)

Also you can choose some sub types, which will be ignored during searching process
For example: If you choose type "castle", then it also search for castles that are ruined, so if you want to skip these, choose here "castle ruin"
Clicking on "Show all", it shows you all possible sub types
Please take in mind that, if you choosed "anything", it would tried to show a large amount of sub types and it takes a very long time to process

Show all ice cave [x](#)

ice cave
natural cave, often lava or limestone, containing percolated water at temperature ≤0°C.

[Continue](#)

Obrázek 6.6: Ukážka výberu kategórie. Používateľ vybral kategóriu "cave" a následne vyhľadáva pod-kategórie s názvom "ice cave".

Choose the way of selecting area, in which collectibles will be searched.

[Back to type choosing](#)

Radius
Find and then set center of circle, also set distance from center. It will search for collectibles in this circle.
[Use](#)

Administrative Area
Search for administrative area in world. It will search for collectibles, which lies in this administrative area. You can also select here country.
[Use](#)

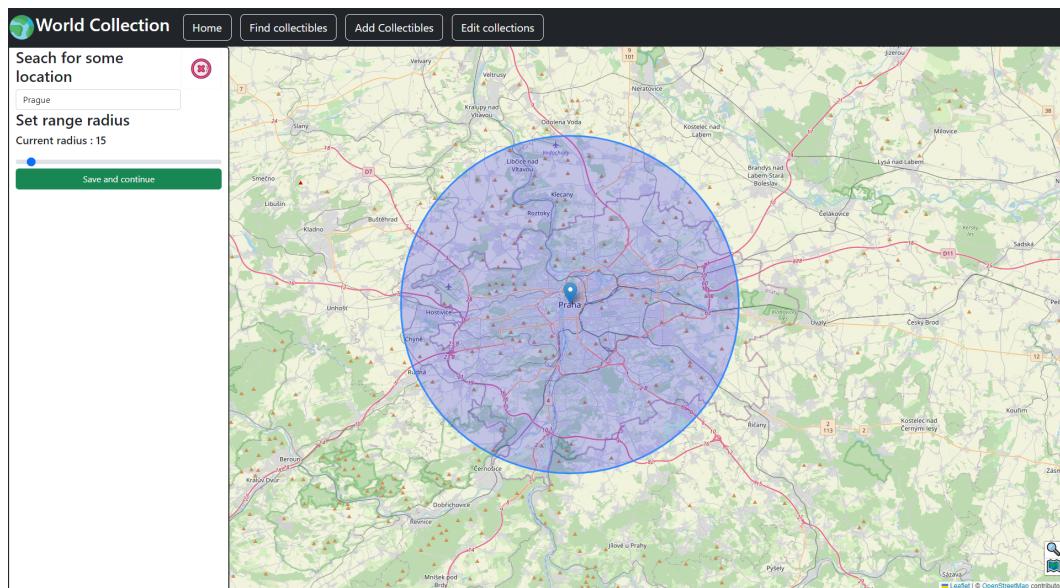
Region
Search for region like Europa , East Asian ... It will search for collectibles, which lies in country that belongs to the selected region.
[Use](#)

World
It will search for collectibles, which lies somewhere in the world. Please take in mind that, int this type it must search all around world, so do not use it if your query will contains a huge amount of collectibles. Use for example if searching for all UNESCO heritage places.
[Use](#)

Obrázek 6.7: Vyber spôsobu zadania oblasti.

kruhu. Ten sa nastaví posúvaním bodu v bare. Rozsah je od 1km po 250 km. Tento širší panel zároveň ponúka možnosť na základe názvu vyhľadať lokáciu mapového objektu, na ktorý sa bod určujúci stred kruhu presunie.

- Administratívne územie - zobrazí vyhľadávacie pole, ktoré vyhľadáva súčasné administratívne územia, pod ktoré spadajú aj jednotlivé štáty. Vyhľadanie a spôsob výberu je rovnaký ako u vyhľadávaní kategórii. Potom ako je územie vybrane aplikácia umožní používateľovi zadať aj administratívne územia, ktoré spadajú pod vybrané administrovane územie, ako výnimky. To znamená, že hľadané mapové objekty sa musia nachádzať vo zvolenom administratívnom území, ale zároveň nepatria do administratívneho územia definovaného ako výnimka.
- Región - zobrazí vyhľadávacie pole, ktoré vyhľadáva veľké územne celky ako regióny. Napríklad "Európa", alebo "stredná Európa". Vyhľadávacie pole obsahuje aj tlačidlo "Show all", po stlačení ktorého sa vyhľadajú všetky možnosti zadania regiónu.
- cely svet - nezobrazí nič, pretože cely svet nie je potrebne definovať a rovno



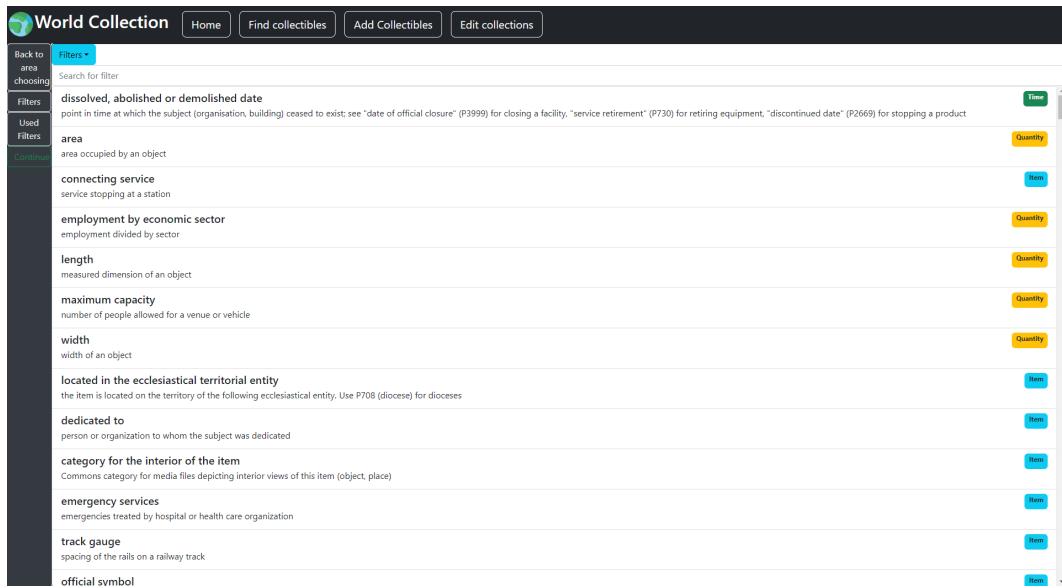
Obrázek 6.8: Ukážka zadania oblasti vyhľadávania okolo bodu.

sa preskočí na ďalší krok

Aplikácia filtrov

V tomto kroku používateľ môže aplikovať jednotlivé filtre na vyhľadávanie. Kazdy filter reprezentuje nejakú vlažnosť ako napríklad "nadmorská výška", "populácia" a podobne. Nájdene mapové objekty musia zadanú vlastnosť splňať. Aplikácia v tomto kroku zobrazí na lavej strane úzky panel s možnosťami na pokračovanie na ďalší krok, vrátenie sa na vyber oblasti, zobrazenie zoznamu možných filtrov a zoznam už aplikovaných filtrov. V zozname filtrov si môže používateľ vybrať, či chce aby boli zobrazené odporúčané filter pre danú zadanú kategóriu, alebo všetky možné filtre. Prepínanie medzi odporúčanými a všetkými filtrami je uskutočnené pomocou tlačidla "Filters". Filtre sa dajú vyhľadávať pomocou vyhľadávacieho pola. Každý filter obsahuje svoj názov, popis a na lavej strane aj tag reprezentujúci typ filtra. Po kliknutí na filter sa zobrazia prostriedky na zadanie hodnoty daného filtru. Pre každý typ filtra sa zobrazia iné prostriedky. Výpis typov filtrov:

- "item"- zadávanie hodnoty ako entita, objekt. Zobrazí buď selektor s možnosťami, ktoré sa dajú použiť ako hodnota filtru, alebo vyhľadávacie pole, v ktorom vie používateľ vyhľadať jednotlivé entity, objekty vhodné na požitie pre daný filter.
- "quantity"- zadávanie hodnoty ako desatinne číslo pomocou vstupného pola, v danom rozsahu, ktorý je zobrazení taktiež používateľovi. Aplikácia zároveň zobrazí selektor s možnosťami porovnávacích operátov. Zvolený z nich bude použitý vo filtro. Pre niektoré filtre zobrazí aplikácia pomocou selektoru aj možnosť zadat jednotku, v ktorej bude zadaná hodnota vyjadrená.
- "Time"- zadanie hodnoty ako bodu v čase. Aplikácia zobrazí selektor s možnosťami zadania času. Čas sa dá vyjadriť ako dátum, mesiac a rok, rok



Obrázek 6.9: Zoznam “všetkých filtrov”.

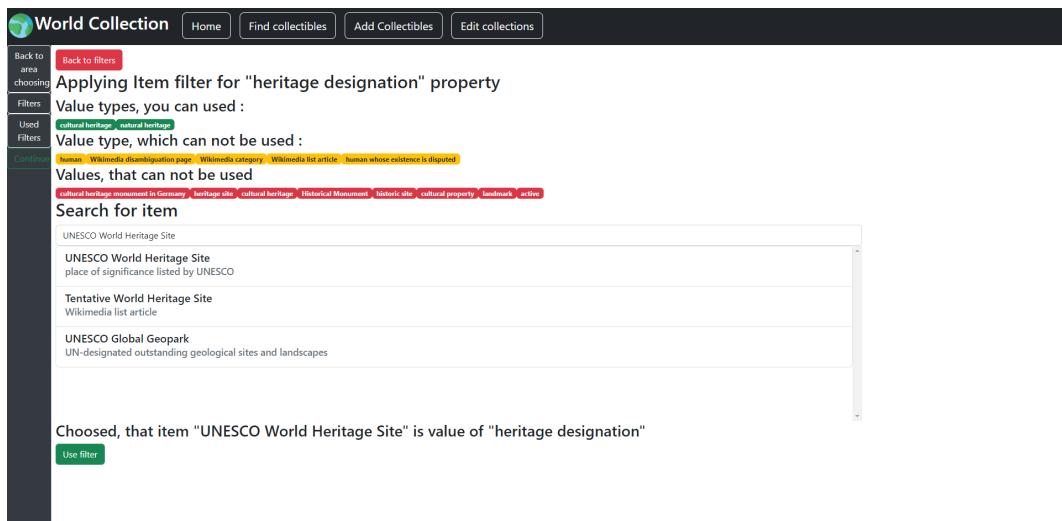
alebo ako storočie. Na základe vybratej možnosti aplikácia zobrazí vhodne vstupné polia na zadanie času. Aby bol filter možne aplikovať je potrebne aby používateľ vybral zo selektoru porovnávací operátor, ktorý sa použije vo filtroch.

V zozname použitých filtrov si môže používateľ pozrieť aké filtre sú už aplikované, alebo ich zrušiť pomocou tlačidla s textom ”remove”.

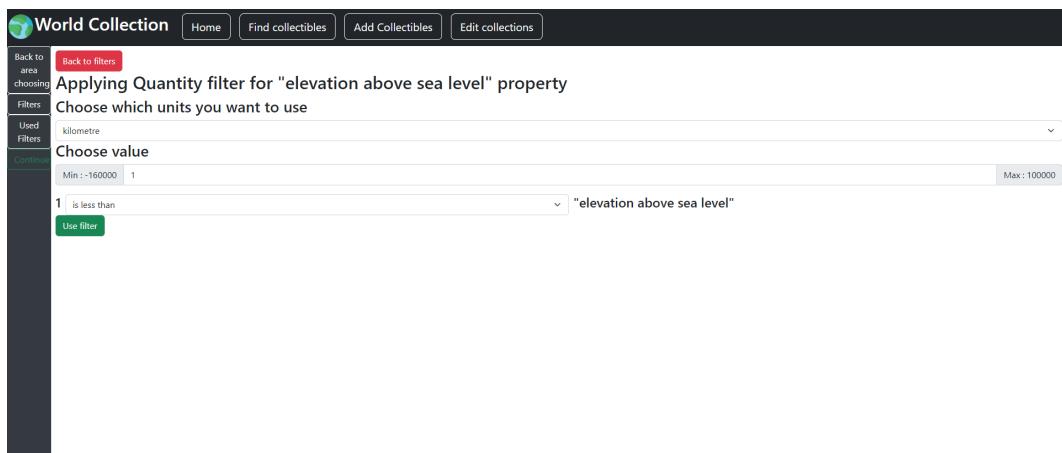
Prehľad výsledkov

Po poslednom kroku začne aplikácia vyhľadávať na základe poskytnutých parametrov mapové objekty. Ak vyhľadávanie neskončí do 60. Sekúnd, aplikácia preruší vyhľadávanie a informuje o tom používateľa. V opačnom prípade vyhľadávanie skončí a nájdene mapové objekty sa zobrazia používateľovi buď v tabuľke, alebo ako body na interaktívnej mape. Spôsob prezentácie nádejnych mapových objektov si vie používateľ vybrať pomocou tlačidla "View". Nájdené mapové objekty si vie používateľ filtrovať na základe mena alebo kategórie pomocou textových polí poskytnutých aplikáciou. Každý nájdený mapový objekt môže používateľ tlačidlom "Remove" zmazať s množinou nájdených mapových objektov, editovať jeho meno stlačením tlačidla "Edit", alebo zobraziť detaile mapového objektu tlačidlom "Show details". V mapovom pohľade sa informácie a detaile zobrazia po kliknutí na bod reprezentujúci daný mapový objekt. Aplikacia ponuka možnosť vratisť cinnosť editovania, alebo zmazania pomocou tlačidla "Undo".

Po tom ako používateľ je spokojný z množinou, si objekty v nej následne môže uložiť pomocou tlačidla "Save Collectibles". Toto tlačidlo zobrazí selektor s výberom existujúcich kolekcii, do ktorých môžu byť mapové objekty uložené. v prípade, že používateľ chce vytvoriť novu kolekciu a do nej uložiť objekty, tak stlačí tlačidlo "Create new Collection". Aplikácia zobrazí vstupné textové pole, kde používateľ zadá názov novej kolekcie a následne ju uloží tlačidlom "Save". Potom sa vráti naspäť do výberu kolekcii, kde už bude na vyber aj novo vytvorená kolekcia.



Obrázek 6.10: Ukážka zadania filtrov. Filter "kultúrne dedičstvo"s hodnotou "UNESCO"nájde objekty patriace do zoznamu UNESCO



Obrázek 6.11: Ukážka zadania filtrov. Filter "nadmorská výška", zadaná hodnota "1", zvolené jednotky "kilometre" a operátor "menší než" nájde objekty, ktorých nadmorská výška je väčšia ako 1000 metrov.

6.3 Pridávanie mapových objektov

Tento stav sa zobrazí používateľovi po kliknutí tlačidla "Add collectibles" v hornom navigačnom menu.

V tomto stave si používateľ na základe názvu vyhľadá mapový objekt, ktorý si následne môže uložiť do niektornej zo svojich kolekcií.

Aplikácia zobrazí interaktívnu mapu sveta a na ľavej strane od nej zobrazí úzky panel s jedným tlačidlom. Po stlačení tohto tlačidla sa panel rozšíri. V rozšírenom panely je vyhľadávacie textové pole na vyhľadávanie mapových objektov. Po tom ako používateľ vyhľadá a zvolí mapový objekt sa na mape zobrazí bod reprezentujúci daný mapový objekt. Zároveň sa pohľad mapy zmení tak aby pohľad bol sústredený na tento bod. Zvolený bod sa dá následne tlačidlom "Add", v tom istom panely, pridať do zoznamu mapových objektov. Ten je zobrazený v panely pod vyhľadávacím textovým polom. Pridaný mapový objekt v zoznamu dokáže

World Collection

Home Find collectibles Add Collectibles Edit collections

Back to filters

Applying Time filter for "inception" property

Select time representation you want to use :

Century

Choose Century:

Century : 12

Select comparison operator

Selected time is equal to "inception"

Use filter

Obrázek 6.12: Ukážka zadania filtrov. Filter "vznik"s hodnotou "12. storočie" a porovnávacím operátom "rovná sa" nájde objekty ktoré vznikli v 12. storočí.

World Collection

Home Find collectibles Add Collectibles Edit collections

Collectibles

View

Search for name

Search for sub-type

Undo Save Collectibles

38 results

#	Name	Sub-Type of	
1	Migovec System	cave	Details Edit Remove
2	Zadaz Cave	cave	Details Edit Remove
3	Hell Cave	cave show cave	Details Edit Remove
4	Otok Cave	archaeological site solutional cave	Details Edit Remove
5	Predjama	archaeological site solutional cave	Details Edit Remove
6	Črnelsko brezno	cave	Details Edit Remove
7	Huda luknja	cave archaeological site	Details Edit Remove
8	Potok Cave	cave archaeological site	Details Edit Remove
9	Bestažovca	cave	Details Edit Remove
10	Velika Pasica	cave	Details Edit Remove

<< prev 1 / 4 next >> 10

Obrázek 6.13: Ukážka výsledkov vyhľadania v tabuľke.

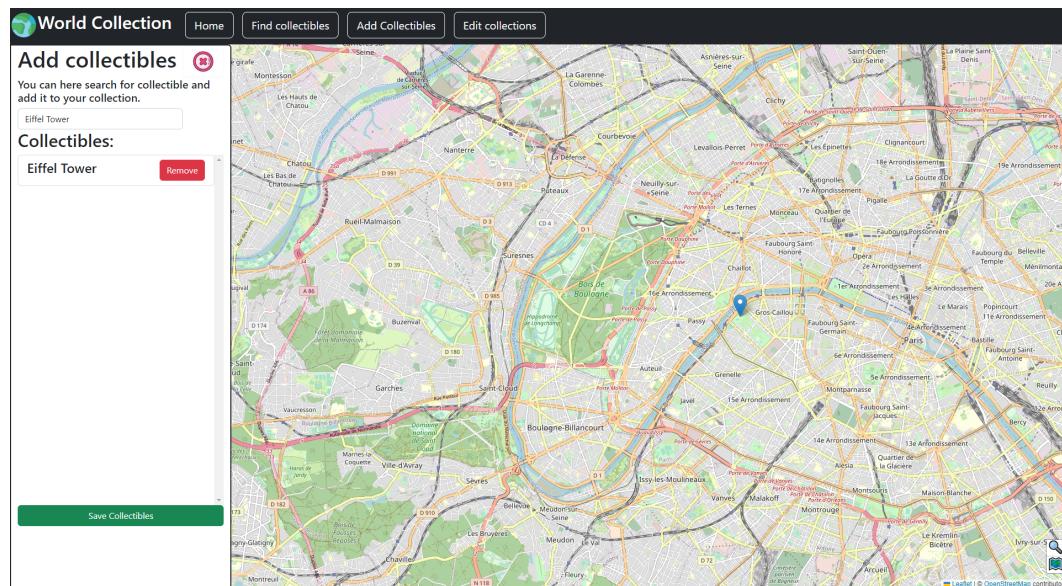
používateľ tlačidlom "Remove" odobráť. Na spodku panelu je zelené tlačidlo "Save Collectibles", ktoré sa ale zobrazí iba ak je aspoň jeden mapový objekt pridaný do zoznamu. Po kliknutí na tlačidlo sa používateľovi zobrazí rovnako ako u ukladania nájdených mapových objektov možnosť na uloženie objektov do kolekcií alebo vytvorenie novej kolekcie.

6.4 Spravovanie kolekcii

Tento stav sa zobrazí používateľovi po kliknutí tlačidla "Edit collections" v hornom navigačnom menu.

V tomto stave môže používateľ editovať svoje kolekcie a mapové objekty v nich.

Aplikácia zobrazí tabuľku, v ktorej každý riadok reprezentuje kolekciu. Kolekcie sa dajú filtrovať pomocou vstupného textového pola. Pre každú kolekciu tabuľka zobrazí počet mapových objektov, ktoré obsahuje a možnosť kolekciu:



Obrázek 6.14: Ukážka pridania mapového objektu do zoznamu.

- zmazať - vymaže kolekciu a aj všetky mapové objekty, ktoré do nej patria
- spojiť - tlačidlo "Merge". Zobrazí selektor so všetkými zvyšnými kolekciami, okrem tej pre ktorú je táto možnosť zvolená. Potom ako používateľ vyberie kolekciu a stlačí tlačidlo "Merge" sa všetky mapové objekty kolekcie presunú do zvolenej kolekcie v selektore. Kolekcie sa spoja do kopii.
- editovať - zmeniť názov kolekcie, alebo zvoliť obrázok bodu pre všetky mapové objekty v kolekcií. Vhodné ak používateľ má uložené v kolekcií také mapové objekty, ktoré chce aby všetky mali rovnaký obrázok ikonky bodu.
- editovať mapové objekty - pomocou tlačidla "Edit Collectibles", zobrazí sa podobná tabuľka, ale teraz v nej nebudú kolekcie, ale mapové objekty zvolenej kolekcie. V tomto bode vie používateľ jednotlivé mapové objekty zmazať alebo meniť pomocou tlačidla "Edit" názov mapového objektu. Tlačidlom "Back to Collections" sa používateľ vráti na tabuľku s kolekciami.

#	Name of collection	#Collectibles
1	Ice caves 2	11
2	Big cities	132

Set Icon for all collectibles Save Cancel

Edit Collectibles Edit Merge Remove

Obrázek 6.15: Ukážka spravovania kolekcii a zmena mena kolekcie.

7. Testovanie

V tejto kapitole si popíšeme testovanie aplikácie. Nazačiatku zvolíme vhodnú technológiu na testovanie webovej aplikácie, popíšeme v krátkosti základ vybranej technológie, ako inštalácia a spustenie. Na záver si prejdeme testy, ktoré majú za úlohu našu aplikáciu otestovať.

7.1 Playwrig

Pre testovanie aplikácie sme zvolili technológiu Playwrig. Jedná sa o Node.js knižnicu, ktorá podporuje moderné renderujúce nástroje ako Chromium, WebKit a Firefox. Knižnicu Playwrig môžeme používať v rôznych jazykoch a jeden z nich je aj Typescript, v ktorom sme implementovali frontend časť aplikácie a zároveň v ním budeme písat zdrojové kódy jednotlivých testov. Pri testovaní sa každý test testuje izolované od ostatných. Inak povedane sa pre každý test vytvorí novy kontext prehliadača.

Použitím Playwrig sa vieme navigovať na rôzne URL adresy a potom na nich interagovať s elementami nachádzajúcimi sa na stránke. V našich testoch budeme simulaovať používateľa, ktorý používa aplikáciu. V každom teste používate príde na URL. Následne so stránkou interaguje pomocou tlačidiel a vstupných polí, do ktorých zadáva hodnoty. Všetky tieto kroky testami nasimulujueme a potom overíme či sa na stránke zobrazili správne elementy obsahujúce správny kontext.

7.2 Inštalácia a spustenie

Inštalácia

Playwrig sa inštaluje veľmi jedného použitím npm. V termináli z adresára world-collection-app spustíme nasledujúci príkaz:

```
npm init playwright@latest
```

Počas inštalácie si vyberieme jazyk Typescript a pomenujeme adresár, v ktorom budeme mať dokumenty obsahujúce testy. Po inštalácii sa nám vytvorí adresár tests a tests-examples obsahujúce základné ukážkové testy pre získanie lepšej predstavy ako sa píšu testy. Zároveň sa nám do dokumentu package.json pridajú závislosti potrebné na spúštanie testy.

Spustenie

Bez zmenenia konfigurácie v dokumente playwright.config.ts sa testy budú spustené pre tri prehliadače a to chromium, firefox a WebKit. Spustené testy nám neotvoria žiadne okno v prehliadači, sú takzvane "headless". Výsledky testov sa nám zobrazia v termináli.

Nasledujúcim príkazom spustíme všetky testy:

```
npx playwright test
```

Aby testy správne otestovali aplikáciu je potrebné aby bola spustená backend aj frontend časť aplikácie. V opačnom prípade testy zlyhajú.

VScode rozšírenie

Pre jednoduchšie testovanie si nainštalujeme rozšírenie “Playwright Test for VSCode” do nášho vývojového prostredia VSCode. Toto rozšírenie nám umožní napríklad spustiť testy jedným kliknutím, alebo spustiť viacero testov naraz. Zároveň nám to poskytne mechanizmus na vyber lokátora v HTML dokumente, ktorým získame konkrétny element.

7.3 Playwright testy

Štruktúra testov

Všetky testy nájdeme v adresári world-collection-app/tests. V tomto adresári nájdeme päť dokumentov obsahujúcich skupinu testov a dokument MocksData.ts. V každom dokumente sú testy, ktoré spolu testujú jeden stav aplikácie. Naša aplikácia má štyri stavy, a preto pre každý jeden z nich vytvoríme nasledujúce dokumenty:

- Prehľad kolekcii - collectionsOverview.spec.ts
- Vyhladanie mapových objektov - findCollectibles.spec.ts
- Pridanie mapového objektu - addCollectible.spec.ts
- Editácia - editCollections.spec.ts

V stave “Prehľad kolekcii” môže používateľ kliknúť na bod nachádzajúci sa na interaktívnej mape. Po kliknutí na tento bod sa zobrazí “okienko” obsahujúce informácie o mapovom objekte a možnosti modifikovať poznámky a podobne. Správnosť tohto okienka otestujeme testami v samostatnom dokumente collectibleCard.spec.

Mocks dátá

Aby sme otestovali aplikáciu je potrebné získať dátá z backendu. Dátá poškrytnuté z WikidataAPI sú v poriadku, pretože od ničoho nezávisia. Problém je ale z dátami z WorldCollectionAPI. Tieto dátá už závisia od toho, čo je uložené v databáze. Z toho dôvodu použijeme Mock API požiadavky.

```
import { collectibles, collections } from "./MocksData";
//...
await page.route(
    'http://localhost:3000/WorldCollectionAPI/
    get/collections?data=%7B%7D',
    async route => {
        const json = collections;
        await route.fulfill({ json });
    }
)
```

```
    }
};

//....
```

Predchádzajúca ukážka kódu sa postará o to aby počas testovania sa všetky volania na uvedený URL koncový uzol zachytili. To znamená, že skutočne volania nebudú uskutočnené a vrátene výsledky sa nahradia testovacími dátami. Testovacie dátá nájdeme v dokumente MocksData.ts. V ňom vytvoríme dve Json testovacie dátá reprezentujúce kolekcie a mapové objekty patriace do kolekcii. Tieto dve entity nám reprezentujú databázu.

Štruktúra testu

Každý jeden dokument obsahuje skupinu testov. V tejto skupine vytvoríme konkrétné testy. Každý test berie ako argument názov testu a testovaciu funkciu. Táto funkcia berie jeden argument a to takzvaný objekt "page" reprezentujúci jednoducho povedané webovú stránku.

Aby sme nemuseli pred každým testom vykonávať tu istú činnosť tak využijeme možnosť definovať v skupine testov aj test "tests.beforeEach". Tento test sa automaticky spustí pred každým jedným testoviacim do skupiny testov. V tomto teste nastavíme Mock API požiadavky, aby sme dostávali testovacie dátá namiesto dát z databázy. Zároveň dostaneme stránku do stavu v akom sa pred každým testom bude nachládať.

V nasledujúcej ukážke si ukážeme jeden test:

```
test.describe("collection overview tests", () => {
  test.beforeEach(async ({ page }) => {
    // Mock API requests
    // collections
    await page.route('http://localhost:3000/WorldCollectionAPI/
      get/collections?data=%7B%7D', async route => {
      const json =
        collections
      ;
      await route.fulfill({ json });
    });
    // collectibles
    await page.route('http://localhost:3000/WorldCollectionAPI/
      get/collectibles?data=%7B%22collectionID%22%3A1%7D', async route => {
      const json =
        collectibles
      ;
      await route.fulfill({ json });
    });
    await page.goto('http://localhost:3000/');
  })

  test("open and close list of collections", async ({ page }) => {
    await expect(page.getTestId('collectionsMenu')).not.toBeVisible()
    await page.getTestId('buttonToOpenCollectionList').click();
  })
})
```

```
    await expect(page.getByTestId('collectionsMenu')).toBeVisible()
    await page.getByTestId('close collectionsMenu').click();
    await expect(page.getByTestId('collectionsMenu')).not.toBeVisible()
});
```

Tento test otestuje funkčnosť širšieho bočného panelu vo stave "Prehľad kolekcii", či dôkaze používateľ otvoriť a následne zatvoriť tento panel.

Aby sme mohli pracovať s elementami, tak je potrebne ich najprv nájsť v objekte "page". Pre výber môžeme použiť viacero spôsobov. Jeden z nich je použitý v ukážke kódu. Funkcia page.getByTestId() nám vráti element, ktorý obsahuje "data-testid".

Nasledujúca ukážka ukazuje tento element ako je označený v zdrojovom kóde:

```
<button data-testid="buttonToOpenCollectionList" ... </button>
```

V testoch často krát testujeme, či je konkrétny element viditeľný, alebo naopak neviditeľný. To môžeme vidieť aj v ukážke vyššie. Najprv otestujeme, že širší panel nie je zobrazení, po kliknutí správneho tlačidla sa tento panel zobrazí a následne kliknutím iného tlačidla sa znova nezobrazí.

7.4 Prehľad' testov

V tejto sekcii si v krátkosti popíšeme funkciu skupiny testov v jedovlivých dokumentov, čo testy v nich majú za úlohu otestovať. Testy pokrývajú iba najdôležitejšiu časť aplikácie. Aby sme otestovali úplne všetko je potrebné napísat viacej testov. Pre túto prácu nám to príde nie, až tak podstatné.

collectionsOverview.spec

Testy nachádzajúce sa v tomto dokumente majú za úlohu otestovať funkčnosť bočného panelu. Tým myslíme, že jeden test otestuje či je možné tento panel otvoriť a zároveň aj zatvoriť. Ďalej či je v panely správne vykreslený zoznam obsahujúci informácie o kolekciách. Funkčnosť vstupného pola, ktoré sa nachláda v tomto panely. Potom ako do neho používateľ uvedie názov kolekcie, tak sa zobrazí v zozname iba táto kolekcia. Posledným testom otestujeme, či po sa kliknutí na bod zobrazí na stránke "okienko" obsahujúce informácie.

collectibleCard.spec

Týmito testami otestujeme vyššie spomenuté okienko. Nájdeme tu testy testujúce jednotlivé elementy ako zobrazenie obrázku, informáciu o mapovom objekte a tlačidlá, ktoré po kliknutí vykreslia správne elementy.

addCollectible.spec

Nájdeme tu testy, ktoré otestujú funkčnosť vyhľadávania mapového objektu, jeho pridávania do a zmazania zo zoznamu. Zároveň je tu jeden test na otestovanie bočného panelu. Ten sa otestuje rovnakým princípom ako bol test v collectionsOverview.spec.

findCollectibles.spec

V tejto skupine testov nájdeme tri testy. Každý jeden z nich ma za úlohu otestovať jeden krok zadávania parametrov počas procesu vyhľadávania skupiny mapových objektov na základe parametrov. V prvom teste otestujeme výber kategórie. Nasimulujeme, že používateľ zadá názov kategórie do vyhľadávacieho pola a následne vyberie hľadanú kategóriu. Týmto otestujeme aj WikidataAPI, či správne funguje a vracajú správny výsledok.

V druhom teste zvolíme vyber oblasti a zadáme oblasť. V poslednom teste otestujeme vybranie filtra z ponúknutých možnosti a následne zvolenie hodnoty tohto filtra.

Testy netestujú všetky možnosti. V každom kroku sa zvolí iba jedna.

editCollections.spec

Nájdeme tu dva testy. Jeden nám otestuje, či sa nám správne zobrazí editačná tabuľka pre kolekcie, obsahujúca testovacie dáta, ktoré reprezentujú kolekcie. Či informácie v nej sa správne zobrazene a zároveň, či sa zobrazia aj tlačidla reprezentujúce upravovanie kolekcií.

V druhom otestujeme podobným princípom editačnú tabuľku mapových objektov, ktorá sa ma zobrazí potom ako používateľ klikne na správne tlačidlo nachádzajúce sa v prvej editačnej tabuľke.

8. Ukázky

Vlastní text bakalářské práce je uspořádaný hierarchicky do kapitol a podkapitol, každá kapitola začíná na nové straně. Text je zarovnán do bloku. Nový odstavec se obvykle odděluje malou vertikální mezerou a odsazením prvního řádku. Grafická úprava má být v celém textu jednotná.

Práce se tiskne na bílý papír formátu A4. Okraje musí ponechat dost místa na vazbu: doporučen je horní, dolní a pravý okraj 25 mm, levý okraj 40 mm. Číslují se všechny strany kromě obálky a informačních stran na začátku práce; první číslovaná strana bývá obvykle ta s obsahem.

Písmo se doporučuje dvanáctibodové (12 pt) se standardní vzdáleností mezi řádky (pokud píšete ve Wordu nebo podobném programu, odpovídá tomu řádkování 1,5; v \TeX u není potřeba nic přepínat). Pro běžný text používejte vzpřímené patkové písmo. Text matematických vět se obvykle tiskne pro zdůraznění skloněným (slanted) písmem, není-li k dispozici, může být zastoupeno kurzívou.

Primárně je doporučován jednostranný tisk (příliš tenkou práci lze obtížně svázat). Delší práce je lepší tisknout oboustranně a přizpůsobit tomu velikosti okrajů: 40 mm má vždy *vnitřní* okraj. Rub titulního listu zůstává nepotištěný.

Zkratky použité v textu musí být vysvětleny vždy u prvního výskytu zkratky (v závorce nebo v poznámce pod čarou, jde-li o složitější vysvětlení pojmu či zkratky). Pokud je zkratek více, připojuje se seznam použitých zkratek, včetně jejich vysvětlení a/nebo odkazů na definici.

Delší převzatý text jiného autora je nutné vymezit uvozovkami nebo jinak vyznačit a řádně citovat.

8.1 Jednoduché příklady

Čísla v českém textu obvykle sázíme v matematickém režimu s desetinnou čárkou: $\pi \doteq 3,141\,592\,653\,589$. V matematických textech se považuje za přípustné používat desetinnou tečku (pro lepší odlišení od čárky v roli oddělovače). Numerické výsledky se uvádějí s přiměřeným počtem desetinných míst.

Mezi číslo a jednotku patří úzká mezera: šířka stránky A4 činí 210 mm, což si pamatuje pouze 5 % autorů. Pokud ale údaj slouží jako přívlastek, mezera vynecháváme: 25mm okraj, 95% interval spolehlivosti.

Rozlišujeme různé druhy pomlček: červeno-černý (krátká pomlčka), strana 16–22 (střední), 45 – 44 (matematické minus), a toto je — jak se asi dalo čekat — vložená věta ohrazená dlouhými pomlčkami.

V českém textu se používají „české“ uvozovky, nikoliv „anglické“.

Na některých místech je potřeba zabránit lámání řádku ($v\sim\text{\TeX}$ u značíme vlnovkou): $u\sim$ předložek (neslabičných, nebo obecně jednopísmenných), vrchol $\sim v$, před $k\sim$ kroky, $a\sim$ proto, … obecně kdekoli, kde by při rozložení čtenář „škobrt-nul“.

8.2 Matematické vzorce a výrazy

Proměnné sázíme kurzívou (to TeX v matematickém módu dělá sám, ale nezapomínejte na to v okolním textu a také si matematický mód zapněte). Názvy funkcí sázíme vzpřímeně. Tedy například: $\text{var}(X) = \mathbb{E} X^2 - (\mathbb{E} X)^2$.

Zlomky uvnitř odstavce (třeba $\frac{5}{7}$ nebo $\frac{x+y}{2}$) mohou být příliš stísněné, takže je lepší sázet jednoduché zlomky s lomítkem: $5/7$, $(x+y)/2$.

Nechť

$$\mathbb{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix}.$$

Povšimněme si tečky za maticí. Byť je matematický text vysázen ve specifickém prostředí, stále je gramaticky součástí věty a tudíž je zapotřebí neopomenout patřičná interpunkční znaménka. Výrazy, na které chceme později odkazovat, je vhodné očíslovat:

$$\mathbb{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix}. \quad (8.1)$$

Výraz (8.1) definuje matici \mathbb{X} . Pro lepší čitelnost a přehlednost textu je vhodné číslovat pouze ty výrazy, na které se autor někde v další části textu odkazuje. To jest, nečíslujte automaticky všechny výrazy vysázené některým z matematických prostředí.

Zarovnání vzorců do několika sloupečků:

$$\begin{aligned} S(t) &= \mathbb{P}(T > t), & t > 0 & \quad (\text{zprava spojitá}), \\ F(t) &= \mathbb{P}(T \leq t), & t > 0 & \quad (\text{zprava spojitá}). \end{aligned}$$

Dva vzorce se spojovníkem:

$$\left. \begin{aligned} S(t) &= \mathbb{P}(T > t) \\ F(t) &= \mathbb{P}(T \leq t) \end{aligned} \right\} \quad t > 0 \quad (\text{zprava spojité}). \quad (8.2)$$

Dva centrované nečíslované vzorce:

$$\mathbf{Y} = \mathbb{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

$$\mathbb{X} = \begin{pmatrix} 1 & \mathbf{x}_1^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^\top \end{pmatrix}.$$

Dva centrované číslované vzorce:

$$\mathbf{Y} = \mathbb{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad (8.3)$$

$$\mathbb{X} = \begin{pmatrix} 1 & \mathbf{x}_1^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^\top \end{pmatrix}. \quad (8.4)$$

Definice rozdělená na dva případy:

$$P_{r-j} = \begin{cases} 0, & \text{je-li } r-j \text{ liché,} \\ r! (-1)^{(r-j)/2}, & \text{je-li } r-j \text{ sudé.} \end{cases}$$

Všimněte si použití interpunkce v této konstrukci. Čárky a tečky se dávají na místa, kam podle jazykových pravidel patří.

$$\begin{aligned} x &= y_1 - y_2 + y_3 - y_5 + y_8 - \cdots = && \text{z (8.3)} \\ &= y' \circ y^* = && \text{podle (8.4)} \\ &= y(0)y' && \text{z Axiomu 1.} \end{aligned} \quad (8.5)$$

Dva zarovnané vzorce nečíslované:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^n f_i(y_i; \boldsymbol{\theta}), \\ \ell(\boldsymbol{\theta}) &= \log\{L(\boldsymbol{\theta})\} = \sum_{i=1}^n \log\{f_i(y_i; \boldsymbol{\theta})\}. \end{aligned}$$

Dva zarovnané vzorce, první číslovaný:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^n f_i(y_i; \boldsymbol{\theta}), && (8.6) \\ \ell(\boldsymbol{\theta}) &= \log\{L(\boldsymbol{\theta})\} = \sum_{i=1}^n \log\{f_i(y_i; \boldsymbol{\theta})\}. \end{aligned}$$

Vzorec na dva řádky, první řádek zarovnaný vlevo, druhý vpravo, nečíslovaný:

$$\begin{aligned} \ell(\mu, \sigma^2) &= \log\{L(\mu, \sigma^2)\} = \sum_{i=1}^n \log\{f_i(y_i; \mu, \sigma^2)\} = \\ &= -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mu)^2. \end{aligned}$$

Vzorec na dva řádky, zarovnaný na =, číslovaný uprostřed:

$$\begin{aligned} \ell(\mu, \sigma^2) &= \log\{L(\mu, \sigma^2)\} = \sum_{i=1}^n \log\{f(y_i; \mu, \sigma^2)\} = \\ &= -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mu)^2. \end{aligned} \quad (8.7)$$

8.3 Definice, věty, důkazy, . . .

Konstrukce typu definice, věta, důkaz, příklad, . . . je vhodné odlišit od okolního textu a případně též číslovat s možností použití křížových odkazů. Pro každý typ těchto konstrukcí je vhodné mít v souboru s makry (`makra.tex`) nadefinované jedno prostředí, které zajistí jak vizuální odlišení od okolního textu, tak automatické číslování s možností křížově odkazovat.

Definice 1. Nechť náhodné veličiny X_1, \dots, X_n jsou definovány na témž pravděpodobnostním prostoru (Ω, \mathcal{A}, P) . Pak vektor $\mathbf{X} = (X_1, \dots, X_n)^\top$ nazveme náhodným vektorem.

Definice 2 (náhodný vektor). Nechť náhodné veličiny X_1, \dots, X_n jsou definovány na témž pravděpodobnostním prostoru (Ω, \mathcal{A}, P) . Pak vektor $\mathbf{X} = (X_1, \dots, X_n)^\top$ nazveme náhodným vektorem.

Definice 1 ukazuje použití prostředí pro sazbu definice bez titulku, definice 2 ukazuje použití prostředí pro sazbu definice s titulkem.

Věta 1. Náhodný vektor \mathbf{X} je měřitelné zobrazení prostoru (Ω, \mathcal{A}, P) do $(\mathbb{R}_n, \mathcal{B}_n)$.

Lemma 2 (Anděl, 2007, str. 29). Náhodný vektor \mathbf{X} je měřitelné zobrazení prostoru (Ω, \mathcal{A}, P) do $(\mathbb{R}_n, \mathcal{B}_n)$.

Důkaz. Jednotlivé kroky důkazu jsou podrobně popsány v práci Anděl (2007, str. 29).

□

Věta 1 ukazuje použití prostředí pro sazbu matematické věty bez titulku, lemma 2 ukazuje použití prostředí pro sazbu matematické věty s titulkem. Lemmata byla zavedena v hlavním souboru tak, že sdílejí číslování s větami.

9. Tabulky, obrázky, programy

Používání tabulek a grafů v odborném textu má některá společná pravidla a některá specifická. Tabulky a grafy neuvádíme přímo do textu, ale umístíme je buď na samostatné stránky nebo na vyhrazené místo v horní nebo dolní části běžných stránek. L^AT_EX se o umístění plovoucích grafů a tabulek postará automaticky.

Každý graf a tabulku očíslovujeme a umístíme pod ně legendu. Legenda má popisovat obsah grafu či tabulky tak podrobně, aby jím čtenář rozuměl bez důkladného studování textu práce.

Na každou tabulku a graf musí být v textu odkaz pomocí jejich čísla. Na příslušném místě textu pak shrneme ty nejdůležitější závěry, které lze z tabulky či grafu učinit. Text by měl být čitelný a srozumitelný i bez prohlížení tabulek a grafů a tabulky a grafy by měly být srozumitelné i bez podrobné četby textu.

Na tabulky a grafy odkazujeme pokud možno nepřímo v průběhu běžného toku textu; místo „*Tabulka 9.1 ukazuje, že muži jsou v průměru o 9,9 kg těžší než ženy*“ raději napíšeme „*Muži jsou o 9,9 kg těžší než ženy (viz Tabulka 9.1)*“.

9.1 Tabulky

U **tabulek** se doporučuje dodržovat následující pravidla:

- Vyhýbat se svislým linkám. Silnějšími vodorovnými linkami oddělit tabulku od okolního textu včetně legendy, slabšími vodorovnými linkami oddělovat záhlaví sloupců od těla tabulky a jednotlivé části tabulky mezi sebou. V L^AT_EXu tuto podobu tabulek implementuje balík `booktabs`. Chceme-li výrazněji oddělit některé sloupce od jiných, vložíme mezi ně větší mezeru.
- Neměnit typ, formát a význam obsahu políček v tomtéž sloupci (není dobré do téhož sloupce zapisovat tu průměr, onde procenta).
- Neopakovat tentýž obsah políček mnohokrát za sebou. Máme-li sloupec *Rozptyl*, který v prvních deseti řádcích obsahuje hodnotu 0,5 a v druhých deseti řádcích hodnotu 1,5, pak tento sloupec raději zrušíme a vyřešíme to jinak. Například můžeme tabulku rozdělit na dvě nebo do ní vložit popisné řádky, které informují o nějaké proměnné hodnotě opakující se v následujícím oddíle tabulky (např. „*Rozptyl = 0,5*“ a níže „*Rozptyl = 1,5*“).
- Čísla v tabulce zarovnávat na desetinnou čárku.

Efekt	Odhad	Směrod. chyba ^a	P-hodnota
Abs. člen	-10,01	1,01	—
Pohlaví (muž)	9,89	5,98	0,098
Výška (cm)	0,78	0,12	< 0,001

Pozn.:^a Směrodatná chyba odhadu metodou Monte Carlo.

Tabulka 9.1: Maximálně věrohodné odhady v modelu M.

- V tabulce je někdy potřebné používat zkratky, které se jinde nevyskytují. Tyto zkratky můžeme vysvětlit v legendě nebo v poznámkách pod tabulkou. Poznámky pod tabulkou můžeme využít i k podrobnějšímu vysvětlení významu některých sloupců nebo hodnot.

9.2 Obrázky

Několik rad týkajících se obrázků a grafů.

- Graf by měl být vytvořen ve velikosti, v níž bude použit v práci. Zmenšení příliš velkého grafu vede ke špatné čitelnosti popisků.
- Osy grafu musí být rádně popsány ve stejném jazyce, v jakém je psána práce (absenci diakritiky lze tolerovat). Kreslíme-li graf hmotnosti proti výšce, nenecháme na nich popisky `ht` a `wt`, ale osy popíšeme *Výška [cm]* a *Hmotnost [kg]*. Kreslíme-li graf funkce $h(x)$, popíšeme osy x a $h(x)$. Každá osa musí mít jasně určenou škálu.
- Chceme-li na dvourozměrném grafu vyznačit velké množství bodů, dáme pozor, aby se neslyly do jednolité černé tmy. Je-li bodů mnoho, zmenšíme velikost symbolu, kterým je vykreslujeme, anebo vybereme jen malou část bodů, kterou do grafu zaneseme. Grafy, které obsahují tisíce bodů, dělají problémy hlavně v elektronických dokumentech, protože výrazně zvětšují velikost souborů.
- Budeme-li práci tisknout černobíle, vyhneme se používání barev. Čáry rozlišujeme typem (plná, tečkovaná, čerchovaná, …), plochy dostatečně rozdílnými intensitami šedé nebo šrafováním. Význam jednotlivých typů čar a ploch vysvětlíme buď v textové legendě ke grafu anebo v grafické legendě, která je přímo součástí obrázku.
- Vyhýbejte se bitmapovým obrázkům o nízkém rozlišení a zejména JPEGům (zuby a kompresní artefakty nevypadají na papíře pěkně). Lepší je vytvářet obrázky vektorově a vložit do textu jako PDF.

9.3 Programy

Algoritmy, výpisy programů a popis interakce s programy je vhodné odlišit od ostatního textu. Jednou z možností je použití L^AT_EXového balíčku `fancyvrb` (fancy verbatim), pomocí něhož je v souboru `makra.tex` nadefinováno prostředí `code`. Pomocí něho lze vytvořit např. následující ukázky.

```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Menší písmo:

```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

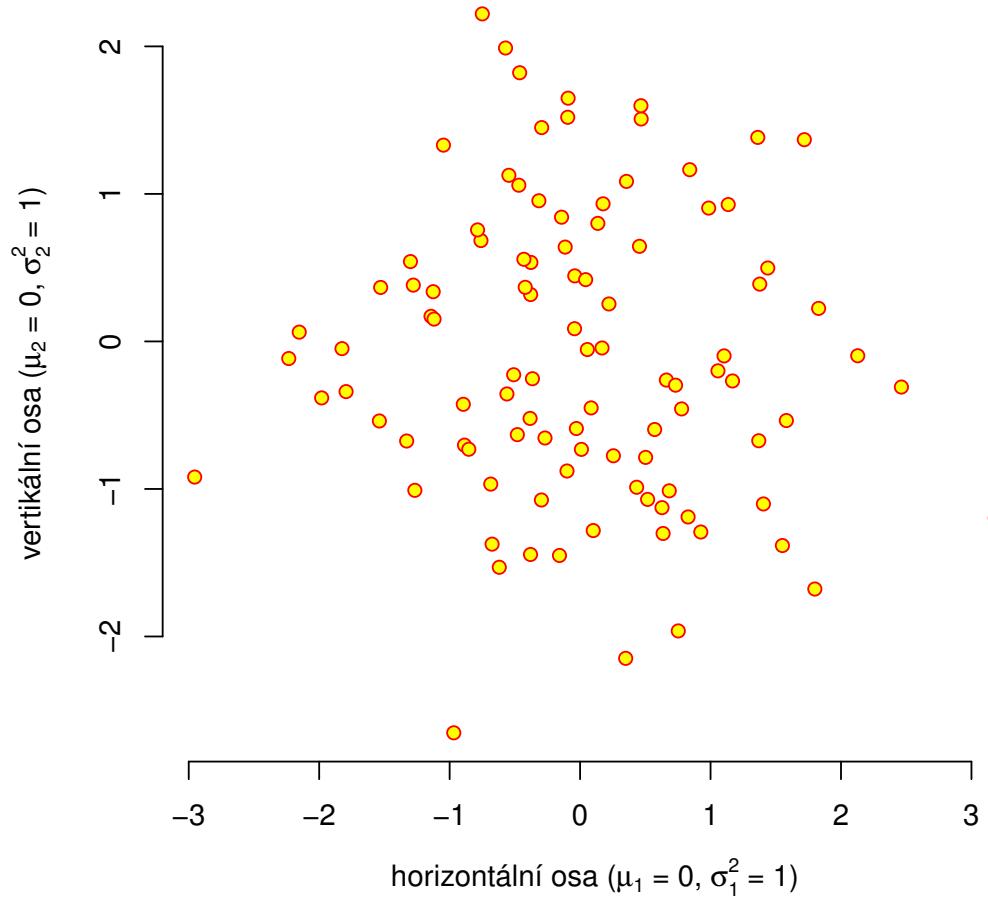
Bez rámečku:

```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

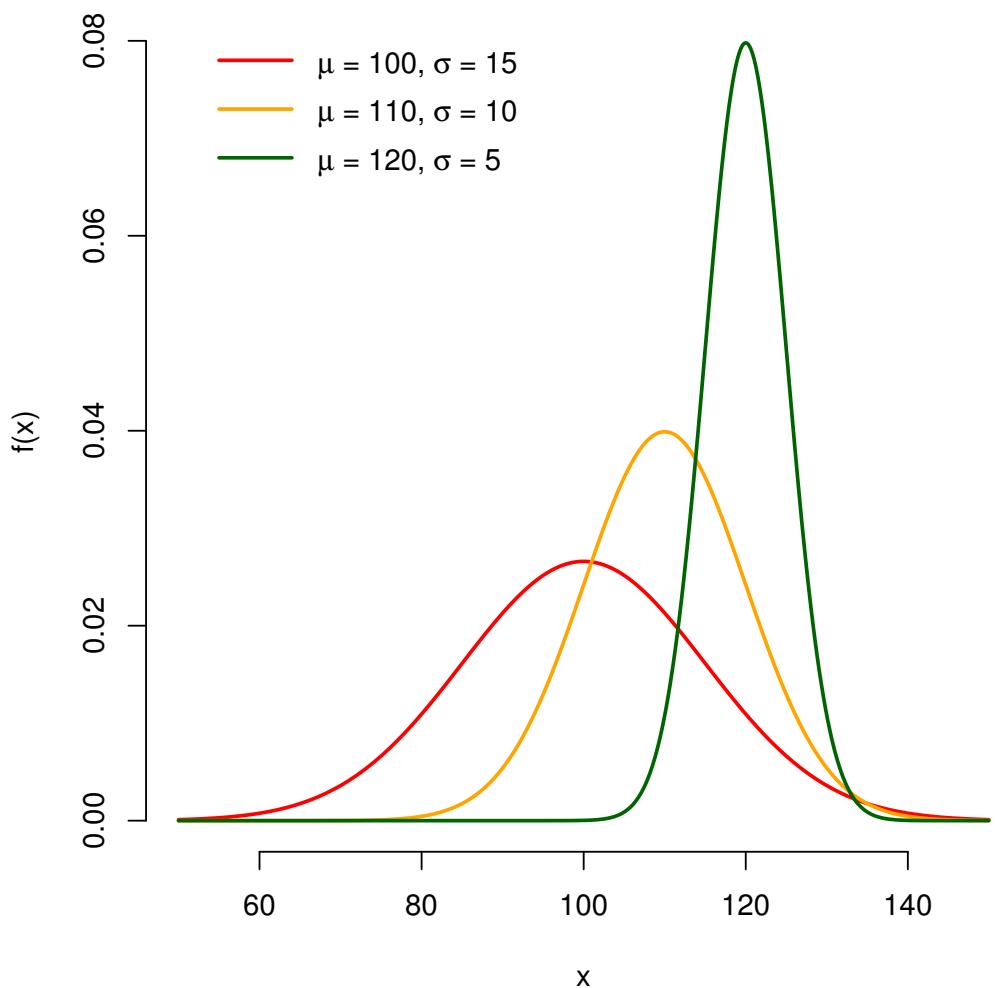
Užší rámeček:

```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Bodový graf

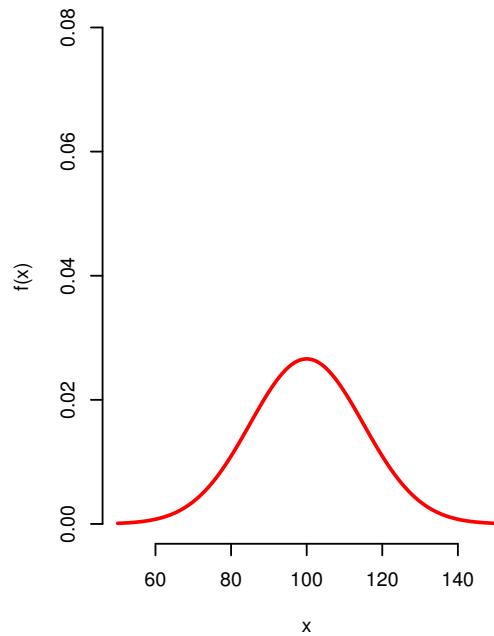


Obrázek 9.1: Náhodný výběr z rozdělení $\mathcal{N}_2(\mathbf{0}, I)$.

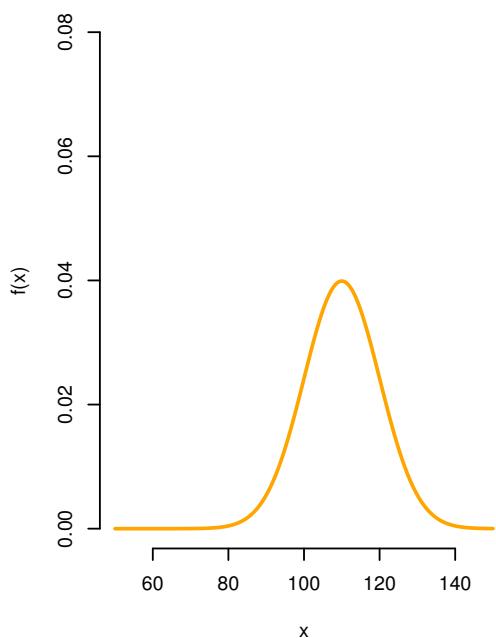


Obrázek 9.2: Hustoty několika normálních rozdělení.

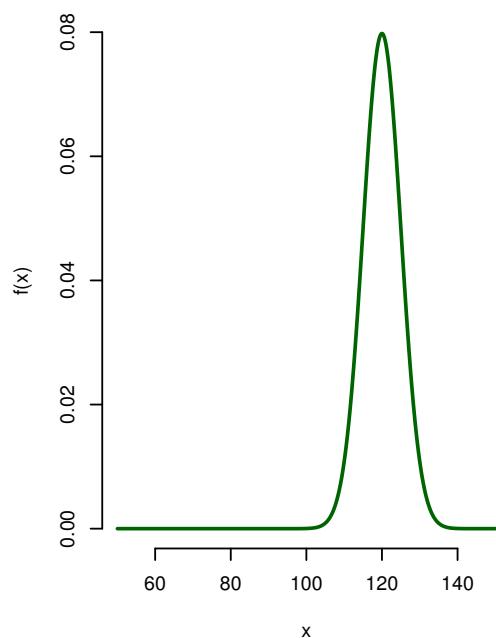
$$\mu = 100, \sigma = 15$$



$$\mu = 110, \sigma = 10$$



$$\mu = 120, \sigma = 5$$



Obrázek 9.3: Hustoty několika normálních rozdělení.

10. Odkazy na literaturu

Odkazy na literaturu vytváříme nejlépe pomocí příkazů `\citet`, `\citep` atp. (viz L^AT_EXový balíček `natbib`) a následného použití BibTeXu. V matematickém textu obvykle odkazujeme stylem „Jméno autora/autorů (rok vydání)“, resp. „Jméno autora/autorů [číslo odkazu]“. V českém/slovenském textu je potřeba se navíc vypořádat s nutností skloňovat jméno autora, respektive přechylovat jméno autorky. Je potřeba mít na paměti, že standardní příkazy `\citet`, `\citep` produkují referenci se jménem autora/autorů v prvním pádě a jména autorek jsou nepřechýlena.

Pokud nepoužíváme bibTeX, řídíme se normou ISO 690 a zvyklostmi oboru.

Jména časopisů lze uvádět zkráceně, ale pouze v kodifikované podobě.

10.1 Několik ukázek

Mezi nejvíce citované statistické články patří práce Kaplan a Meiera a Coxe (Kaplan a Meier, 1958; Cox, 1972). Student (1908) napsal článek o t-testu.

Prof. Anděl je autorem učebnice matematické statistiky (viz Anděl, 1998). Teorii odhadu se věnuje práce Lehmann a Casella (1998). V případě odkazů na specifickou informaci (definice, důkaz, ...) uvedenou v knize bývá užitečné uvést specificky číslo kapitoly, číslo věty atp. obsahující požadovanou informaci, např. viz Anděl (2007, Věta 4.22) nebo (viz Anděl, 2007, Věta 4.22).

Mnoho článků je výsledkem spolupráce celé řady osob. Při odkazování v textu na článek se třemi autory obvykle při prvním výskytu uvedeme plný seznam: Dempster, Laird a Rubin (1977) představili koncept EM algoritmu. Respektive: Koncept EM algoritmu byl představen v práci Dempstera, Lairdové a Rubina (Dempster, Laird a Rubin, 1977). Při každém dalším výskytu již používáme zkrácenou verzi: Dempster a kol. (1977) nabízejí též několik příkladů použití EM algoritmu. Respektive: Několik příkladů použití EM algoritmu lze nalézt též v práci Dempstera a kol. (Dempster a kol., 1977).

U článku s více než třemi autory odkazujeme vždy zkrácenou formou: První výsledky projektu ACCEPT jsou uvedeny v práci Genbergové a kol. (Genberg a kol., 2008). V textu *nepapišeme*: První výsledky projektu ACCEPT jsou uvedeny v práci Genberg, Kulich, Kawichai, Modiba, Chingono, Kilonzo, Richter, Pettifor, Sweat a Celentano (2008).

11. Formát PDF/A

Opatření rektora č. 13/2017 určuje, že elektronická podoba závěrečných prací musí být odevzdávána ve formátu PDF/A úrovně 1a nebo 2u. To jsou profily formátu PDF určující, jaké vlastnosti PDF je povoleno používat, aby byly dokumenty vhodné k dlouhodobé archivaci a dalšímu automatickému zpracování. Dále se budeme zabývat úrovní 2u, kterou sázíme \TeX em.

Mezi nejdůležitější požadavky PDF/A-2u patří:

- Všechny fonty musí být zabudovány uvnitř dokumentu. Nejsou přípustné odkazy na externí fonty (ani na „systémové“, jako je Helvetica nebo Times).
- Fonty musí obsahovat tabulku ToUnicode, která definuje převod z kódování znaků použitého uvnitř fontu do Unicode. Díky tomu je možné z dokumentu spolehlivě extrahovat text.
- Dokument musí obsahovat metadata ve formátu XMP a je-li barevný, pak také formální specifikaci barevného prostoru.

Tato šablona používá balíček `pdfx`, který umí \LaTeX nastavit tak, aby požadavky PDF/A splňoval. Metadata v XMP se generují automaticky podle informací v souboru `prace.xmpdata` (na vygenerovaný soubor se můžete podívat v `pdflatex.xmp`).

Validitu PDF/A můžete zkontolovat pomocí nástroje VeraPDF, který je k dispozici na <http://verapdf.org/>.

Pokud soubor nebude validní, mezi obvyklé příčiny patří používání méně obvyklých fontů (které se vkládají pouze v bitmapové podobě a/nebo bez unicodových tabulek) a vkládání obrázků v PDF, které samy o sobě standard PDF/A nesplňují.

Další postřehy o práci s PDF/A najdete na <http://mj.ucw.cz/vyuka/bc/pdfFAQ.html>.

Závěr

Seznam použité literatury

- ANDĚL, J. (1998). *Statistické metody*. Druhé přepracované vydání. Matfyzpress, Praha. ISBN 80-85863-27-8.
- ANDĚL, J. (2007). *Základy matematické statistiky*. Druhé opravené vydání. Matfyzpress, Praha. ISBN 80-7378-001-1.
- COX, D. R. (1972). Regression models and life-tables (with Discussion). *Journal of the Royal Statistical Society, Series B*, **34**(2), 187–220.
- DEMPSTER, A. P., LAIRD, N. M. a RUBIN, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, **39**(1), 1–38.
- GENBERG, B. L., KULICH, M., KAWICHAJ, S., MODIBA, P., CHINGONO, A., KILONZO, G. P., RICHTER, L., PETTIFOR, A., SWEAT, M. a CELENTANO, D. D. (2008). HIV risk behaviors in sub-Saharan Africa and Northern Thailand: Baseline behavioral data from project Accept. *Journal of Acquired Immune Deficiency Syndrome*, **49**, 309–319.
- KAPLAN, E. L. a MEIER, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, **53**(282), 457–481.
- LEHMANN, E. L. a CASELLA, G. (1998). *Theory of Point Estimation*. Second Edition. Springer-Verlag, New York. ISBN 0-387-98502-6.
- STUDENT (1908). On the probable error of the mean. *Biometrika*, **6**, 1–25.

Seznam obrázků

3.1	Ukažka Wikidát s popisom	25
4.1	Systém Kontext diagram	34
4.2	Kontajner diagram	39
4.3	Server Komponent diagram	40
4.4	SPA Komponent diagram	41
6.1	Vzhľad aplikácia po navštívený stránky	92
6.2	Prezentovanie mapových objektov danej kolekcie	92
6.3	Zobrazenie informácií pre daný bod	93
6.4	Ukážka výzoru časti obdlžníka zobrazujúcej : obrázok, informácie o objekte, zobrazenie detailov s použitím filtrácie	94
6.5	Ukážka výzoru časti obdlžníka zobrazujúcej : nastavovanie návštevnosti, zmena obrázka bodu, editovanie poznámok	95
6.6	Ukážka výberu kategórie. Používateľ vybral kategóriu "cave" a následne vyhľadáva pod-kategórie s názvom "ice cave".	96
6.7	Vyber spôsobu zadania oblasti.	96
6.8	Ukážka zadania oblasti vyhľadávania okolo bodu.	97
6.9	Zoznam "všetkých filtrov"	98
6.10	Ukážka zadania filtru. Filter "kultúrne dedičstvo"s hodnotou "UNESCO" nájde objekty patriace do zoznamu UNESCO	99
6.11	Ukážka zadania filtru. Filter "nadmorská výška", zadaná hodnota "1", zvolené jednotky "kilometre" a operátor "menší než" nájde objekty, ktorých nadmorská výška je väčšia ako 1000 metrov.	99
6.12	Ukážka zadania filtru. Filter "vznik"s hodnotou "12. storočie" a porovnávacím operátorom "rovná sa" nájde objekty ktoré vznikli v 12. storočí.	100
6.13	Ukážka výsledkov vyhľadania v tabuľke.	100
6.14	Ukážka pridania mapového objektu do zoznamu.	101
6.15	Ukážka spravovania kolekcií a zmena mena kolekcie.	102
9.1	Náhodný výber z rozdelení $\mathcal{N}_2(\mathbf{0}, I)$	115
9.2	Hustoty niekolika normálnych rozdelení.	116
9.3	Hustoty niekolika normálnych rozdelení.	117

Seznam tabulek

9.1 Maximálně věrohodné odhady v modelu M.	112
--	-----

Seznam použitých zkratek

A. Přílohy

A.1 První příloha