SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SYSTÉM PRE ZDIEĽANIE INFORMÁCIÍ A POZNATKOV V BEZPEČNOSTNEJ DOMÉNE

Tímový projekt

ZS 2020/2021

Zadávatelia tímového projektu:

Ing. Štefan Balogh, PhD.

RNDr. Igor Kossaczký, CSc.

Bc. Peter Ňaňko - 83243

Bc. Martin Lacza - 86219

Bc. Filip Furtkevič - 86294

Bc. Martin Hrebeňák - 86308

Bc. Samuel Orth - 86245

Bc. Peter Košarník - 86207

Bc. Matúš Pohančenik - 86142

Poďakovanie

Chceli by sme vyjadriť veľkú vďaku vedúcemu tejto práce, ktorým bol Ing. Štefan Balogh, PhD. za odborné rady, trpezlivosť, pevné nervy a celkovú pomoc pri tvorení riešenia tohto tímového projektu. V neposlednom rade patrí naša veľká vďaka aj Bc. Matej Rychtárik za jeho odborné rady a poskytnutie testovacích dát, ktoré boli nevyhnutnou súčasťou tohto projektu.

Obsah

Zoznam obrázkov a tabuliek

Zoznam výpisov

Slovník pojmov

Položka - Z ang. Item

Vlastnosť - Z ang. Property

Štítok - Z ang. Label

Lexéma - Z ang. Lexeme

Údaj - Z ang. Statement

Dátový typ - Z ang. Datatype

Dotaz - Z ang. Query

Popis - Z ang. Description

Plnotextové vyhľadávanie - Z ang. Fulltext search

Linky

Stránka tímového projektu so zápisnicami - `http://147.175.121.234/`

Webová aplikácia (SQID) - `http://147.175.121.153/`

Git - `https://github.com/tp1-SpZIaPvBD`

Úvod

Cieľ zadania

Cieľom zadania je návrh a realizácia servera pre zdieľanie a uchovávanie poznatkov z bezpečnostnej domény. Môžu to byť poznatky z analýzy malvér, technikách exploitov, zraniteľnostiach softvéru, a ďalšie užitočné informácie o útokoch. Pre efektívne ukladanie a zdieľanie je potrebné využiť poznatky z ontologickej a sémantickej reprezentácie dát ktoré využíva napr. `https://www.wikidata.org`. Úloha pozostáva z viac dielčich úloh a to:

- vytvorenie sémantickej databázy a ontologického modelu pre dáta, ktoré chceme zdielať
- vytvorenie nástrojov pre zber dát z dostupných zdrojov a ich transformáciu do ontologického modelu.
- vytvorenie frontend aplikácie s možnosťou vyhľadávania a zobrazovania dát z ontologickej databázy a pre ich následnú analýzu ako zdroj sémantických testovacích dát.

Úlohy

- 1. Vykonajte inštaláciu a konfiguráciu sémantických databáz
- 2. S využitím dostupných nástrojov a naprogramovaných nástrojov vykonať zber vhodných dát z internetových zdrojov a iných dostupných zdrojov (CSIRT) a ich transformácia do ontologickej reprezentácie
- 3. Vytvorte testovaciu vzorku dát z oblasti počítačovej bezpečnosti a ich sémantickú reprezentáciu pre testovanie vyhľadávania požadovaných informácií a ich súvislostí
- 4. Vytvorte web aplikáciu pre vyhľadávanie a zobrazovanie dát
- 5. Hľadajte optimálny sémantický model pre reprezentáciu dát z domény počítačová bezpečnosť v sémantickej databáze z hľadiska efektívnosti ukladania a vyhľadávania

Tím

Bc. Peter Ňaňko

Absolvent bakalárskeho štúdia FEI STU aplikovanej informatiky, doména bezpečnosť informačných systémov (BIS). V záverečnej práci som sa venoval Linux Security Modulu Medusa a aplikácií Linux Audit Systému do bezpečnostného modulu Medusa. V práci ako junior Java developer som sa venoval IoTback-endovým projektom, používal som frameworkSpring a PostgreSQL databázu, kde som sa riadil MVC dizajnom a micro-servisovou architektúrou. Veľmi rád pracujem s webovými technológiami a databázami v práci aj vo voľnom čase preto ma táto ponuka o túto tému oslovila.

Bc. Martin Lacza

V bakalárskejpráci som sa zaoberal problematike rotačnej vzdialenosti binárnych stromov. Vytvoril som aplikáciu na počítanie rotačnej vzdialenosti v jazykoch C a C++. Zamestnaný som v Orange business services ako správca sietí, využívame rôzne in-housetooly na prácu so sieťovými zariadeniami na veľké vzdialenosti (trafficanalysis and monitoring, troubleshooting, packetsniffing, atd.). Moja doména je BIS.

Bc. Filip Furtkevič

Systém pre zdieľanie informácií a poznatkov v bezpečnostnej doméne:
Som absolventom Bakalárskeho štúdia Aplikovanej informatiky na FEI STU, doména bezpečnosti informačných systémov. Ako tému bc. práce som si zvolil šifrovanie pomocou kvázigrúp, taktiež ma veľmi zaujímajú matematické štruktúry a ich využitie pre kryptosystémy. V práci som nabral niekoľko skúseností z oblasti systému administratingu ale aj informačnej bezpečnosti a databázových riešení / backendu. Veľmi rád sa učím nové technológie v akomkoľvek smere a táto téma je pre mňa preto výzvou, ktorú by som rád skúsil zvládnuť. Vzhľadom na to, že sa v zadaní pracuje s dátami na tému informačnej bezpečnosti a na backendovú časť práce s týmito dátami sa môj záujem o túto tému utvrdzuje. Tiež by som si chcel veľmi rád niekedy frontendový prístup k produktu.

Bc. Martin Hrebeňák

V bakalárskej práci som sa zaoberal strojovým učením, konkrétne generovaním hudby pomocou LSTM sietí v Pythone. V práci sa venujem podpore a testovaniu CRM aplikácie pre industriálne využitie, ktorej hlavným aspektom sú databázy a taktiež mi nie je cudzia tvorba webov a webových aplikácií.

Bc. Samuel Orth

Absolvent bakalárskeho štúdia FEI STU aplikovanej informatiky, doména bezpečnosť informačných systémov (BIS). V študijnom aj pracovnom živote som sa venoval jazykom JAVA (špeciálne frameworkuSpringBoot) a C#, v ktorých skúsenosti by mohli pri tejto práci pomôcť. Rok som sa v pracovnom živote venoval programovaniu a spracovávaniu dát vo veľkých projektoch z IoT zariadení, používal databázy postgresql. Vzhľadom na to že študujem bezpečnosť a chcem sa jej aj venovať, táto téma mi príde veľmi zaujímavá, lebo by som mohol zužitkovať už nadobudnuté schopnosti a mohol by som sa pri nej naučiť mnoho nového.

Bc. Peter Košarník

Ako študent aplikovanej informatiky, si chcem udržiavať všeobecný prehľad o veciach a byť flexibilný. Svoje problem-solving schopnosti som doteraz používal s OOP programovaním v jazyku C# a Java, pri rôznych stanovených problémoch. Jednou z mojich motivácií, je venovať sa problematike bezpečnosti a dozvedieť sa o nej viac, ako už aj napovedá moje zameranie BIS. Tento projekt je vhodný kandidát, pretože už počas vypracovávania prídeme do styku s viacerými zdrojmi v oblasti bezpečnosti.

Bc. Matúš Pohančenik

Absolvent bakalárskeho štúdia FEI STU aplikovanej informatiky, doména modelovanie a simulácia udalostných systémov. V rámci študijného, ale aj profesijného života sa zameriavam na programovanie v jazyku JAVA, predovšetkým na tvorbu webových aplikácii (front-end aj back-end) s použitím frameworkuSpring s čím je aj úzko spojená práca s databázami. V oblasti sémantického webu veľké skúsenosti nemám, no o to radšej by som ich nadobudol a rozšíril tak svoje znalosti aj v tejto oblasti. Taktiež vzdelávať sa v oblastiach bezpečnosti je v dnešnej dobe vždy len veľkou výhodou a práve preto si myslím, že pri práci na tejto téme sa môžem toho veľa naučiť a nadobudnuté vedomosti neskôr zužitkovať aj v iných oblastiach môjho profesijného života.

Motivácia

Toto zadanie je pre nás jedinečná príležitosť naučiť sa inštaláciu a konfiguráciu sémantických databáz a hlbšie sa vnoriť do tejto tematiky, ktorú je podľa nás výborné ovládať v budúcnosti ak by sme chceli robiť nejaké vlastné projekty. Taktiež sa pri tomto zadaní môžeme dozvedieť veľa nových poznatkov o bezpečnosti v informatike, zraniteľnostiach a jednotlivých útokoch. Pri zadaní sa taktiež oboznámime s novými nástrojmi na programovanie vyhľadávania a triedenia informácií. Teší nás aj, že by sme si mohli vyskúšať aj ako tím pracovať na aplikácií, ktorej treba spraviť aj backend aj frontend, tým pádom by nám to mohlo poskytnúť skúsenosť fullstack developera.

Čo môžeme poskytnúť

Našou úlohou je vytvoriť web aplikáciu pre vyhľadávanie a zobrazovanie dát, ktoré bude aplikácia zbierať pomocou nástrojov z vhodných internetových zdrojov a bude ich transformovať do ontologickej reprezentácie. Taktiež vytvoríme a nakonfigurujeme sémantické databázy a budeme hľadať oprimálny sémantický model pre reprezentáciu dát z domény počítačová bezpečnosť v tejto databáze.

Je nutné aby sa všetci členovia tímu oboznámili s danou problematikou, sémantickými databázami a ontologickou reprezentáciou. Oboznámime sa aj s nástrojmi na vyhľadávanie dát a inštaláciu databáz.

Na záver budeme hľadať čo najefektívnejšie možnosti ukladania a vyhľadávania dát. Vytvoríme metodický postup používania nášho riešenia pre ďalších vývojárov, ktoré by mohlo byť uložené napr. na git repozitári, bolo by dobre čitateľné aj vďaka verziám a bude

ďalej rozšíriteľné. Svoje výsledky a dosiahnuté ciele vypracujeme a technicky zdokumentujeme.

Predpokladané zdroje

Server na webovú aplikáciu, databázy a webovú stránku pre predmet, git repozitár, nástroj na obsluhu databáz (napr. niečo ako DBeaver)

Konštruktívne návrhy zmien organizácie predmetu

S návrhom zadania a organizáciou predmetu sme stotožnení a spokojní.

1 Analýza problematiky

1.1 Sémantický web

Sémantický web je vylepšenie World Wide Webu s cieľom rozšíriť jeho kapacitu zdieľania na zdieľanie dát aplikácií, a aj zdieľanie údajov zameraných na človeka. Prostredníctvom RDF poskytuje sémantický web zjednocujúce znázornenie bohato štruktúrovaných údajov. A prostredníctvom webu môžu vývojári aplikácií zdieľať tieto údaje. Pomocou toho môžu aplikácie robiť „rozhodnutia“ na základe kombinácií mnohých rôznych druhov údajov zverejňovaných na webe. RDF stavia na základnom mechanizme ukazovateľa webu Uniform Resource Identifier (URI).[1]

Sémantický web explicitne uvádza, že identifikátory URI možno použiť na pomenovanie ľubovoľného názvu – od abstraktného pojmu („farba“) k fyzickým objektom („budova, v ktorej pracuje personál CSAIL MIT“), taktiež aj elektronické objekty („trojový kód, ktorým sa vykonáva operačný systém Linux“). RDF používa URI na pomenovanie vzťahov medzi objektami ako aj na samotné objekty.[1]

1.1.1 Čo je to ontológia?

V informatike a informačných vedách ontológia zahŕňa reprezentáciu, formálne pomenovanie a definíciu kategórií, vlastností a vzťahov medzi pojmami, údajmi a entitami, ktoré potvrdzujú jednu, veľa alebo všetky oblasti diskurzu. Jednoduchšie, ontológia je spôsob zobrazenia vlastností predmetnej oblasti a toho, ako spolu súvisia, a to definovaním súboru pojmov a kategórií, ktoré predmet reprezentujú. [2]

Každá akademická disciplína alebo odbor vytvára ontológie s cieľom obmedziť zložitosť a usporiadať údaje do informácií a vedomostí. Nové ontológie zlepšujú riešenie problémov v rámci tejto domény. Prekladanie výskumných prác do všetkých oblastí je ľahší problém, keď odborníci z rôznych krajín udržiavajú medzi každým zo žargónov ich jazykov kontrolovanú slovnú zásobu. [2]

Medzi bežné komponenty ontológií patria:

- **Individuals** - inštancie alebo objekty (základné alebo „ground floor“ objekty)
- **Triedy** - množiny, zbierky, koncepty, typy predmetov alebo druhy vecí
- **Atribúty** - aspekty, vlastnosti, charakteristika alebo parametre, ktoré môžu mať objekty (a triedy)
- **Vzťahy** - spôsoby vzájomného prepojenia tried a individuals

- **Funkčné termíny** - zložité štruktúry vytvorené z určitých vzťahov, ktoré možno použiť namiesto jednotlivého výrazu vo výroku
- **Obmedzenia** - formálne uvedené popisy toho, čo musí byť pravdivé, aby bolo možné prijať nejaké tvrdenie ako vstup
- **Pravidlá** - výroky vo forme vety if-then (predchádzajúci-následný), ktoré popisujú logické závery, ktoré možno vyvodiť z tvrdenia v konkrétnej podobe
- **Axiómy** - tvrdenia (vrátane pravidiel) v logickej forme, ktoré spolu tvoria celkovú teóriu, ktorú ontológia popisuje v oblasti jej použitia. Táto definícia sa líši od definície axiómov v generatívnej gramatike a formálnej logike. V týchto disciplínach zahŕňajú axiómy iba výroky uplatnené ako apriori vedomosti. Axiómy zahŕňajú aj teóriu odvodenú z axiomatických výrokov.
- **Eventy** - zmena atribútov alebo vzťahov

Ontológie môžu byť reprezentované formálnymi, semi-formálnymi alebo neformálnymi jazykmi. V dnešnej dobe sa hlavný vývoj uberá predovšetkým v oblasti formálnych jazykov ontológií sémantického webu, najmä RDF a jazykov rodiny OWL.

1.1.2 RDF

Resource Description Framework je štandardný model na výmenu dát na internete. Má vlastnosti, ktoré uľahčujú zjednotenie dát. Podporuje aj vývoj schém bez vyžadovania zmeny všetkých užívateľov dát. [3] [4]

Rozširuje štruktúru linkov na internete tak, že používa URI na pomenovanie vzťahu medzi dvoma linkami. Toto sa nazýva “triple”. Použitím tohto modelu sa môžu štruktúrované a polovične štruktúrované dáta miešať či zdieľať medzi rôznymi aplikáciami. Táto štruktúra vytvára popísaný graf, ktorého okraje reprezentujú pomenované vzťahy medzi dvoma zdrojmi čo je reprezentované vrcholmi grafu. [4]

Štruktúry RDF sú teda trojice subjektov, predikátov a objektov, kde definujeme subjekt pripojením objektu pomocou predikátu.[3] [4]

1.1.3 OWL

Jazyk OWL predstavuje podobný nástroj ako RDF, avšak OWL sa používa na konkretizáciu data modelov. Pomocou neho je možné zostaviť prostriedky na prácu s databázovými dotazmi a pretransformovanie dátových modelov do reálneho sveta. [5]

Okrem toho, že OWL poskytuje tie isté RDFS atribúty, ako type, domain apod, poskytuje taktiež viac technickejšie vzťahy ako unionOf, sameAs, ktoré sa viac podobajú

dnešným moderným dopytovacím jazykom ako napr. SQL. [5] [6]

Ďalším špecifikom OWL oproti RDF je jeho menšia voľnosť a väčšia striktnosť. V RDF môže v trojiciach ("triple-och") vystupovať subjekt alebo objekt ako trieda a inštancia tej istej triedy súčasne. OWL jazyk takéto správanie nepodporuje. [5]

1.1.4 SPARQL

SPARQL je štandardný sémantický jazyk na dopyt dát, ktoré sú mapované do RDF grafov. Používa sa hlavne preto, že pomocou porovnávania vzorov sa dá navigovať v RDF grafoch. Obsahuje niekoľko zaujímavých funkcionalít ako dobrovoľné možnosti, zjednotenie vzorov či filtrovanie hodnôt. Takto sa dajú kombinovať rôzne vzory, a následne v dátach objaviť aj menej očividné vzťahy.[7] [3]

1.1.5 MediaWiki

MediaWiki je kolaboračná a dokumentačná platforma, ktorá poháňa Wikipediu. Táto platforma zbiera a usporadúva informácie, ktoré poskytuje každému. Je založená na princípe wiki, čo je decentralizovaná spolupráca v centralizovanom priestore.[8]

Naprogramovaná je v programovacom jazyku PHP a ukladá všetok textový obsah do databáz. Softvér tejto platformy je optimalizovaný na efektívne narábanie s veľkými projektami, ktoré by mohli mať aj niekoľko terabajtov obsahu a stovky tisíc zobrazení za sekundu.[8]

MediaWiki bola vytvorená pre Wikipediu v roku 2003. Dodnes sa využíva na Wikipedii a takmer všetkých Wikimedia stránkach ako aj Wiktionary, Wikimedia Commons či Wikidata. Pôvodne ju vytvoril Magnus Manske a zlepšil ju Lee Daniel Crocker, a jej ďalší rozvoj bol koordinovaný zo strany Wikimedia Foundation.[8][9]

1.1.6 Wikidata

Wikidata je platforma so základom informácií, ktoré môžu byť zobrazené a upravené kýmkoľvek. Slúži ako centrálne úložisko štruktúrovaných dát iných wiki projektov ako napr. Wikipedia, Wiktionary a Wikisource. Taktiež podporuje aj mnoho servisov mimo Wikimedia projekty. Jej obsah je dostupný cez bezplatnú licenciu.[10]

Dátové modely Wikidata sú usporiadané do stránok a takto sú aj štruktúrované jednotlivé údaje. Každý subjekt, o ktorom má Wikidata štruktúrované údaje sa nazýva entita a každá entita ma svoju vlastnú stránku V súčasnosti sa rozlišujú tri hlavne typy entít a síce položky, vlastnosti a lexémy.

1.1.6.1 Položky, vlastnosti a lexémy

Položky predstavujú triedy, z ktorých má každá svoj štítok, popis a ľubovoľný počet aliasov. Štítky položiek nemusia byť jedinečné, vzhľadom na to, že môžu existovať rovnaké názvy, ktoré však popisujú rozdielne veci. Kombinácia štítku a jeho popisu však musí byť jedinečná. Aby sa zabránilo nejasnostiam, je s touto kombináciou spojený jedinečný identifikátor položky , ktorý začína písmenom Q, za ktorým nasleduje číslo (napríklad Q42). Položky teda vždy pozostávajú z povinného identifikátora a voliteľného počtu aliasov.[10]

Výroky podrobne opisujú charakteristiky jednotlivých položiek a skladajú sa z vlastnosti (pripomínajú vlastnosti RDF) a hodnôt. Jeden výrok môže byť mapovaný na viac ako jednu hodnotu. Napríklad vlastnosť "vzdelanie" nejakej osoby môže byť spojená s viacerými hodnotami, pretože osoba môže alebo mohla študovať na viacerých univerzitách. Vlastnosti vo Wikidata majú označenie P, za ktorým nasleduje číslo (napríklad P69).[10]

Lexéma je lexikálny prvok jazyka (ako napríklad slovo, fráza alebo predpona). Lexému je možné opísať pomocou nasledujúcich informácií:[11]

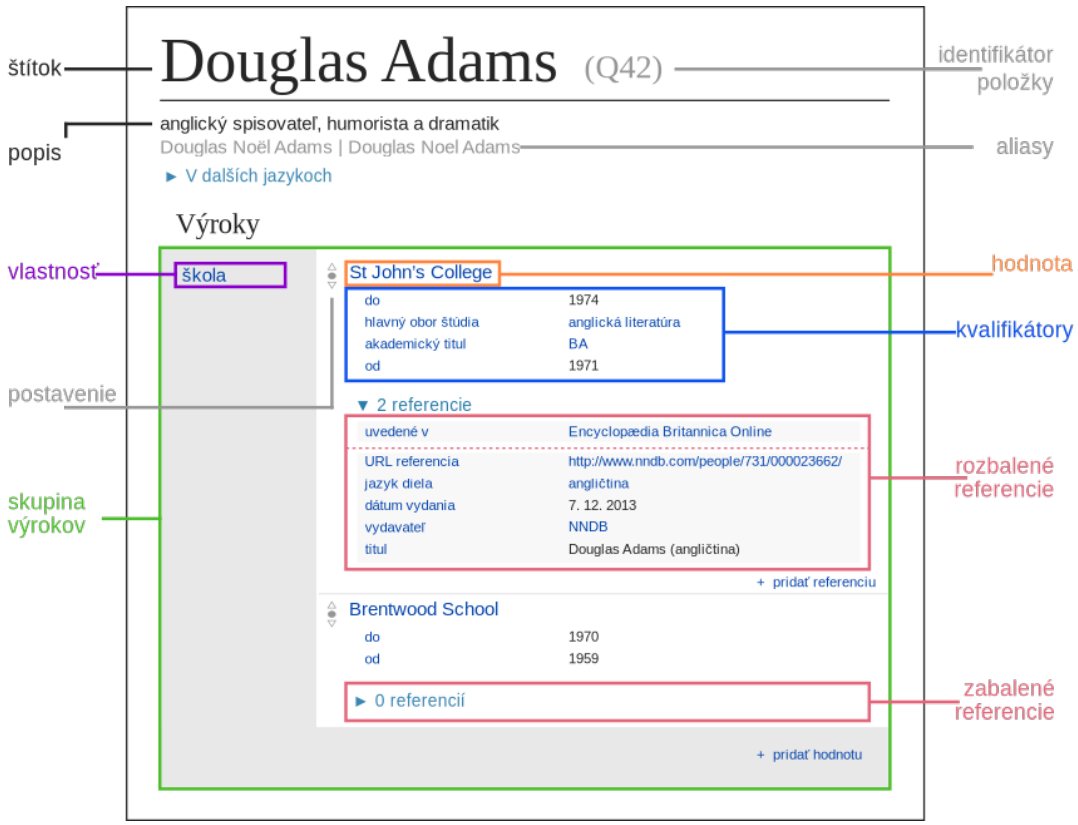
- Identifikačné číslo začínajúce písmenom L, za ktorým nasleduje číslo.
- Lema, človekom zrozumiteľná podoba lexémy (napríklad behať).
- Jazyk, do ktorého lexéma patrí.
- Lexikálna kategória, do ktorej lexéma patrí (napríklad prídavné meno).
- Zoznam výrokov, ktoré popisujú vlastnosti lexémy.
- Zoznam tvarov (forms), obvykle jedna pre každú gramatickú vlastnosť (napríklad 2. osoba/ jednotné číslo/ minulý čas ...).
- Zoznam významov, ktoré popisujú rôzne významy lexémy (napríklad list - "časť rastliny" alebo "písomná správa").

Položka	**Vlastnosť**	**Hodnota**
Q42	P69	Q691283
Douglas Adams	educated at (škola)	St John's College

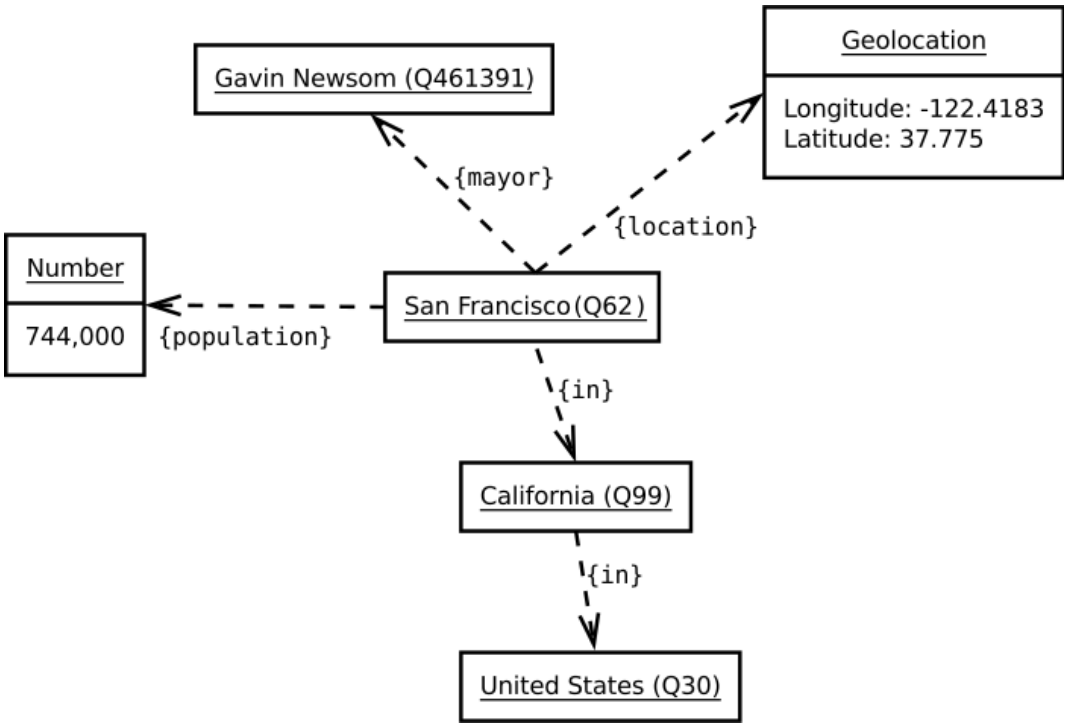
Tabuľka 1: Príklad hlavných pojmov.

Ako už bolo spomenuté vyššie, osobám je možné pridať napríklad vlastnosť určujúcu kde študovali, pričom hodnota bude konkrétna škola. Pri budovách je možné napríklad pridať geografické súradnice.

Vlastnosti môžu tiež odkazovať na externé databázy. Vlastnosti, ktoré spájajú položky s externou databázou, ako napríklad databázy autorov spravované knižnicami a archívmi, sa nazývajú identifikátory.



Obr. 1: Najdôležitejšie pojmy používané vo Wikidata.
[12]



Obr. 2: Vzájomné prepojenie položiek a ich údajov. [13]

1.1.6.2 Údaje

Údaj o položke je zaznamenaný na jeho stránke. Údaj sa skladá z claim-ov, doplnených o referencie (udávajúce zdroj claim-u) a poradie (používané na rozlíšenie medzi niekoľkými claim-ami obsahujúcimi rovnaké vlastnosti, štandardne „normal“). Wikidata neprijíma žiadne predpoklady o správnosti tvrdení, iba ich zhromažďuje a nahlasuje s odkazom na zdroj. Údaj sa často zamieňa s claim-om, ale technicky sa z neho stane údaj až po pridaní aspoň jedného odkazu.[14]

1.1.6.3 Claims

Wikidata pracuje aj s takzvanými claim-ami. Jedná sa o isté dáta ktoré sa týkajú danej entity. Claim sa skladá z vlastnosti a hodnoty, pričom v špeciálnych prípadoch sa môže jednať o tzv. „no value“ či „unknown value“. Pre ukážku, vlasnosť môže byť lokácia a hodnota Slovensko. Claim taktiež môže mať kvalifikátory, ktoré napríklad hovoria o tom, či je daný claim platný v danom čase. Pokiaľ by sme claim porovnali s trojicami v RDF dátovom modeli, claim používa vlastnosť ako predikát a hodnotu ako objekt tejto trojice. Claim-y tvoria súčasť údajov v položkách, kde môžu byť rozšírené o referencie či ranky.[14]

1.1.6.4 Snaks

Snak je technický výraz softvéru Wikibase, s ktorým sa používatelia údajov s najväčšou pravdepodobnosťou stretnú pri prístupe k Wikidata prostredníctvom rozhrania MediaWiki API. Vzťahuje sa na kombináciu vlastnosti a buď hodnoty, alebo jedného zo zvláštnych prípadov (podobne ako u claims) „žiadna hodnota“ a „neznáma hodnota“. Snaks možno nájsť v claims (potom sa im hovorí „hlavné snaks“) alebo v kvalifikáciách ako súčasť údaju (potom sa im hovorí „kvalifikačné snaks“). Napr. Vo vyhlásení „Emma Watson bola členkou obsadenia filmu Harry Potter a Kameň mudrcov v úlohe Hermiony Grangerovej“ existuje hlavný snak „bola členkou obsadenia filmu Harry Potter a Kameň mudrcov“ a kvalifikačný snak „v úlohe Hermiony Grangerovej“. [14]

1.1.6.5 Dátové typy

Wikidata taktiež používa dátové typy, ktoré určujú ako sa údaje budú správať. Rôzne typy údajov majú rôzne vlastnosti a taktiež rôzne dátové typy. Tieto dátové typy nie sú však primitívne v zmysle, že uchovávajú jednu hodnotu typu ako napr. číslo. Často sú to hodnoty so špeciálnou internou štruktúrou ako napr. zemepisné súradnice, ktoré špecifikujú zemepisnú šírku, dĺžku a môžu obsahovať aj výšku.[15][16]

Medzi najčastejšie používané dátové typy patria:

- **Commons media** - Referencie na súbory z Wikimedia Commons. Tieto súbory môžu byť použité na ilustráciu konceptov z Wikidata alebo aj ako súčasť vlastnosti.[15]
- **Item** - Interný link na danú položku, ktorý odkazuje na ozajstný objekt, koncept alebo udalosť s daným identifikátorom.[15][14]
- **Property** - Interný link na vlastnosť, ktorá špecifikuje hodnotu udalosti. Môžeme hovoriť o kategorizácii dát, napr. farba pre hodnotu červená.[15][17]

Týchto dátových typov obsahuje Wikidata mnoho, možno ich nájsť v zdroji[15].

1.1.6.6 Rank

Rank je kvalifikačný faktor, ktorý sa používa na filtrovanie v prípadoch, že existuje viacero údajov pre danú vlastnosť. Prakticky povedané, v takýchto prípadoch treba rozlíšiť dôležitosť údajov. Uvažujú sa 3 možnosti, a to „normal“, „preferred“ a „deprecated“. „Deprecated“ sa používa, pokiaľ je daný údaj podporovaný referenciou, avšak sa považuje

za nesprávny. „Normal“ rank je predvolený pre všetky údaje. Neposkytuje žiaden názor na presnosť hodnoty, čiže je neutrálny. „Preferred“ rank je priradený najviac aktuálnemu údaju ktorý najlepšie reprezentuje konsenzus.[14]

1.2 Databázová časť

V tejto sekcii a jej ďalších podsekciách prezentujeme popis a analýzu problematiky z pohľadu databázovej časti projektu, vypracovanú naším databázovým tímom. Venujeme sa témam ako Protégé, či UCO2, ale aj všeobecnej analýze a jednotlivým systémom a modelom databáz známych zraniteľností.

1.2.1 Protégé

Protégé je open-source nástroj na tvorbu ontológií, vyvíjaný najmä na Stanfordskej univerzite a dostupný pod BSD-2 licenciou. [18] V súčasnej verzií plne podporuje OWL 2 a RDF jazyky. Protégé poskytuje grafické rozhranie na tvorbu a upravovanie ontológií. Nakoľko je to nástroj naprogramovaný v Jave, poskytuje širokú škálu možností rozširovania pomocou pluginov. [19]

1.2.2 Analýza z pohľadu databázového tímu

Aby sme dokázali vytvoriť ontológiu, ktorá bude zoskupovať ontológie z oblasti bezpečnosti, museli sme si urobiť prieskum, ako ostatné ontológie definujú pojmy a ako by sme ich potenciálne mohli zoskupovať. Niekoľko ontológii z hľadiska kybernetickej bezpečnosti zachytáva napríklad projekt UCO2, ktorý zoskupuje všeobecne známe ontológie a databázy zraniteľnosti ako CVE, CAPEC, CCE a podobne. [20]

Analýzou sme zistili, že každá z ontológií definuje pojmy rôzne a preto budeme musieť urobiť hlbšiu analýzu, aby sme čo najviac dát mohli namapovať na nami neskôr vytvorenú ontológiu s čo najmenšími údajovými stratami.

1.2.3 UCO2

Unified Cybersecurity Ontology (UCO) bola vytvorená ako úsilie o pomoc pri vývoji štandardov kybernetickej bezpečnosti od syntaktickej reprezentácie k sémantickejšej reprezentácii. Vidíme niekoľko príspevkov, ktoré UCO ponúka:

- 1. UCO ontológia poskytuje spoločné chápanie domény kybernetickej bezpečnosti a zjednocuje najbežnejšie používané štandardy kybernetickej bezpečnosti.
- 2. V porovnaní s existujúcimi ontológiami kybernetickej bezpečnosti, ktoré sa vyvinuli nezávisle, sa UCO mapovalo na množstvo existujúcich verejne dostupných ontológií kybernetickej bezpečnosti s cieľom podporiť zdieľanie, integráciu a opä-

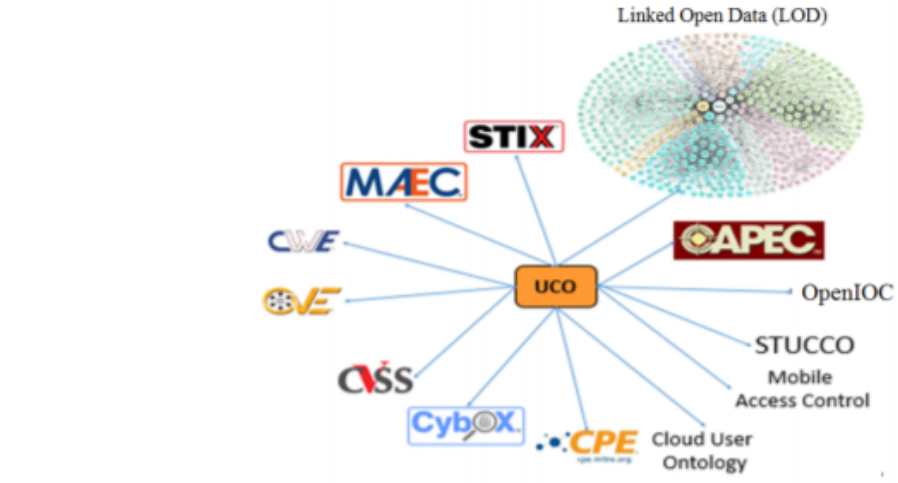
tovné použitie ontológií. UCO slúži ako chrbtová kosť na prepojenie ontológií v oblasti kybernetickej bezpečnosti.

- 3. UCO mapuje koncepty do všeobecných svetových zdrojov znalostí tj. prepojený otvorený dátový cloud na podporu rôznych prípadov použitia.
- 4. Opisuje dôležité prípady použitia, ktoré je možné podporiť zjednotením údajov o kybernetickej bezpečnosti s existujúcimi všeobecnými znalosťami sveta prostredníctvom ontológie UCO.[20]

UCO ontológia poskytuje spoločné chápanie domény kybernetickej bezpečnosti. Spomedzi všetkých štandardov a formátov kybernetickej bezpečnosti je STIX (Structured Threat Information eXpression) (Barnum 2012) najkomplexnejším úsilím o zjednotenie zdieľania informácií o kybernetickej bezpečnosti a umožňuje rozšírenia začlenením slovnej zásoby z niekoľkých ďalších štandardov. Avšak v STIX sú informácie interpretované v XML, a preto nemôžu podporovať uvažovanie, ktoré podporuje UCO. UCO bolo vytvorené ako sémantická verzia systému STIX. Okrem mapovania na STIX bolo UCO rozšírené aj o množstvo príslušných štandardov, slovníkov a ontológií kybernetickej bezpečnosti, ako sú CVE4, CCE5, CVSS6, CAPEC7, CYBOX8, KillChain9 a STUCCO10. Na podporu rôznych prípadov použitia bola ontológia UCO namapovaná na všeobecné svetové znalosti dostupné prostredníctvom znalostného grafu spoločnosti Google, vedomostnej základne DBpedia (Auer et al. 2007), vedomostnej základne Yago (Suchanek, Kasneci a Weikum 2008) atď. Prepojenie s týmito poznatkami sources poskytuje prístup k veľkému množstvu súborov údajov pre rôzne domény (napr. geografické názvy), ako aj výrazom v rôznych jazykoch (napr. ruština).[20]

Zoznam dôležitých tried prítomných v ontológii UCO:[20]

- **Prostriedky** - Táto trieda popisuje rôzne metódy vykonávania útoku a skladá sa z podtried ako BufferOverFlow, SynFlood, LogicExploit, Tcp-PortScan atď., Ktoré sa môžu ďalej skladať z ich vlastných podtried. Trieda Means mapuje na pole TTP v STIXe, ktoré charakterizuje konkrétne podrobnosti pozorovaných alebo potenciálnych taktík, techník a postupov útočníka.
- **Dôsledky** - Táto trieda popisuje možné výsledky útoku. Skladá sa z podtried ako DenialOfService, LossOfConfiguration, PrivilegeEscalation, UnauthUser atď. Mapuje sa na pozorovateľné súbory v systéme STIX.
- **Útok** - Táto trieda charakterizuje útok kybernetickej hrozby a je mapovaný na Incident v STIX.



Obr. 3: Rozšírenia UCO

- **Útočník** - Táto trieda predstavuje identifikáciu alebo charakterizáciu protivníka a je mapovaná na ThreatActor v systéme STIX.
- **AttackPattern** - Attack Patterns sú popisy bežných metód využívania softvéru, ktoré útočníkom poskytujú perspektívu a rady týkajúce sa spôsobov, ako zmierniť ich účinok. Príkladom útočného modelu je phishing.
- **Exploit** - Táto trieda charakterizuje popis jednotlivých exploitov a mapuje ich na ExploitType v schéme STIX.
- **Exploit Target** - Exploit Target sú zraniteľnosti alebo slabosti softvéru, systémov, sietí alebo konfigurácií, ktoré sú zamerané na zneužitie TTP (taktika, technika alebo postup).
- **Indikátor** - Indikátor kybernetickej hrozby je tvorený vzorom identifikujúcim určité pozorovateľné podmienky, ako aj kontextovými informáciami o význame vzorov, o tom, ako a kedy by mali byť ovplyvnené, atď. Táto trieda je mapovaná na IndicatorType v schéme a indikátore STIX triedy v CAPEC ontológii

1.2.4 Systémy databáz zraniteľností

V tejto sekcii si bližšie popíšeme niektoré nami vybrané slovníky, systémy alebo modely databáz zraniteľností, s ktorými budeme chcieť pracovať. Hlavnou myšlienkou je teda najmä kompatibilita nami vytváraného systému s týmito modelmi.

1.2.4.1 Popisovače a rámce vysokej úrovne

Tieto štandardy v sebe kombinujú viacero typov informácií, s ktorými sú schopné pracovať a popisovať ich. Medzi tieto typy patria napríklad aj indikátory includovania (nevyžiadané zahrňovanie), ovplyvnené aktíva, vykonané akcie, či ďalšie kontextové informácie.

1.2.4.1.1 STIX

STIX (z angl. „Structured Threat Information eXpression“, t.j. Vyjadrenie štruktúrovaných informácií o hrozbách) je komunitný koncept založený na spolupráci za cieľom definovania a vývoja štandardizovaného jazyka, ktorý predstavuje štruktúrované informácie o kybernetických hrozbách. Jazyk STIX mieni sprostredkovať celú škálu potenciálnych informácií o kybernetických hrozbách a snaží sa byť plne expresívny, flexibilný, rozšíriteľný, automatizovateľný a v neposlednom rade čo najlepšie čitateľný pre človeka. STIX poskytuje zjednocujúcu architektúru spájajúcu rozmanitú sadu informácií o kybernetických hrozbách a to vrátane:

- **Kyberneticky pozorovateľných udalostí**
- **Indikátorov**
- **Incidentov**
- **Nepriaznivých taktík, techník a postupov** (vrátane útočných schém, malvérov, exploitov, kill chainov, nástrojov, infraštruktúry a zacieľovania sa na obeť)
- **Cieľov exploitovania** (zraniteľnosti, slabé stránky alebo konfigurácie)
- **Plánov postupov** (reakcie na incidenty, náprava alebo zmiernenie zraniteľností/slabostí)
- **Kampaní kybernetických útokov**
- **Aktérov kybernetických hrozieb**

Čo sa týka dohliadajúcej organizácie, STIX patrí pod OASIS (pred tým MITRE a DHS). [21]

1.2.4.2 Pozorovateľné udalosti vyžadujúce akciu

Štandardy kyberneticky pozorovateľných udalostí reprezentujú informácie používané na detekciu útokov alebo škodlivých aktivít (napríklad využívanie systémových knižníc pomocou malvéra). Kyberneticky pozorovateľnou udalosťou sa v tomto prípade myslí akákoľvek merateľná udalosť alebo stavová vlastnosť v kybernetickom kontexte. Príkladom takýchto merateľných udalostí môže byť vytvorenie kľúča registra, odstránenie súboru alebo odoslanie HTTP GET požiadavky. Medzi stavové vlastnosti patrí napríklad MD5 hash súborov, hodnota kľúča registra alebo názvy procesov.

1.2.4.2.1 MAEC

MAEC (z angl. „Malware Attribute Enumeration and Characterization“, t.j. Zoznam a charakteristika atribútov malvéru) je štandardizovaný jazyk na kódovanie a komunikáciu s dôvernými informáciami o malvéri, na základe atribútov ako sú správanie, artefakty a útočné schémy. Elimináciou nejednoznačnosti a nepresnosti, ktorá v súčasnosti pri popisovaní malvérov existuje, si MAEC kladie za cieľ zlepšenie komunikácie ohľadom malvérov na rozhraniach človek-človek, človek-nástroj, nástroj-nástroj a nástroj-človek, čo by malo za výsledok zníženie potenciálnej duplicity analyzovania tých istých malvérov vedcami a pomáhanie rýchlejšiemu vývoju protiopatrení prostredníctvom možnosti využívať reakcie na už známe pozorované inštancie malvérov. Pre tento účel využíva MAEC aj STIX prostredníctvom TPP konštrukcií a navyše aj pre STIX aj MAEC platí, že využívajú CybOX.[21]

1.2.4.3 Zoznamy a slovníky

Zoznamy (enumerácie) definujú globálne identifikátory, ktoré odkazujú na zdieľané dátové objekty. Ako príklad sa dá použiť CVE.

1.2.4.3.1 CVE

CVE (z angl. „Common Vulnerabilities and Exposures“, t.j. Všeobecné chyby a expozície) je slovník verejne známych slabých miest a ohrození bezpečnosti informácii. Bežné CVE identifikátory umožňujú výmenu údajov medzi bezpečnostnými produktmi a poskytujú základný bod indexu pre vyhodnocovanie pokrytia bezpečnosti zo strany nástrojov a služieb. Dohliadajúce organizácie sú MITRE a DHS.[22]

1.2.4.3.2 CCE

CCE (z angl. „Common Configuration Enumeration“, t.j. Zoznam všeobecných konfigurácií) priradzuje jedinečné položky/vstupy (tiež zvané CCE položky) k príkazom a odporúčaniam na konfiguráciu a ovládacím prvkom konfigurácie, za cieľom zlepšiť pracovný tok zastrešením rýchlej a presnej korelácie konfiguračných problémov, ktoré sa vyskytujú v nezlučiteľných doménach. V tomto zmysle je podobný iným porovnateľným dátovým štandardom ako napríklad spomínaný CVE zoznam, ktorý prideľuje identifikátory k verejne známym zraniteľnostiam systémov. Dohliadajúca organizácia je NIST (pred tým MITRE).[23]

1.2.4.3.3 CAPEC

CAPEC (z angl. „Common Attack Pattern Enumeration and Classification“, t.j. Zoznam a klasifikácia všeobecných útočných schém) je verejne dostupný, komunitne vyvíjaný zoznam bežných (známych) útočných schém a vzorov spolu s komplexnou schémou a klasifikačnou taxonómiou. Útočné schémy sú popisy známych metód exploitovania softvérových systémov. Vychádzajú z konceptu dizajnových vzorov aplikovateľných skôr v deštruktívnom, než konštruktívnom kontexte a sú generované z hĺbkovej analýzy konkrétnych príkladov exploitovania z reálneho sveta. STIX dokáže využívať CAPEC na štruktúrovanú charakterizáciu taktických, technických a procedurálnych útočných schém, a to pomocou rozšírenia CAPEC Schema Extension. Dohliadajúce organizácie sú MITRE a DHS.[21]

1.2.4.3.4 CWE

CWE (z angl. „Common Weakness Enumeration“, t.j. Zoznam všeobecných slabín) poskytuje jednotnú a merateľnú sadu softvérových slabín, ktoré umožňujú účinnejšiu diskusiu, popis, výber a použitie softvérových bezpečnostných nástrojov a služieb, ktoré dokážu nájsť tieto slabosti v zdrojovom kóde a operačných systémoch. Taktiež poskytuje lepšie porozumenie a manažment pri softvérových slabinách súvisiacich s architektúrou a dizajnom. Dohliadajúce organizácie sú MITRE a DHS.[21]

1.2.4.3.5 CPE

CPE (z angl. „Common Platform Enumeration“, t.j. Zoznam všeobecných platforiem) je štandardizovaná metóda opisu a identifikácie tried aplikácií, operačných systémov a hardvérových zariadení, ktoré sú prítomné v rámci výpočtových prostriedkov spoločnosti. CPE neidentifikuje jedinečné inštancie produktov v systémoch, ako napríklad inštaláciu XYZ

Visualizer Enterprise Suite 4.2.3 s nejakým konkrétnym výrobným číslom Q472B987P113, ale identifikuje abstraktné triedy produktov, čiže napríklad XYZ Visualizer (vo všetkých verziách). Dohliadajúca organizácia je NIST (pred tým MITRE).[23]

1.2.5 Doplnkové ontológie pre CVE zraniteľnosti a OVAL

Paralelne s týmto projektom bol vyvíjaný nový ontologický model, ktorý mal za cieľ obohatiť bezpečnostnú oblasť o novú ontológiu. Matej Rychtárik analyzoval vo svojej diplomovej práci nedostatky v bezpečnostných ontológiách a navrhol doplnkový model, ktorý vypĺňal práve medzery v UCO ontológii. Novonavrhnutý model kombinujeOVAL popisy s CVE zraniteľnosťami, čím sú popísané dodatočné časti, ktoré sú v rámci UCO ontológií zatiaľ nepopísané. Zraniteľnosti v CVE modeli majú svoj identifikátor, popis, názov a taktiež referencie na dodatočné zdroje, ktoré pomáhajú popisovať danú zraniteľnosť. Ontológia pre OVAL neobsahuje plnú OVAL špecifikáciu. Autor sa zameral iba na malú časť a do ontológie zahrnul všeobecné informácie popisujúce a definujúce OVAL záznamy, ich vzťahy s jednotlivými operačnými systémami, ich verziami a produktami. Tieto ontologické modely a súvisiace dáta s nimi boli použité v rámci tohto projektu.[24]

1.3 Backendová časť

V tejto sekcii a jej ďalších podsekciách prezentujeme popis a analýzu problematiky z pohľadu backend-ovej časti projektu, vypracovanú naším backend-ovým tímom. Venujeme sa témam ako Blazegraph, Blueprints, či Sesame API.

1.3.1 Blazegraph

Blazegraph je štandardná, vysoko výkonná, škálovateľná, open-source grafová databáza. Je napísaná v prostredí Java, platforma podporuje Blueprints a RDF / SPARQL 1.1 špecifikácie vrátane Query, Update, Basic Federated Query a Service Description. Aplikácia Blazegraph podporuje nové rozšírenia pre trvácne pomenované sady riešení, efektívne ukladanie a dopytovanie modelov a škálovateľnú grafickú analytiku. Databáza podporuje multi-tenancy a je možné ju nasadiť ako zabudovanú databázu, samostatný server, vysoko dostupný replikačný klaster a ako horizontálne rozdelené združenie služieb podobné ako Google bigtable, Apache Accumulo alebo Cassandra.[25]

Grafy sú výkonným a flexibilným prostriedkom na reprezentáciu všetkých druhov prepojených údajov. RDF je dátový model, ktorý poskytuje štandardný spôsob popisu, výmeny a dopytovania údajov grafu. RDF modeluje graf ako zbierku RDF príkazov. RDF príkazy, ktoré vám umožňujú pridať vrcholy, hrany a vlastnosti vrcholov do vášho grafu. RDR je jednoduché rozšírenie RDF, ktoré umožňuje pridať aj vlastnosti vrcholov.

Blazegraph je plne otvorená vysoko výkonná databáza grafov podporujúca dátový model RDF a RDR. Funguje ako zabudovaná databáza alebo cez rozhranie REST API typu klient/server. Blazegraph podporuje vysokú dostupnosť a dynamické zdieľanie, ako aj rozhrania Blueprints a Sesame API.[25]

1.3.2 Blueprints

Blueprints je kolekcia rozhraní, implementácií, duplikácií a testovacích balíkov pre dátový model grafu. Blueprints sú analogické s JDBC, ale pre grafové databázy. Ako taký poskytuje spoločnú sadu rozhraní, ktorá umožňuje vývojárom pripojiť a spustiť ich backend-ové grafové databázy. Softvér napísaný na vrchole Blueprints navyše funguje vo všetkých grafových databázach s povoleným Blueprints. V softvérovom balíku TinkerPop slúži Blueprints ako základná technológia pre:[25]

- **Pipes** - framework dátového toku
- **Gremlin** - prechodný jazyk
- **Frames** - mapper objektov do grafov
- **Furnance** - súbor grafových algoritmov
- **Rexster** - grafový server

1.3.3 Sesame API

Blazegraph implementuje Sesame API Storage and Inference Layer (SAIL), ktorá slúži na vytvorenie repozitára a otvorenie komunikácie s databázou Blazegraph. Model API poskytuje kolekciu objektov Java, ktoré predstavujú Resource Definition Framework (RDF) koncepty. Blazegraph implementácie sa dajú nájsť v balíku `com.bigdata.rdf.model`. Rozhranie Value sa používa na vyjadrenie generickej entity RDF, ktorá môže byť buď Resource alebo Literal:

- URI ako pomenovaný Resource
- Blank node ako nepomenovaný Resource
- Literál ako špecifický typ Value, ktorý predstavuje jeden zo základných typov údajov
- Statement je usporiadaná trojica: subjekt (Resource), predikát (URI) a objekt (Value)

Dáta sa dajú pridávať ako súbory RDF formátu: RDF/XML, Notation3, Turtle, Turtle-RDR, N-Triples-RDR, N-Triples, N-Quads, JSON, TriG, TriX a zároveň čítať pomocou SPARQL dotazov.[25]

1.4 Frontendová časť

V tejto sekcii a jej ďalších pod-sekciách prezentujeme popis a analýzu problematiky z pohľadu frontend-ovej časti projektu, vypracovanú naším frontend-ovým tímom. Venujeme sa témam ako SQID, či SPARQL.

1.4.1 SQID

SQID je nástroj, ktorý ponúka možnosť rýchleho vyhľadávania a dopytovania informácii z Wikidata. Wikidata je sesterský projekt Wikipédie, ktorý spravuje informácie použité na Wikipédii alebo na iných projektoch Wikimedia. SQID je implementovaný ako prehliadač (5), ktorý zobrazuje údaje získané z Wikidata API alebo pomocou Wikidata SPARQL query service. Okrem iného SQID tiež ponuka prehliadač tried a vlastnosti, ktoré umožňujú zobraziť väčšie a podrobnejšie zoznamy výsledkov zo SPARQL dotazov.[26]

SQID je primárne napojený na 3 endpointy, a to commons endpoint, wikidata endpoint a sparql endpoint. Commons endpoint[1] slúži na prenos obrázkov, audio a video súborov, a podobne. Z wikidata endpointu[2] sa získavajú konkrétne údaje o entitách (ako názov, popis, URL alebo odkazy na súvisiace entity). SPARQL enpoint[3] je napojenie na SPARQL server, z ktorého sa SPARQL dotazmi získavajú dáta, takto získane dáta sa následne prelinkujú s jednotlivými entitami z Wikidata endpointu.[27]

Časti aplikácie sú napísané v rôznych programovacích jazykoch, napríklad Java, Python, Ansible. Na front-ende sa používa Vue.js. Zdrojový kód SQID-u sa nachádza na Githube[4], s manuálom na inštaláciu.[26]

Našim cieľom je využiť tento prehliadač a upraviť ho tak aby vedel pracovať a komunikovať s našou vlastnou databázou informácii. Nevýhodou SQID-u je fakt, že k nemu nie je takmer žiadna technická dokumentáciu a keďže je napísaný vo veľa rôznych jazykoch nie je najjednoduchšie sa v kóde zorientovať.[26]

[1]Pre viac informácií pozri `https://commons.wikimedia.org/w/api.php`
[2]Pre viac informácii pozri `https://www.wikidata.org/w/api.php`
[3]Pre viac informácii pozri `https://query.wikidata.org/bigdata/namespace/wdq/sparql`
[4]https://github.com/Wikidata/SQID

2 Praktická časť

V tejto časti popisujeme praktické riešenie nášho projektu. Zameriavame sa na technické a implementačné detaily, ktoré sa týkali našej backendovej, frontendovej ale aj databázovej časti projektu ako aj na problémy, s ktorými sme sa museli počas tvorby riešenia vysporiadať.

2.1 MediaWiki

V rámci našich pokusov o prepojenie nástroja SQID s našou vlastnou Blazegraph databázou sme nadobudli motiváciu skúsiť do celého riešenia integrovať MediaWiki. Hlavným zdrojom motivácie bol fakt, že Wikidata vychádza pôvodne z projektu MediaWiki. Tým, že by sme skúsili pripojiť najprv vlastnú Wiki ku nástroju SQID, sme chceli získať lepšiu predstavu o tom, ako funguje komunikácia Wikidaty a SQID-u. Všimli sme si, že SQID by potenciálne mohol využívať MediaWiki API[5], čo sa nám neskôr potvrdilo ako falošný predpoklad. Hlavný cieľ, ktorý sme chceli dosiahnuť je prepojiť MediaWiki so SQID-om a následne pripraviť komunikáciu s Blazegraphom skrz MediaWiki.

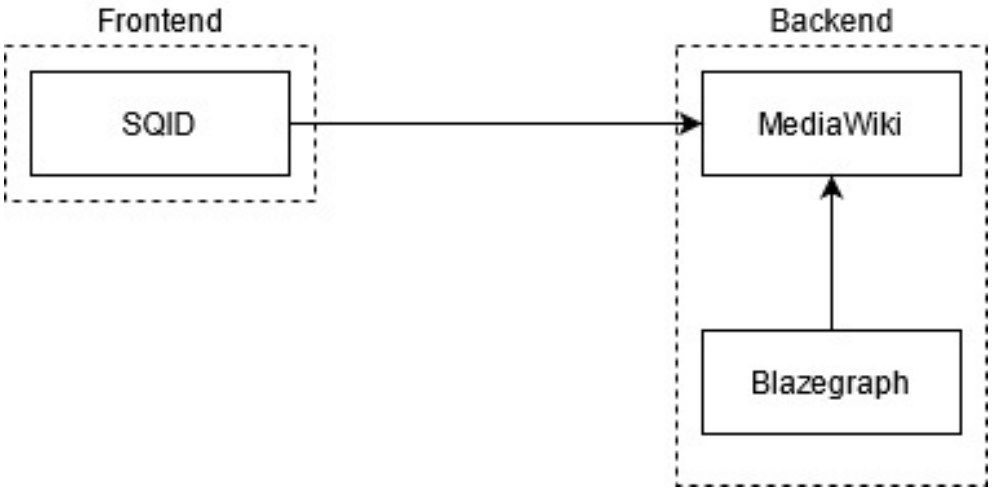
Po úspešnom pripravení a nainštalovaní MediaWiki na virtuálnom stroji sme skúšali prepojiť MediaWiki s lokálnou inštanciou Blazegraph databázy. Samostatná MediaWiki v základe nepodporuje komunikáciu zo žiadným SPARQL úložiskom. Pre tento účel je potrebné nainštalovať doplnok[6], ktorý umožní sémantické vyhľadávanie, čoho dôsledkom je možnosť prepojiť inštanciu podporovaného SPARQL úložiska (čo bol v našom prípade Blazegraph) s MediaWiki. Takýmto spôsobom sme rozšírili našu infraštruktúru o MediaWiki, čím sme docielili nasledovnú architektúru.

Na Obrázku 4 je možné si všimnúť orientovanie šípky medzi MediaWiki a Blazegraphom. Po hlbšom študovaní MediaWiki sme zistili, že MediaWiki má vlastnú databázu, v ktorej si ukladá zobrazované údaje. Doplnok MediaWiki, ktorý pridáva možnosť sémantického vyhľadávania pridáva možnosť používať označenia, ktoré Blazegraph dokáže čítať, vykonávať nad nimi SPARQL dotazy a hľadať ku ním ďalšie súvislosti. Nastáva tu však zásadný problém, ktorý istým spôsobom narúša celú filozofiu cieľa, ku ktorému sme sa snažili priblížiť.

Vzťah medzi MediaWiki a Blazegraphom je iba jednosmerný, to znamená, že Blazegraph dokáže čítať a operovať nad zobrazovanými údajmi z MediaWiki, ale neplatí to naopak, tj. MediaWiki nedokáže zobrazovať ľubovoľné údaje, ktoré sa nachádzajú v

[5]Pre viac informácii pozri `https://github.com/addwiki/mediawiki-api`

[6]Pre viac informácií pozri `https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki`



Obr. 4: Architektúra po integrovaní MediaWiki do infraštruktúry

Blazegraph úložisku.

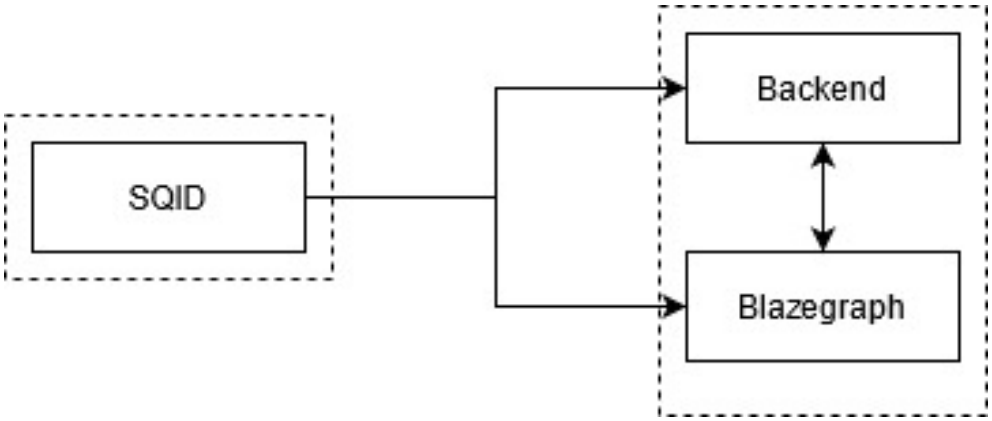
Toto predstavuje určitý problém, pretože takéto riešenie nenapĺňa náš cieľ, ktorý sme sa snažili dosiahnúť. Ako sme písali na úvod, prijateľným riešením by bolo keby sme vedeli využívať MediaWiki API zo SQID-u a pristupovať k údajom, ktoré sa nachádzajú v Blazegraphe. Riešenie by sa potenciálne dalo použiť, keby sme mali ako hlavného aktora MediaWiki s tým, že by sme hľadali ďalšie doplnky, ktoré by umožňovali MediaWiki efektívne pracovať s ontológiami.

Mať MediaWiki ako hlavného aktora je pre nás nevýhodné, pretože okrem Blazegraph databázy by sme museli v rámci infraštruktúry dbať aj na údržbu Wiki. Taktiež by sme museli skúmať ďalší nástroj a zisťovať, či sa vôbec do nášho riešenia hodí.

Z vyššie uvedených dôvodov sme zmenili perspektívu a z riešenia sme vylúčili MediaWiki. Zistili sme, že Wikidata využíva servis na komunikáciu so SPARQL úložiskom, ale tento servis sa nám nevidel byť komplikovaný. Vďaka skúsenostiam s MediaWiki sme došli k záveru, že nám bude stačiť vlastný backend spoločne s vlastnou inštanciou Blazegraphu a stačí nám komunikáciu zjednodušiť tak, ako je to znázornené na nasledovnom obrázku.

Backend môže pripraviť rovnaké rozhranie aké využíva Wikidata. V prípade potreby môže backend priamo operovať nad Blazegraphom a spracovávať pokročilejšie požiadavky, ktorých logika by nemala byť implementovaná v SQID-e. Zároveň SQID môže využívať Blazegraph v prípade, že bude potrebovať vyhľadávať súvislosti s entitami s pomocou logických inferencií (na ktoré by použil SPARQL dotazy).

Tento experiment posunul tímový projekt dopredu vo viacerých ohľadoch. Vďaka tomuto experimentu sme zistili, že viac sa treba orientovať na to, ako pracuje backend,



Obr. 5: Návrh novej architektúry

ktorý Wikidata využíva, než skúmať samotné správanie Wikidaty. Zistili sme, že MediaWiki môže naše riešenie skomplikovať. V neposlednom rade sme dospeli k novému návrhu architektúry.

2.2 SQID

Predtým, než sa hlbšie ponoríme do toho, ako sme so SQID-om experimentovali je vhodné uviesť, aké boli naše ciele, ktoré sme sa snažili dosiahnúť a prečo sme si zvolili práve SQID. SQID je projekt, ktorý vznikol práve kvôli Wikidata. Jeho cieľom bolo efektívnejšie zobrazovať vzťahy a súvislosti medzi entitami, pričom ponúka samozrejme širšiu škálu funkcionalít [28]. Načo sa my chceme zamerať je fakt, že je to projekt, ktorý pracuje s obrovským projektom ako je Wikidata. Je vhodné si uvedomiť, že Wikidata sama o sebe je veľmi komplexný projekt, ktorý naozaj veľmi zaujímavým spôsobom zbiera a centralizuje údaje. Práve toto je dôvod, kvôli čomu sme chceli preskúmať už osvedčené existujúce riešenie, pretože ponúka množstvo inšpirácie, akým spôsobom je možné dosiahnuť naše ciele.

Študovanie SQID-u nebolo triviálnou záležitosťou, nakoľko SQID neobsahuje žiadnu dokumentáciu, s čím sme sa museli popasovať. Napriek tomu musíme priznať, že kód SQID-u je naozaj inšpiratívne a čisto napísaný a dá sa relatívne rýchlo porozumieť tomu, čo kód SQID-u vykonáva. Problém sme skôr mali s tým, že je to naozaj komplexný projekt a teda kódu, ktorý bolo potrebné študovať bolo mnoho, pričom si zároveň bolo treba vyrábať v hlave myšlienkovú mapu, ako sú jednotlivé moduly poprepájané a ako spolu navzájom interagujú. Nakoľko analýza projektu a kódu bola dosť podstatnou časťou celého procesu, jej výsledky sa pokúsime zdokumentovať čo najpodrobnejšie.

2.2.1 Architektúra

Zo zistení analýzy by sme chceli popísať architektúru z dvoch hľadísk. Prvotný popis by sme chceli venovať globálnej architektúre, teda z akých logických celkov pozostáva súborová štruktúra SQID-u a následne sa lepšie pozrieť, čo jednotlivé celky v tejto štruktúre reprezentujú. Popis z druhého hľadiska by mal obsahovať najmä ako je členený zdrojový kód a ktoré moduly sú pre nás podstatné.

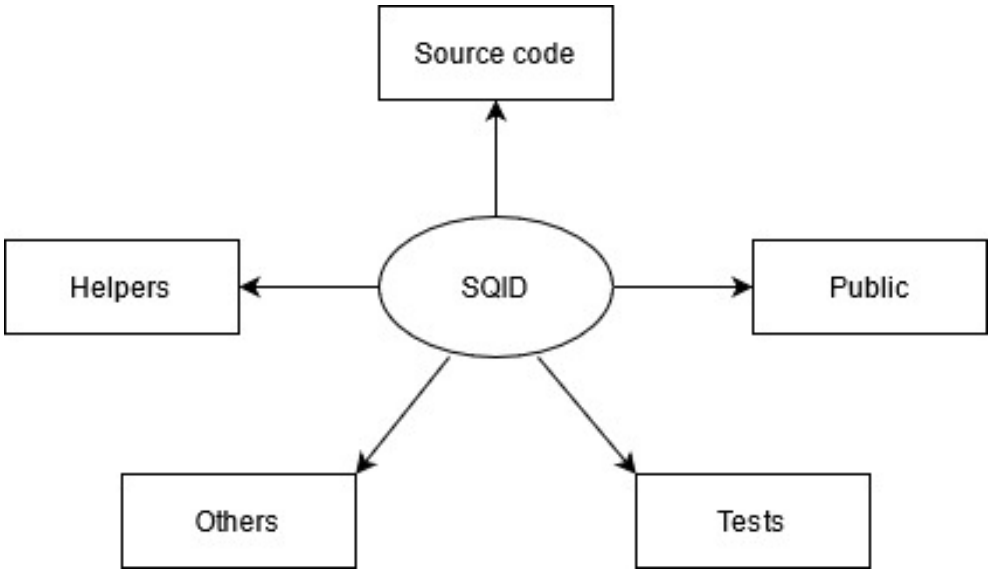
Globálnu architektúru sme sa pokúsili zobraziť na Obrázku 6. Architektúra teda pozostáva z nasledovných logických celkov:

- **Helpers** - Obsahuje súbory, ktoré by sa dali charakterizovať ako doplňujúce. Pod doplňujúcimi súbormi je možné si predstaviť konfiguračné súbory ako napríklad pre preloženie a nasadenie projektu. Pre konkrétnejšiu predstavu sa jedná o skupinu .yml súborov, pom.xml a pod.
- **Public** - Predstavuje súbory, ktoré by mali popisovať projekt pre verejnosť, resp. entity, ktoré by potenciálne mohli naraziť na nasadený SQID projekt na internete. Konkrétne sú tam definované prijateľné spôsoby autorizácie a autentizácie.
- **Source code** - (alebo v preklade zdrojový kód) bola najpodstatnejšou časťou a túto časť si rozoberieme bližšie neskôr.
- **Tests** - Priečinok obsahuje niekoľko testov, ktorými sme sa nezaoberali bližšie.
- **Others** - Nemá v SQID-e samostatný priečinok, ale sú to súbory ktoré neboli nikde zaradené. Jedná sa o všeobecné súbory ako napríklad .gitignore, nastavenie environmentu, nainštalované balíčky (napr. packages.json a packages.lock.json) a pod.

Najrozsiahlejšou časťou je časť so zdrojovým kódom, zároveň táto časť je pre nás aj najpodstatnejšia. Ako sme uviedli v úvode, SQID nedisponuje detailnejšou dokumentáciou, než dokumentáciou na úrovni kódu. Preto je vhodné popísať bližšie jednotlivé hlavné moduly, ktoré sú v SQID-e implementované.

SQID pozostáva z nasledovných častí:

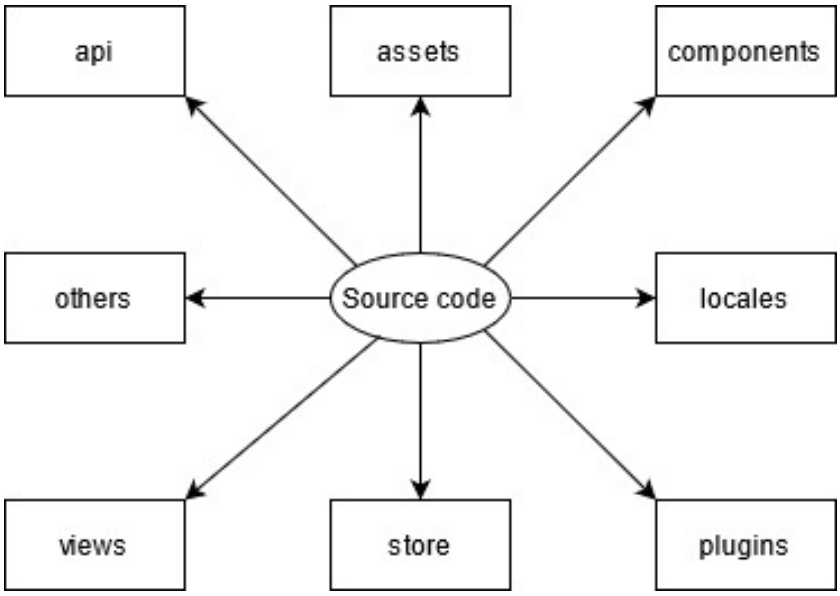
- **Api modul** - Implementuje rozhranie pre požiadavky, ktoré sú posielané na Wikidata. Pod tým je možné predstaviť si napr. definovanie dátových typov, do ktorých možno serializovať odpovede, ktoré prídu z Wikidata API. Taktiež pod to spadá rozhranie pre SQID, aby bolo možné na Wikidata API posielať HTTP a SPARQL požiadavky.



Obr. 6: Architektúra projektu SQID

- **Assets modul** - Obsahuje zobraziteľné fixné assety, ako napríklad logo SQID-u v SVG formáte.
- **Plugins modul** - Importuje do projektu externé pluginy.
- **Components modul** - Obsahuje vue.js zobraziteľné komponenty, ktoré sú prepoužité na viacerých stránkach SQID-u. Príkladom je zdieľané menu, päta stránky, vyhľadávacie pole a pod.
- **Locales modul** - Obsahuje lokalizačné reťazce pre rôzne jazyky, ktoré sa staticky zobrazujú v SQID-e. Sú to teda také reťazce, ktoré sa zobrazia vždy, bez ohľadu na to, či je SQID pripojený na Wikidata API (alebo nejaké iné API). Príkladom sú reťazce, ktoré sa zobrazujú na domovskej stránke SQID-u, či už to je popis stránky alebo jednotlivé položky menu.
- **Store modul** - Zaoberá sa správnym ukladaním dát. Definuje tzv. "getre" a rôzne podobné mechanizmy, aby bolo možné bezpečne meniť stav komponentov a systému.
- **Views** - Obsahuje vue komponenty, ktoré reprezentujú jednotlivé stránky, ktoré je možné zobraziť.
- **Nezaradené súbory** - Nachádzajú sa na vrchu hierarchickej súborovej štruktúry. Medzi nezaradené súbory patria napríklad skripty, s pomocou ktorých sa celý SQID pripravuje pri spustení.

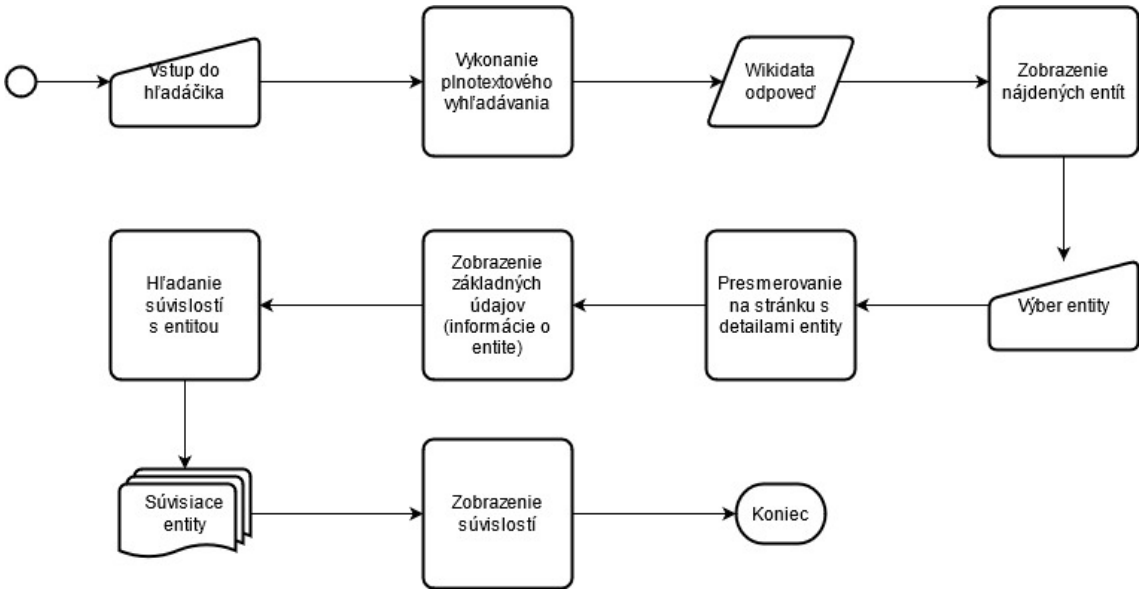
V rámci tejto práce sme väčšinu času strávili v api module, nakoľko sme potrebovali zistiť, ako funguje Wikidata API, tj. aké je správanie endpointov podľa dodaných parametrov, ktoré parametre pre API sú navzájom kompatibilné apod. Grafická reprezentácia spomínaných modulov je zobrazená na Obrázku 7.



Obr. 7: Grafická reprezentácia modulov

2.2.2 Základný workflow

Silnou vlastnosťou SQID-u je vyhľadávanie súvislostí. Výsledkom nášho snaženia má byť prispôsobenie SQID-u na naše účely, teda aby sme dokázali zobrazovať údaje z rôznych ontológií, ktoré sa nachádzajú v našej Blazegraph databáze. Kvôli tomuto bolo potrebné najprv pochopiť na tej najvyššej úrovni, ako SQID pracuje. Pre ľahšie pochopenie pridávame diagram, ktorý zachytáva workflow SQID-u na tej najvyššej úrovni.



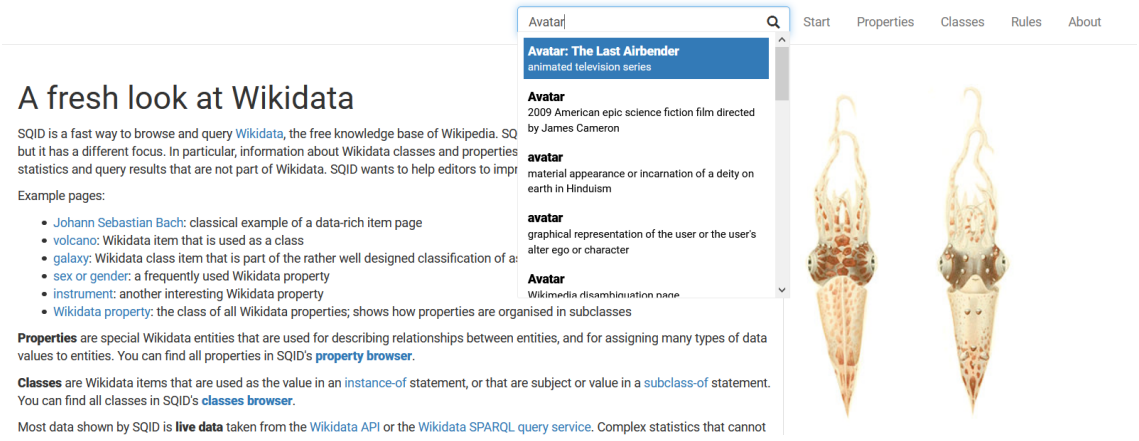
Obr. 8: Prehľad workflowu SQID-u na najvyššej úrovni

Diagram by sme dokázali popísať v nasledovných krokoch:

1. Užívateľ sa nachádza na stránke SQID-u. Užívateľ v SQID-e zadá vstup do vyhľadávacieho poľa (vyhľadávanú entitu napr. Avatar).
2. SQID pošle GET požiadavku na Wikidata, kde sa vykoná plnotextové vyhľadávanie nad entitami v databáze Wikidaty.
3. Wikidata vráti SQID-u odpoveď s entitami, ktoré boli nájdené pri zadanom výraze.
4. SQID zobrazí užívateľovi názvy a popis entít, ktoré vrátila Wikidata.
5. Uživateľ si zo zobrazeného zoznamu vyberie entitu.
6. SQID presmeruje užívateľa na stránku s detailami entity.
7. SQID vykoná dodatočnú požiadavku na Wikidata, v ktorej si vypýta základné informácie o entite. SQID tieto informácie zobrazí.
8. SQID pošle sériu SPARQL dotazov na SPARQL server, v ktorých vyhľadáva súvislosti s aktuálne zobrazenou entitou.
9. SQID dostane odpoveď od SPARQL servera.
10. SQID pošle dodatočné požiadavky na Wikidata API, aby si vypýtal informácie o súvisiacich entitách.
11. SQID dostane od Wikidata odpovede s entitami.
12. SQID zobrazí údaje o entitách uživateľovi.

2.2.3 Analýza plnotextového vyhľadávania

Plnotextové vyhľadávanie jednotlivých entít je možné realizovať pomocou zadania textu do vyhľadávacieho poľa v hornom menu SQID-u. Po poslaní požiadavky na Wikidata API a spracovania odpovede, sa zobrazia možné entity, ktoré sú vystihnuté zadaným textom.



Obr. 9: Príklad plnotextového vyhľadávania

V rámci analýzy sme sa snažili pozerať na 3 aspekty Wikidata endpointu:

- Aké parametre tento endpoint podporuje?
- Nad ktorými vlastnosťami entity je vykonávaná plnotextové vyhľadávanie?
- Ktoré atribúty z odpovede SQID reálne využíva?

Otázku aké parametre podporuje Wikidata endpoint bolo potrebné zodpovedať, aby sme dokázali skonštruovať špecifikáciu endpointu a následne pripraviť náš vlastný endpoint, s ktorým dokážeme nahradiť Wikidata endpoint. Aspekt, ktorý je zahrnutý v druhom bode bolo potrebné skúmať aby sme odpozorovali správanie endpointu a vedeli na základe čoho treba filtrovať entity. Posledný aspekt je kvôli optimalizácií a zníženiu úsilia pri implementácií backendu, tým že vyčleníme atribúty, ktoré SQID reálne ani nepotrebuje.

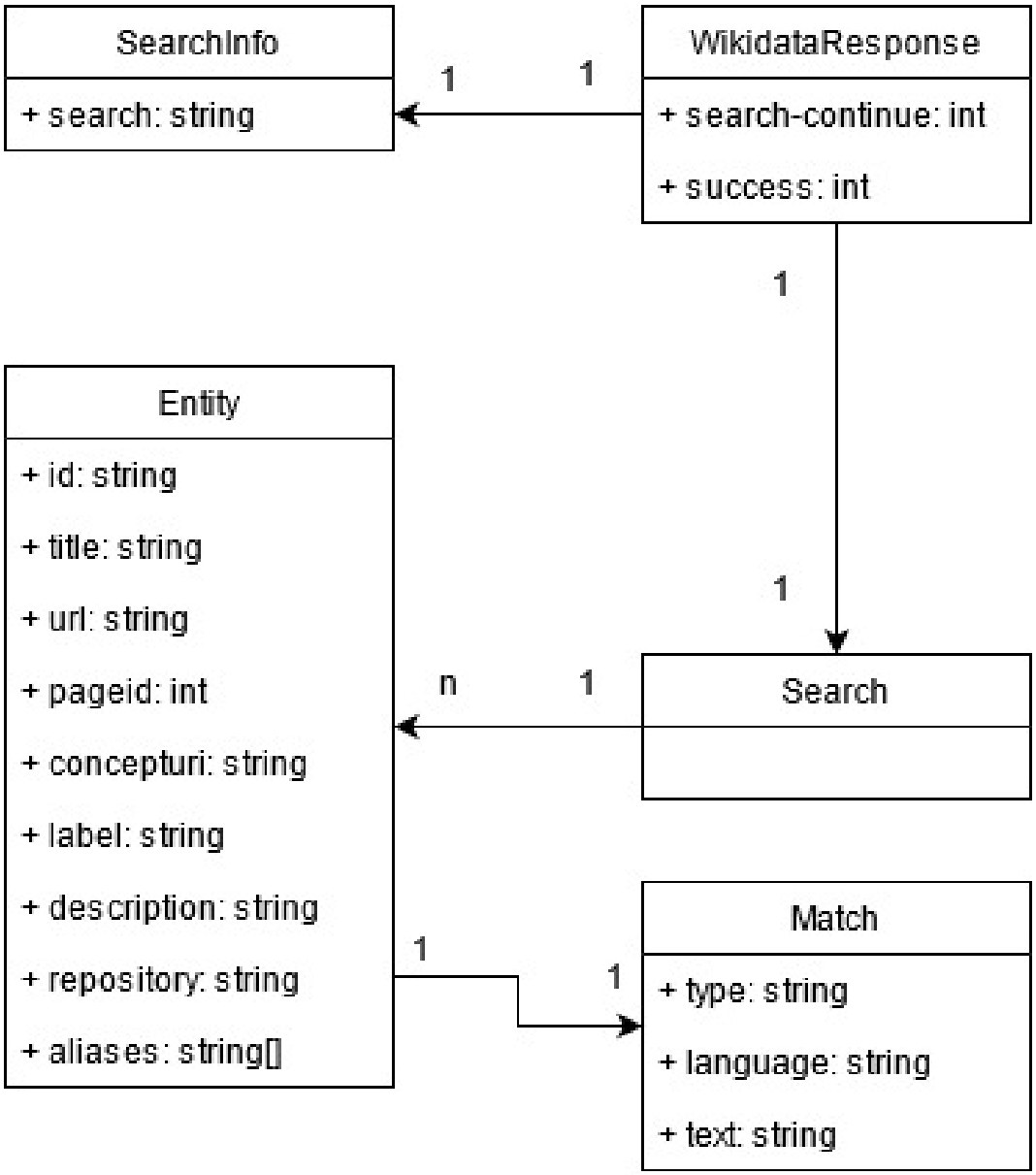
Aspekt zahrnutý v prvom bode sme zisťovali len pozorovaním rôznych volaní, ktoré SQID vykonáva. Endpoint, ktorý SQID využíva na vyhľadávanie entít, zároveň využíva aj na získavanie detailov o entite (detailnejší popis získavania údajov bude súčasťou inej kapitoly), preto bolo potrebné vyfiltrovať iba tie požiadavky, ktoré sa týkajú vyhľadávania entít. Prehľad parametrov je zobrazený v Tabuľke 2.

Názov parametra	**Popis**	**Príklad akceptovanej hodnoty**
Format	Formát odpovede	JSON
Origin	Informácia pre Wikidata, z akej URL sú vyžadované zdroje/informácie (súvisí s CORS hlavičkami)	* (asterisk)
Search	Vyhľadávané kľúčové slovo	Avatar
Language	Jazyk, v ktorom je vrátená odpoveď	en
Action	Akcia, používa sa na rozlíšenie medzi hľadaním entít a získavania údajov o entitách	wbsearchentities
Limit	Maximálny počet záznamov, ktoré sa vrátia v odpovedi	10

Tabuľka 2: Parametre pri plnotextovom vyhľadávaní entít

Endpoint vykonáva plnotextové vyhľadávanie nad špecifickými atribútmi entity. Všetky entity, nakoľko spadajú do jednej ontológie, majú vždy podobnú, ak nie identickú štruktúru údajov. Wikidata vykonáva prehľadávanie nad názvom entity (technicky reprezentované ako štítok) alebo ID entity. Okrem toho je potrebné, aby Wikidata rozpoznaná entita mala správny typ. Ako sme uvádzali v teórií, Wikidata rozlišuje typy entít, teda či sa jedná o typ položka, vlastnosť alebo lexéma. Plnotextové vyhľadávanie v tomto prípade vráti iba entity, ktoré sú typu položka.

Po tom, čo Wikidata dostane požiadavku, vráti odpoveď. Odpoveď je totižto štandardizovaná a nie všetky atribúty, ktoré prídu v odpovedi sú pre SQID potrebné. Pre úplnosť prikladáme UML diagram s atribútmi v odpovedi od Wikidata.

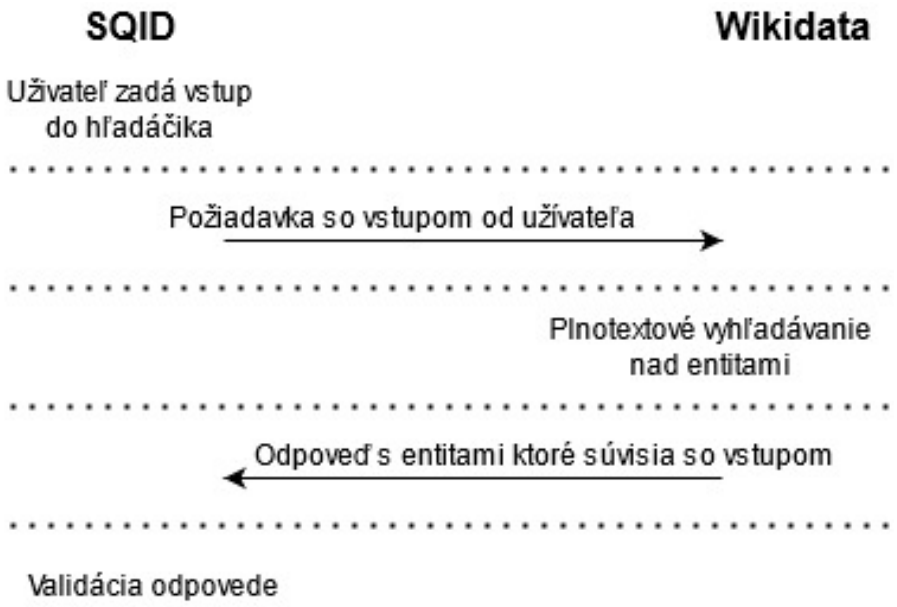


Obr. 10: UML diagram odpovede z Wikidata pri plnotextovom vyhľadávaní

Na prvý pohľad odpoveď vyzerá komplexne. Tabuľka SearchInfo obsahuje jediný atribút search, v ktorom je uložený vyhľadávaný reťazec. Podstatnou časťou pre SQID je hlavne časť tabuľky Entity. Z tejto tabuľky je pre SQID podstatné hlavne ID, pomocou ktorého sa dá entita jednoznačne identifikovať. Ďalej potrebujeme štítok a popis, pretože to sú metainformácie, ktoré SQID zobrazuje užívateľovi. Na záver, odpoveď potrebuje obsahovať Match. Podstatným faktom je, že Match môže obsahovať ľubovoľné hodnoty atribútov. Žiadny z týchto atribútov sa nikde nezobrazuje, ale SQID si vyžaduje aby v odpovedi táto tabuľka bola vyplnená, inak záznam o entite nebude zobrazený. Ostatné atribúty, ktoré sú na diagrame sú podstatné pre Wikidata, ale SQID ich úplne ignoruje.

Z hľadiska optimalizácie by malo zmysel pre náš backend zahrnúť aj atribút pageid a s tým ďalej pracovať, čím by sme do systému zahrnuli aj stránkovanie a teda vyhľadávania na backende by mohlo byť efektívnejšie. Z časových dôvodov sme toto riešenie nestihli implementovať.

Proces plnotextového vyhľadávania zachytáva Obrázok 11.



Obr. 11: Proces vykonania plnotextového vyhľadávania

2.2.4 Zobrazenie stránky pre vlastnú entitu

Po zvládnutí plnotextového vyhľadávania bolo ďalším krokom zobrazenie vlastnej entity. Táto kapitola popisuje detailnejšie krok 7 z kapitoly Základný workflow. Po prijatí odpovede z Wikidata, potom čo si užívateľ vyberie konkrétnu entitu si SQID začne robiť vlastnú validáciu. Ako sme uvádzali už v úvode, SQID je veľmi v zásadnej miere namapovaný na Wikidata.

Vlastnou validáciou máme na mysli validáciu odpovede. V celom procese je najdôležitejším prvkom ID, ktoré SQID príjme. Aby sa v SQID-e ľahšie hľadali chyby a ID zobrazovanej entity sa využíva na mnohých miestach, SQID robí niekoľko dopytov na Wikidata endpoint, aby si zakaždým overil, či ID je platné. Zároveň vždy po prijatí entity a údajov, ktoré k nej patria sa vykonáva ďalšia validácia na strane SQID-u, či prijaté ID je platné Wikidata ID a ak áno, tak na základe prefixu sa určí aký typ entity to je (položka, vlastnosť alebo lexéma).

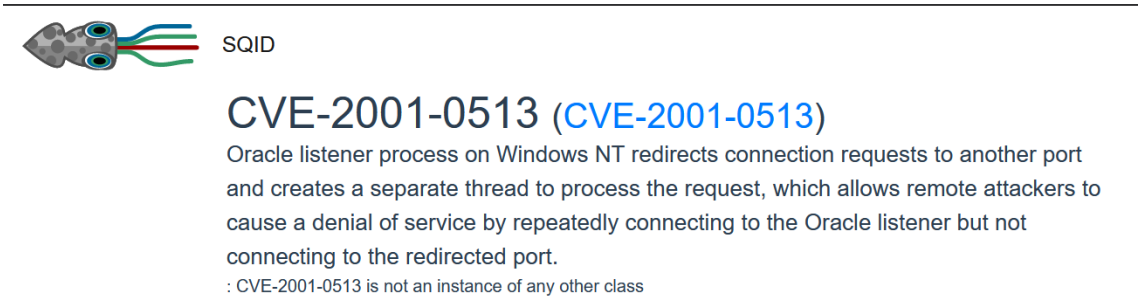
Túto časť validácie sme museli modifikovať, nakoľko v našich používaných ontológiách nedokážeme na základe prefixu rozlišovať, o aký typ entity sa jedná. Navyše naše ontológie

nepoužívajú takéto značenie vôbec. V ideálnom prípade by bolo vhodné, keby sme takéto rozlišovanie nemuseli robiť vôbec, ale všetky komponenty v SQID-e, ktoré zobrazujú údaje o entitách si vyžadujú tento typ mať priradený, preto sme sa rozhodli ísť defenzívnou stratégiou a ponechať tieto typy v SQID-e, pričom typ sa určuje na základe jednoduchého kritéria. Nakoľko v momentálnej situácií sme uvažovali iba CVE zraniteľnosti, tak všetky entity, ktoré začínajú prefixom CVE rozlišujeme ako položky a všetko ostatné je typu vlastnosť.

V časoch, keď sme nemali ešte implementovaný backend, sme museli priamo do kódu vkladať mockované odpovede, ktoré sa použili vždy namiesto odpovede z Wikidata. Po modifikácií validácie ID, bolo možné v SQID-e presmerovať užívateľa na stránku s našou entitou, ale na tejto stránke sa stále nezobrazovali žiadne údaje.

2.2.5 Zobrazovanie údajov na stránke entity

Vo workflowe sa dostávame na Krok 8, kedy sa pokúšame zobraziť konkrétne údaje. Tieto údaje sa získavajú z ďalšej odpovede Wikidata endpointu, na ktorý sa SQID dopytuje po načítaní stránky entity. Namockovaním tejto odpovede sa nám podarilo zobraziť základné údaje ako je popis a názov entity.



Obr. 12: Príklad zobrazenia zraniteľnosti s názvom a popisom.

Dôkladnou analýzou sme sa pokúšali odpovedať opäť na rovnaké 3 otázky, ako sme uviedli v kapitole Plnotextové vyhľadávanie. Wikidata používa jeden endpoint, ktorý slúži jednak na vyhľadávanie entít (kapitola Plnotextové vyhľadávanie) a zároveň aj na získavanie údajov o entitách. Spôsob ako sa endpoint líši je len ako používa parametre a ktoré parametre sú v akej kombinácií kompatibilné. Tabuľka 3 zachytáva aké parametre sú medzi sebou kompatibilné v prípade, že naším úmyslom je získať údaje pre entity.

Názov parametra	**Popis**	**Príklad akceptovanej hodnoty**
Format	Formát odpovede	JSON
Origin	Informácia pre Wikidata, z akej URL sú vyžadované zdroje/informácie (súvisí s CORS hlavičkami)	* (asterisk)
Action	Akcia, používa sa na rozlíšenie medzi hľadaním entít a získavania údajov o entitách	wbsearchentities
Ids	Zoznam entít, pre ktoré má API vrátiť údaje	P41|P31
Props	Zoznam vlastností, ktoré bližšie definujú, aké konkretné údaje chceme vrátiť ku entite.	info|labels
Languages	Zoznam jazykov, v ktorých budú vratené údaje.	en|de
Languagefallback	Reprezentuje možnosť vrátiť údaje v aktuálne zvolenom jazyku (na strane klienta), ak údaje nie sú dostupné v žiadnom z jazykov, ktoré sú špecifikované v parametri Languages	true/false

Tabuľka 3: Parametre endpointu pre získanie údajov ku entite/entitám

Pri porovnaní Tabuľky 2 a Tabuľky 3 je možné si všimnúť, že niektoré parametre sú úplne rovnaké. Podstatný rozdiel v správaní robí práve parameter action, ktorý v prípade, že nadobudne hodnotu "wbgetentities", začne vracať konkrétne údaje o entite. ID entít, pre ktoré má vrátiť endpoint údaje sú špecifikované s pomocou parametra "ids", pričom tieto entity sú oddelené symbolom "|". Parameter props bližšie špecifikuje, aké konkrétne údaje by sme chceli od endpointu vyžiadať.

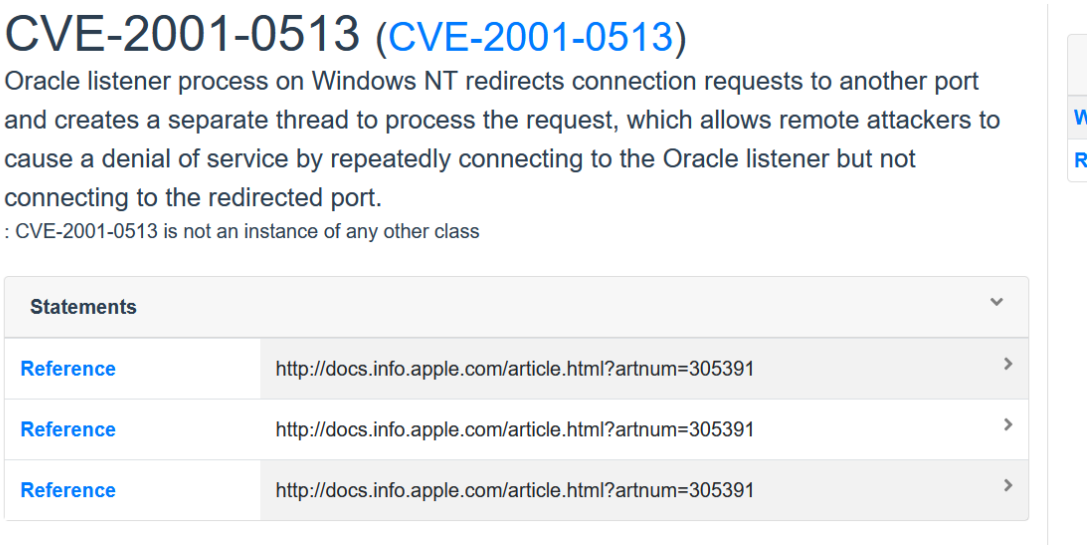
Príkladom props môžu byť napríklad štítky entity alebo popis entity. Správanie endpointu je implementované tak, aby sa pri vyžiadaní popisov alebo názvov entity vracali

vždy údaje vo všetkých dostupných jazykoch. Následnou úlohou SQID-u je, aby zobrazil popis a názov v korektnom jazyku.

V prípade, že je v požiadavke špecifikovaných viacero entít a zároveň viacero props, potom endpoint vráti pre všetky entity všetky špecifikované props. Parameter props môže obsahovať nasledovné hodnoty:

- **Info** - Vráti info o entite (metainformácie ako napr. kedy bola entita modifikovaná, pageid vo wikidata a pod).
- **Sitelinks** - Vráti zoznam Wikimedia stránok (ako napríklad Wikipedia), na ktorých sa entita tiež nachádza.
- **sitelinks/urls** - Vráti zoznam stránok s konkretnými URL, na ktorých sa entita tiež nachádza.
- **Aliases** - Vráti zoznam známych aliasov pre entitu.
- **Labels** - Vyžiada si názov entity (vo všetkých dostupných jazykoch).
- **Descriptions** - Vyžiada si popisy entity (vo všetkých dostupných jazykoch).
- **Claims** - Špeciálna kategória dát, ktoré sú špecifické pre Wikidata.
- **Datatype** - Dátový typ entity

Po zanalyzovaní celého endpointu sa nám podarilo pripraviť mockované údaje, ktoré zobrazili pre entitu zmysluplný obsah. Mockované údaje obsahovali náš vlastný názov, popis a “claims” údaje.



Obr. 13: Príklad zobrazenia mockovaných údajov

2.2.6 Transformácia SPARQL dotazov z Wikidata modelu

V rámci tejto kapitoly sa dostávame k posledným krokom z kapitoly Základný workflow. V tejto kapitole chceme bližšie popísať ako prebieha proces hľadania súvislostí. Proces hľadania súvislostí sa nám z veľkej časti podarilo dotiahnuť do konca z technickej stránky, avšak veľký problém máme so zobrazovaním súvislosti kvôli sémantike zobrazovaných dát.

Aby bolo zrejmé, čo bolo myslené predošlým paragrafom, je potrebné oboznámiť čitateľa, ako SQID vykonáva vyhľadávanie súvislostí počas jeho bežnej prevádzky. Za normálnych okolností SQID pracoval s Wikidata dátovým modelom, v ktorom každá entita má svoj jedinečný identifikátor, pričom typ entity sa určuje na základe prefixu identifikátora entity. Entity vo Wikidata majú ďalšiu špeciálnu vlastnosť, a to takú, že každá entita má svoju unikátnu URL, pričom s pomocou Wikidata API sa dá vďaka tejto vlastnosti s tou entitou ďalej pracovať.

Obrovskou výhodou takéhoto prístupu je, že SQID má vždy zaručené, že pokiaľ pracuje s ľubovoľným typom entity (či už to je položka, vlastnosť alebo lexéma), vždy je možné sa dotazovať na atribúty tejto entity (názov, popis apod) a vždy konzistentným spôsobom. Pre nás to naopak predstavuje obrovský problém, pretože naše riešenie má zhlukovať viaceré ontológie, avšak my nemáme pri každej zaručené, že každá bude používať napríklad štandardizované rdf vlastnosti ako napríklad rdfs:label, dc:description namiesto nejakých svojich vlastne definovaných vlastností. Taktiež nie všetky vlastnosti záznamu sú URI, ale častokrát sa jedná napr. o literály.

Napriek tomu, že na problémy uvedené v predošlom paragrafe sme nenašli jedno-

značné riešenia, v našom prípade to ani nie je potrebné. Našim cieľom ani nie je nájsť v rámci tohto tímového projektu úplne všeobecné riešenie, ale pokúsiť sa nájsť aspoň nejaký model a nejaké údaje v takomto ontologickom modeli, ktoré dokážeme zobraziť. Zobrazovanie je záležitosťou technického charakteru a preto sa nám v budúcnosti môže stať, že do databázy bude importovaný ontologický model, ktorého údaje nami modifikovaný SQID nebude vedieť nájsť a zobraziť. Wikidata je príklad dátového modelu, ktorý sme popisovali vyššie, pretože si implementoval vlastné atribúty pre názvy a popis a teda ak by sme do našej databázy importovali ontológiu s referenciami na Wikidata model, nami modifikovaný SQID by takéto údaje z Wikidata nebol schopný zobraziť.

SQID implementuje niekoľko SPARQL dotazov, ktoré sú žiaľ veľmi úzko späté s Wikidata dátovým modelom. Všetky tieto SPARQL dotazy sme sa pokúsili transformovať, aby bolo možné ich použiť v našej verzií SQID-u. Na záver úvodu len spomenieme, že SPARQL dotazy síce vracajú výsledky, ale nateraz nič s nimi nerobíme, pretože výsledkom nášho snaženia zatiaľ je zobraziť aspoň nejaké údaje v SQID-e. Zobrazovania ďalších údajov o entite z iných entít môže byť predmetom ďalšieho vývoja projektu. Nasledovné SPARQL dotazy, ktoré budeme popisovať sú len predpripravením pôdy pre ďalších pokračovateľov na tomto projekte.

2.2.6.1 Dotaz pre získavanie objektov ku dvojici (subjekt, predikát)

Najjednoduchšími SPARQL dotazmi sú dotazy na získavanie súvisiacich objektov pre zadanú dvojicu (subjekt, predikát). SPARQL dotazy zároveň berie do úvahy aj jazyk, v ktorom majú byť dodané výsledky. Toto konkrétne SPARQL dotazy má viacero použití v rámci SQID-u, konkrétnejšie sa používa na vyhľadávanie hodnôt, ktoré súvisia so zadanou dvojicou (subjekt, predikát). SQID v rámci dotazu vracia aj premennú ?plabel, ktorá je v kóde neskôr využívaná.

```
SELECT ?p ?pLabel WHERE {{
   SELECT DISTINCT ?p WHERE {
            <subj> wdt:<propertyId> ?p .
      }
      SERVICE wikibase:label {
          bd:serviceParam wikibase:language "<lang>"
   }
}
```

Listing 1: SPARQL dotaz na získavanie súvislostí pre dvojicu (subjekt, predikát) pred modifikáciou

SPARQL dotaz sme modifikovali v niekoľkých ohľadoch. V prvom rade nepotrebujeme filtrovať výsledky podľa jazykov, pretože naše dáta, ktoré máme sú iba v anglickom jazyku.

Výsledky, ktoré potenciálne môžeme dostať budú buď v anglickom jazyku, alebo nebudú prítomné vôbec.

Ďalšou modifikáciou bolo, že sme úplne odstrihli vnutornú SELECT klauzulu, čím sa dotaz zjednodušil. V neposlednom rade sme odstránili akékoľvek používanie prefixov, čím riešenie už nie je viazané len na Wikidata.

Poslednou modifikáciou bolo využitie premennej ?plabel. Táto premenná je v novom SPARQL dotaze vyplnená priamo v rámci dotazu.

```
SELECT ?p ?pLabel WHERE {
    <subj> <propertyId> ?p .
    ?p rdfs:label ?pLabel .
}
```

Listing 2: SPARQL dotaz na získavanie objektov pre dvojicu (subjekt, predikát) po modifikácií

2.2.6.2 Dotaz pre získavanie subjektov ku dvojici (predikát, objekt)

Veľmi podobný SPARQL dotaz existuje aj pre zadanú dvojicu (predikát, objekt) kde sa hľadajú všetky súvisiace subjekty. Takéto dotazy SQID využíva na hľadanie inštancií konkrétnej triedy (triedy v zmysle ontológií), súvisiace podtriedy, prípadne ďalšie súvisiace entity. Transformácia SPARQL dotazu prebiehala podobne ako pri prepisovaní dotazu z Listingu 1, preto nebudeme popisovať celý proces znova.

```
SELECT ?p ?pLabel WHERE {{
    SELECT DISTINCT ?p WHERE {
        ?p <propertyId> <obj> .
    }
    SERVICE wikibase:label { bd:serviceParam wikibase:language "<lang>" }
}
```

Listing 3: SPARQL dotaz na získavanie objektov pre dvojicu (predikát, objekt) pred modifikáciou

```
SELECT ?p ?pLabel WHERE {
       ?p <propertyId> <obj> .
       ?p rdfs:label ?pLabel .
}
```

Listing 4: SPARQL dotaz na získavanie objektov pre dvojicu (predikát, objekt) po modifikácií

2.2.6.3 Dotaz pre získavanie súvisiacich údajov pre entitu

Prvým rozličným dotazom oproti dvom predošlým je dotaz na získavanie súvisiacich údajov pre zvolenú entitu. V SQID-e sa po načítaní základných údajov (ako je popis a názov, prípadne "claims") doťahujú ďalšie údaje, ktoré súvisia s aktuálne zvolenou entitou. Treba spomenúť, že tieto údaje sú síce o našej zvolenej entite, avšak tieto údaje pochádzajú z inej entity, a našu entitu iba referencujú.

```
SELECT DISTINCT ?it ?s ?p ?r WHERE {
    ?p wikibase:statementProperty ?ps ;
        wikibase:claim ?pc .
    ?s ?ps wd:<entityId> ;
        wikibase:rank ?r .
    ?it ?pc ?s .
    FILTER( ?p != <http://www.wikidata.org/entity/P31> )
}
```

Listing 5: SPARQL dotaz na získavanie súvisiacich údajov pre zvolenú entitu pred modifikaciou

Transformácia v tomto konkrétnom prípade bola komplikovanejšia ako v predošlých dvoch prípadoch. V prvom rade sme opäť odstraňovali všetky neštandardizované prefixy, ktoré patrili do dátového modelu Wikidata.

Ďalšou podstatnou zmenou bola transformácia RDF literálov v našich dátach na typ URI. Ako sme uvádzali skôr, všetky entity, ktoré sú vo Wikidata majú svoju vlastnú URL. V našom prípade je problém, že naše dáta nie sú takto konzistentne ukladané a môže sa stať, že naše súvisiace údaje nie sú typu URI, ale napr. nejaké reťazcové údaje. Pri takýchto údajoch by SQID hodil chybu a nepokračoval vo vykonávaní kódu preto sme sekvenciou volaní BIND skúšali meniť typ jednotlivých premenných na typ URI.

Treťou podstatnou zmenou je ignorovanie rankovania. Nakoľko rankovanie v našich dátach nemáme a tým pádom ich nemáme ako zoraďovať, tento údaj nevieme použiť. V súčasnom SPARQL dotaze sme ponechali premennú ?r, ktorá v pôvodnom Wikidata dotaze ukladala rank, pretože SQID ju aj tak potrebuje mať prítomnú, hoci aj so žiadnou hodnotou.

Dotaz sa v istom ohľade zjednodušil vďaka tomu, že nemusíme používať FILTER. V pôvodnom dotaze sa mohlo stať, že pôvodná entita je zároveň inštanciou niektorej triedy, pričom v prípade Wikidaty určite každá entita bude inštanciou niektorej triedy. Táto informácia nie je podstatná pretože SQID túto informáciu tak či tak zobrazuje v detaile stránky pri popise a názve.

```
SELECT DISTINCT ?item ?statement ?entity ?r WHERE {
       <entityId> ?property ?item .
       ?property rdf:type owl:ObjectProperty .
       BIND(str(?item) as ?statement) .
       BIND(<entityId> as ?entity) .
       OPTIONAL {?item rdfs:label ?statement .} .
}
```

Listing 6: SPARQL dotaz na získavanie súvisiacich údajov pre zvolenú entitu po modifikácií

2.2.6.4 Dotaz pre súvisiace vlastnosti s entitou

Nasledovný SPARQL dotaz, ktorý ideme popisovať sa vykonáva iba za výnimočných situácií, ktoré môžu nastať. V SQID-e je stanovený limit, koľko súvisiacich údajov sa môže zobraziť, v prípade, že SPARQL dotaz z Listing 5 vráti menej ako 101 záznamov (101 je limit určený SQID-om.), následne sa spúšťajú ďalšie SPARQL dotazy, ktoré sa snažia zarovnať počet záznamov na 100. Tento SPARQL dotaz sa používa na získanie súvisiacich vlastností so zadanou entitou.

```
SELECT DISTINCT ?p {
   ?s ?ps wd:<entityId> .
   ?p wikibase:statementProperty ?ps .
   FILTER( ?p != <http://www.wikidata.org/entity/P31> )
}
```

Listing 7: SPARQL dotaz na získavanie súvisiacich vlastností pre zvolenú entitu pred modifikáciami

Dotaz z Listing 6 je relatívne priamočiary, akurát nesnaží sa vrátiť štandardné vlastnosti, ktoré sú dostupné ku záznamu v sémantickej databáze ale entity, ktoré sú typu vlastnosť. Nakoľko my v našej sémantickej databáze nemáme takéto entity, vraciame všetky vlastnosti, ktoré ku objektu so zadaným ID máme. Odstránili sme FILTER presne pre rovnaké dôvody ako v predošlom SPARQL dotaze.

```
SELECT DISTINCT ?property {
<entityId> ?property ?item .
}
```

Listing 8: SPARQL dotaz na získavanie súvisiacich vlastností pre zvolenú entitu po modifikáciách

2.2.6.5 Dotaz pre údaje ku vlastnostiam

Tento dotaz je druhou fázou po dotaze v predchádzajúcej sekcií. Dotaz pre súvisiace vlastnosti s entitou získa všetky možné vlastnosti, ktoré sa ku entite viažu a pomocou tohto dotazu sa získávajú bližšie informácie. Je veľmi podobný dotazu pre získanie údajov o entite, avšak tentokrát sa hľadajú súvislosti z vlastností entity.

```
SELECT DISTINCT ?it ?s ?p ?r WHERE {
    BIND(wd:<propertyId> AS ?p) .
    ?s ps:<propertyId> wd:<entityId> ;
        wikibase:rank ?r .
    ?it p:<propertyId> ?s .
}
```

Listing 9: SPARQL dotaz na získavanie údajov súvisiacich vlastností pre zvolenú entitu pred modifikaciou

Transformácia bola mierne jednoduchšia oproti dotazu pre súvisiace údaje s entitou. Opäť ignorujeme ranky a premennú ponechávame pretože na pozadí SQID-u nastavujeme rank na konštantnú hodnotu, ktorá pre nás nemá význam.

Toto je jeden z mála dotazov, pri ktorom sme nevedeli urobiť 1:1 transformáciu. V tomto prípade nie sme úplne presvedčení, či dotaz bude podávať vždy správne výsledky, ale je to najbližší možný dotaz, ktorý sa nám podarilo skonštruovať.

```
SELECT DISTINCT ?item ?statement ?property WHERE {
    BIND(<propertyId> AS ?property) .
    ?statement <propertyId> <entityId> .
    ?item <propertyId> ?statement .
}
```

Listing 10: SPARQL dotaz na získavanie údajov súvisiacich vlastností pre zvolenú entitu po modifikácií

2.3 Backend

V tejto sekcii bližšie popíšeme aplikačné riešenie programu. Úlohou implementácie backend servisu je sprostredkovať úspešnú komunikáciu medzi systémom SQID a dátami ontológie. Na prácu s Blazegraph databázou sme použili Java knižnicu com.blazegraph a pomohli sme si Java frameworkom Spring, ktorý nám uľahčil a urýchlil prácu.

Na to aby sme vedeli komunikovať s databázou bolo potrebné naprogramovať a otvoriť spojenie pomocou aplikačného repozitára. To sa implementuje pomocou journal súboru (ktorý ma koncovku .jnl). Je to konkrétna implementácia pre lokálnu a nerozdelenú databázu, pomocou ktorej sa dopytujeme na jednotlivé dáta alebo pomocou ktorej ich

importujeme. Je dôležité podotknúť, že tento súbor sa dá používať iba jednou aplikáciou súčasne. V prípade potreby môžeme tiež otvoriť spojenie k databáze Blazegraph iba v režime na čítanie (napr. servis v produkcii).

Vykonávanie dotazov SPARQL poskytuje najflexibilnejší a najpohodlnejší spôsob získavania informácií z daného repozitára. Po dotazovaní je potrebná iterácia nad výslednými riešeniami. Po iterácií riešení je vždy potrebné ukončiť a uzavrieť tieto riešenia a uvoľniť súvisiace zdroje spojenia. Odpovede servisu boli mapované do JSON formátu.

Tento servis obsahuje ešte aj API dokumentáciu Swagger, ktorá bližšie opisuje jednotlivé endpointy a ich parametre pre ľahšie riešenie integrácie. Návod na spustenie a bližší popis aplikácie sa nachádza v README.md na Github stránke nášho projektu.

2.3.1 Webové servisy REST API

- **CREATE ONTOLOGY AND IMPORT DATA**

  Slúži na automatizované importovanie ontológie a dát, ktoré k nej prislúchajú. Jej cieľom bolo na-importovať jednotlivé dáta rýchlejším spôsobom ako ručne pomocou grafického užívateľského rozhrania Blazegraphu. Endpoint používa metódu POST. Endpoint zlyhá v prípade, že nie sú splnené predispozície endpointu, ktoré sú ovplyvniteľné administrátorom backendu. Splnenie predispozicií sa odvíja od detailov v konfiguračnom súbore Application.yaml. V tomto konfiguračnom súbore je podstatný atribút ontologies, ktorý si vyžaduje cestu ku obsahu github repozitáru našej ontológie, ktorý musí byť uložený na systéme, kde je uložený aj backend. Taktiež sa vyžaduje platná cesta ku .jnl súboru, do ktorého sa budú ukladať dáta a v neposlednom rade je to zoznam názvov súborov, ktoré obsahujú dáta pre OVAL časť ontológie. Ak nebudú splnené niektoré z uvedených podmienok, endpoint vráti návratový kód 500. Nakoľko je tento projekt pre interné používanie tak sme neriešili vhodnejšie odpovede. Endpoint dokáže zlyhať teda z viacerých dôvodov, ktoré treba brať do úvahy. Jedným problémom môže byť, že cesta ku niektorým súborom nemusí byť platná. Ďalším problémom môže byť, že niektoré zo súborov môžu byť poškodené a nebude ich možné rozparsovať. Raritným problémom môže byť, že aplikácia nebude mať ku súborom prístup a teda nebude ich môcť prečítať (môže sa stať najmä na Unixových systémoch). Pri vykonávaní kódu endpointu sa v prvom kroku importuje súbor ontology.rdf, čím bude rozpoznaná celá ontológia. V druhej fáze nasleduje import CVE zraniteľností. V tretej fáze sa importujú všetky OVAL dáta, ktoré boli špecifikované v Application.yaml konfiguračnom súbore. Ak celá operácia bude úspešná, vtedy a len vtedy endpoint dokáže vrátiť návratový kód 200. Jej odpoveď je SUCCESS (úspech) alebo FAILURE (neúspech).

- **QUERY**

  Slúži na dopytovanie k databáze pomocou vstupného query parametru požiadavky query. Odpoveďou je serializovaný JSON podľa špecifikácie W3C skladajúci sa z elementov, ktoré boli použité pri dopyte. Táto API používa GET metódu. Vstupné **query**, parameter API, sa použije ako SPARQL dotaz a výstup sa transformuje ako stream bajtov na špecifikovaný JSON súbor pomocou org.openrdf knižnice. V prípade, že endpoint nezlyhal, klientovi je vrátený návratový kód 200.

  Môže sa stať, že query parameter neobsahuje platný SPARQL dotaz, v takom prípade môže byť vrátený chybový návratový kód 400. Iným scenárom je ak náhodou v parametri query sú nepodporované znaky, v takom prípade dotaz nie je vyhodnotený a endpoint vráti chybový kód 403.

- **SEARCH**

  Slúži na vyhľadanie jednotlivých zraniteľností podľa mena platformy. Endpoint používa metódu GET. Vstupom je meno platformy, ktorá je doplnená a použitá k dotazu na dohľadanie súvisiacich výsledkov. Výsledky sú následne spracované do výslednej štruktúry objektov a to Searchinfo, reprezentujúci informácie o požadovanej platforme, listu objektov Search, reprezentujúce detailné hodnoty vyhľadávaných zraniteľností k platforme, hodnota search-continue, ktorá reprezentuje počet nájdených zraniteľností a hodnoty success, ktorá reprezentuje úspešnosť hľadania. JSON formátu, vhodného pre zobrazenie v systéme SQID.

  Endpoint dokáže prakticky vrátiť iba návratový kód 200. V prípade, že parameter platform je platná platforma, pre ktorú existujú zraniteľnosti, ktoré ju ohrozujú, endpoint vráti tieto zraniteľnosti. Ak náhodou neexistuje žiadná zraniteľnosť ku zadanej platforme alebo neexistuje platforma samotná tak endpoint vráti prázdnu odpoveď.

- **ENTITY**

  Slúži na vyhľadanie požadovanej entity a spracovanie výsledkov pre jej podrobnejší popis do JSON formátu, ktorý využíva systém SQID pre zobrazenie. Operácia používa metódu GET. Vstupným parametrom API je id objektu v ontológií a parameter props, ktorý reprezentuje vlastnosti entity. Výstupom tejto API môžu byť tri druhy odpovedí. Keď je požiadavka poslaná s viacerými id-čkami s použitým delimiterom "|" slovne povedané vertikálna čiara, tak odpoveďou budú viaceré entity s detailným popisom. Ak je id-čko iba jedno a parameter props je o hodnote "info" tak sa vráti jednoduchý popis entity. Ak je id-čko iba jedno a parameter props je iný ako hod-

nota reťazca "info", tak sa vráti odpoveď plnej entity, ktorá obsahuje detailný popis požadovanej entity.

Entita vráti odpoveď 200 pri nájdenej entite a 404 pri nenájdenej entite.

2.3.2 Konfiguračný súbor

Aplikácia využíva konfiguračný súbor, s pomocou ktorého sa dá prispôsobiť konfiguráciu servera. Súbor je možné nájsť v projekte na ceste src/main/resources/Application.yaml Pre jednoduchosť konfigurácia backend využíva yaml súbor, v ktorom sú potrebné nasledovné atribúty.

- **Server** - je hlavná sekcia ktorá definuje konfiguráciu pre server, na ktorom bude bežať backend
- **Port** -prezentuje číselnú položku, ktorá definuje na ktorom porte bude server bežať.
- **Tomcat** - atribút prezentuje konkrétnejšiu konfiguráciu pre Tomcat. V rámci tejto časti je definovaný atribút **relaxed-path-chars**, čo je zoznam znakov, ktoré by mali byť rozpoznané pri parametroch v ceste (tzv. „path param“). Taktiež je možné definovať podobný atribút **relaxed-query-chars**, čo je zoznam znakov, ktoré by mali byť rozpoznané pri argumentoch, ktoré sú za hlavnou cestou v URI (tzv. query parametre, ktoré sú od cesty oddelené symbolom ?).
- **Cross** - atribút obsahuje atribút origin. Ten zase vnorene obsahuje atribút resource, ktorý predstavuje URL, pre ktorú majú byť spristupnené údaje z databázy pri dopytoch.
- **Base** - atribút obsahuje iba atribút url, čo reprezentuje základnú adresu, na ktorej je nasadený server.
- **Resource** - atribút prezentuje načítateľné súbory, ktoré si aplikácia vyžaduje. V rámci tohto atribútu môže byť definovaný atribút **path**, ktorý môže obsahovať dva ďalšie vnorené atribúty. Jedným je atribút **ontologies**, ktorý definuje cestu ku obsahu repozitáru `https://github.com/tp1-SpZIaPvBD/ontologies`, ktorý aplikácia bude potrebovať v prípade, že do aplikácie budú vkladané dáta pomocou CREATE ONTOLOGY AND IMPORT DATA endpointu. Druhým vnoreným atribútom atribútu **path** je atribút **journal**, ktorý definuje cestu ku .jnl súboru. Tento súbor už musí existovať na danej ceste. Atribút **resource** obsahuje ešte aj ďalší atribút **oval**, ktorý obsahuje zoznam súborov oddelených čiarkou, ktoré spadajú do OVAL časti ontológie.

2.3.3 Súčasný stav

Keď sa pozrieme na backend z hľadiska SQID-u, v súčasnom stave backend poskytuje pre frontend možnosť vyhľadávať zraniteľnosti na základe platformy. Následne ak je vrátený nejaký výsledok, SQID sa dokáže dopytovať na detailné informácie ako názov zraniteľnosti, jej popis a referencie na ďalšie zdroje.

Backend ešte nie je v úplne dokončenom stave a detegujeme stále niekoľko možností na vylepšenie. Napríklad je potrebné vylepšiť validáciu na endpointoch. Zároveň treba optimalizovať odpovede ktoré backend posiela, nakoľko v čase implementácia mapoval odpovede na Wikidata odpovede 1:1. SQID nepotrebuje úplne všetky atribúty, ktoré mu chodia z backendu.

V neposlednom rade sa backend neustále bude musieť prispôsobovať požiadavkám SQID-u nakoľko ostalo na strane frontendu niekoľko otvorených problémov, ktoré sa nepodarilo vyriešiť.

Záver

Našim cieľom bolo pripravenie platformy na zdieľanie a uchovávanie poznatkov z bezpečnostnej domény. V rámci projektu sme preskúmali rôzne prístupy, ktoré by boli vhodné na dosiahnutie nášho cieľu tak, aby to stálo čo najmenej úsilia. Na záver sme z nich vybrali najlepšiu možnosť a tou bolo využitie technológie SQID na frontendovú časť, vlastnú Java Springboot aplikáciu na backendovú časť a Blazegraph sémantickú databázu na ukladanie sémantických vzťahov.

Podarilo sa nám pripraviť sémantickú databázu Blazegraph, ktorá tieto poznatky uchováva a eviduje medzi nimi súvislosti. Tieto dáta majú špecifickú štruktúru vo formáte tzv. trojíc, ktoré je možné popísať pomocou RDF jazyka.

Frontendová časť využíva aplikáciu SQID. Táto aplikácia bola upravená tak, aby dokázala zobrazovať dáta a súvislosti medzi nimi z našej databázy. Pomocou napojení na backend a databázu sa dopytuje na jednotlivé dáta, ktoré majú rôzne vlastnosti a údaje. Aplikácia následne prehľadne zobrazí zraniteľnosť pre danú platformu, jej popis a súvislosti s inými dátami, ktoré sú evidované v databáze.

Vytvorili sme aj vlastný backend, ktorý sa na našu sémantickú databázu Blazegraph dopytuje pomocou SPARQL dotazov a vracia dáta, ktoré SQID dokáže zobraziť. Backend bol navrhnutý tak, aby sme nemuseli robiť veľké zmeny na strane SQID-u. To znamená že rozhranie, formát akceptovaných požiadavok a formát odosielaných odpovedí je namapovaný v čo najväčšej možnej miere na rozhranie SQID-u, ktoré bolo implementované na komunikáciu s Wikidata.

V neposlednom rade sme spísali podrobnú dokumentáciu ako aj inštalačný manuál pre dané rozhrania. Ciele tohto projektu sme teda splnili, avšak existuje ešte mnoho ďalších funkcionalít, ktoré by mohli byť v budúcnosti implementované.

Medzi ne patrí napríklad obohatenie funkcionality SQID-u o podporu obrázkov. V pôvodnom riešení bol za toto zodpovedný commons endpoint, pre ktorý sme v súčasnej situácií nenašli vhodný ekvivalent. Ďalším otvoreným problémom je hľadanie súvisiacich poznatkov o danej entite, ktoré je možné získať z poznatkov iných entít, ktoré s danou entitou súvisia. V rámci SQID-u je ďalej potrebné navrhnúť vhodný spôsob ako vytvoriť ekvivalenty ku Wikidata vlastnostiam, nakoľko v bežných ontológiach nemusia mať vlastnosti vždy definovaný názov a teda pri SQID-e je problém ich zobrazovať. V neposlednom rade SQID nemá plnú podporu práce iba s URI, oproti pôvodnému riešeniu, ktoré nepoužívala celé URI ale iba ID Wikidata entít. Taktiež chýba podpora zobrazovania detailov o entitách, ktoré v pôvodnom riešení boli klasifikované ako “property” a zobrazovali všetky

entity, ktoré sa so zadanou vlastnosťou viazali.

Zo strany backendu je možné zapracovať ďalšie vylepšenia vo forme pridania ďalších údajov, ktoré backend dokáže zo sémantickej databázy získať a následne poskytnúť SQID-u. V súčasnom riešení sme sa zameriavali najmä na zobrazenie platformy pre zvolenú zraniteľnosť, ale bol to iba jeden z prvých krokov, aby sme v rámci tohto tímového projektu dosiahli MVP[7].

[7]Minimum Viable Product, teda minimálna verzia produktu, ktorý demonštruje funkčnosť riešenia

Zoznam použitej literatúry

1. BERNERS-LEE, TIM AND CONNOLLY, DAN AND KAGAL, LALANA AND SCHARF, YOSI AND HENDLER, JAMES. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*. 2007, roč. 8. Dostupné tiež z: https://www.researchgate.net/publication/1736816_N3Logic_A_logical_framework_for_the_World_Wide_Web.
2. JAN ENGBERG Heidrun Gerzymisch-Arbogast, Walter de Gruyter. *Knowledge Systems and Translation*. [B. r.]. Dostupné tiež z: https://books.google.sk/books?id=IL2E9xuJLAAC&pg=PA113&redir_esc=y#v=onepage&q&f=false.
3. *What is SPARQL?* Dostupné tiež z: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/.
4. *Resource Description Framework (RDF)*. Dostupné tiež z: https://www.w3.org/RDF/.
5. *RDFS vs. Owl*. Dostupné tiež z: https://www.cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/.
6. *OWL Web Ontology Language Semantics and Abstract Syntax Section 5. RDF-Compatible Model-Theoretic Semantics*. Dostupné tiež z: https://www.w3.org/TR/owl-semantics/rdfs.html.
7. *Semantics and Complexity of SPARQL*. Dostupné tiež z: http://marceloarenas.cl/publications/tods09a.pdf.
8. 2020-05-26. Dostupné tiež z: https://www.mediawiki.org/wiki/MediaWiki.
9. *MediaWiki history*. 2012-12-20. Dostupné tiež z: https://www.mediawiki.org/w/index.php?title=MediaWiki_history&oldid=618636.
10. *Wikidata:Introduction*. 2021-03-29. Dostupné tiež z: https://www.mediawiki.org/w/index.php?title=MediaWiki_history&oldid=618636.
11. *Wikidata:Lexicographical data/Documentation*. 2021-04-26. Dostupné tiež z: https://www.wikidata.org/wiki/Wikidata:Lexicographical_data/Documentation#Lexeme.
12. KRITSCHMAR, Charlie. *Datamodel in Wikidata*. 2016-06-21. Dostupné tiež z: https://en.wikipedia.org/wiki/Wikidata#/media/File:Datamodel_in_Wikidata.svg.

13. JEBLAD. *Linked Data*. 2012-12-31. Dostupné tiež z: https://www.wikidata.org/wiki/Wikidata:Introduction#/media/File:Linked_Data_-_San_Francisco.svg.

14. *Wikidata:Glossary*. 2021-01-16. Dostupné tiež z: https://www.wikidata.org/wiki/Wikidata:Glossary.

15. *Help:Data type*. 2021-05-12. Dostupné tiež z: https://www.wikidata.org/wiki/Help:Data_type.

16. *Wikibase/DataModel*. 2021-02-02. Dostupné tiež z: https://www.mediawiki.org/wiki/Wikibase/DataModel#Datatypes.

17. *Help:Properties*. 2021-05-06. Dostupné tiež z: https://www.wikidata.org/wiki/Help:Properties.

18. *Protege License*. Dostupné tiež z: https://github.com/protegeproject/protege/blob/master/license.txt.

19. *Stanford Protege*. Dostupné tiež z: https://protege.stanford.edu/.

20. ZAREEN SYED, ANKUR PADIA, TIM FININ, LISA MATHEWS AND ANUPAM JOSHI. *UCO: A Unified Cybersecurity Ontology*. Dostupné tiež z: https://ebiquity.umbc.edu/_file_directory_/papers/781.pdf.

21. ZAREEN SYED, TIM FININ AND ANKUR PADIA. *Catalogue of Cybersecurity Standards*. Dostupné tiež z: https://github.com/Ebiquity/uco2/blob/master/docs/Catalogue%5C%20of%5C%20Cybersecurity%5C%20Standards.pdf?fbclid=IwAR2Sqj55ZvD85ZqItC653JUt_nQ6099H0ALc992GOtw_Q5ll7uErZpSBsTU.

22. THE MITRE CORPORATION. *CVE - The Standard for Information Security Vulnerability Names*. Dostupné tiež z: https://cve.mitre.org/docs/cve-intro-handout.pdf.

23. ANDREW BUTTNER. *Enumerations – CPE, CVE, CCE*. Dostupné tiež z: https://csrc.nist.gov/CSRC/media/Projects/Security-Content-Automation-Protocol/documents/docs/2008-conf-presentations/scaptutorial/2-enumerations.pdf.

24. MATEJ RYCHTÁRIK, Bratislava: Univerzita Komenského v Bratislave. *Sémantické publikovanie spravodajských dát o bezpečnostných hrozbách"*. 2021.

25. BEBEE, Brad (zost.). *Blazegraph - wiki*. 2020-02-13. Dostupné tiež z: https://github.com/blazegraph/database/wiki.

26. *A fresh look at Wikidata*. Dostupné tiež z: https://sqid.toolforge.org/#/.

27. Dostupné tiež z: https://github.com/Wikidata/SQID.

28. KROETZSCH, Markus. *SQID: the new "Wikidata classes and properties browser"*. 2016-04-20. Dostupné tiež z: https://lists.wikimedia.org/pipermail/wikidata/2016-April/008554.html.

Prílohy

A Štruktúra zip súboru

/Dokumentácia TP

· .pdf súbor tejto dokumentácie

/Zápisnice

· priečinok obsahujúci zápisnice z I-TP1-AI a I-TP2-AI

/api_response_json_examples

· priečinok obsahujúci jednotlivé súbory príkladov predstavujúce odpovede z web service-ov na našom back-ende

ENTITY_API_RESPONSE.json

.

QUERY_API_RESPONSE.json

.

SEARCH_API_RESPONSE_PLATFORM_SOLARIS.json

.