VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INTERAKTIVNÍ GRAFICKÁ REPREZENTACE DAT NA WWW

INTERACTIVE GRAPHICAL DATA PRESENTATION ON THE WWW

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **KRISTIÁN ŠKROBÁNEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20646/2017/xskrob05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Škrobánek Kristián**
Obor: Informační technologie
Téma: **Interaktivní grafická prezentace dat na WWW**
Interactive Graphical Data Presentation on the WWW
Kategorie: Web

Pokyny:

1. Prostudujte možnosti a technologie interaktivní grafické prezentace dat ve webových aplikacích.
2. Seznamte se s existujícím systémem pro sběr událostí a jeho aplikačním rozhraním.
3. Na základě konzultací s vedoucím navrhněte architekturu aplikace pro grafickou prezentaci událostí a vztahů mezi nimi pomocí webového rozhraní.
4. Implementujte navrženou aplikaci pomocí vhodných technologií. Zaměřte se i na zpracování velkého množství dat.
5. Proveďte testování vytvořené aplikace na reálných datech.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Lasila, I., Swick, R. R.: Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015
- Scalable Vector Graphics, The Wold Wide Web Consortium, http://www.w3.org/Graphics/SVG/

Pro udělení zápočtu za první semestr je požadováno:

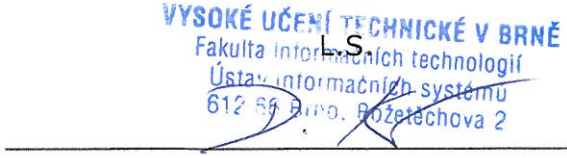
- Body 1 až 3

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

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

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

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018



L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá interaktivní grafickou reprezentací velkého množství dat, tak aby byl uživateli poskytnut co nejlepší přehled o požadovaných informacích. Po průzkumu možností a technologií zabývajících se danou problematikou, byl určen vhodný nástroj, vymyšlen koncept a následně navrhnuta samotná data zobrazující aplikace. Mezi výhody tohoto řešení patří přehlednost, rychlost zobrazení, snadná manipulace a intuitivní průzkum dat. První část práce se zabývá technologiemi, přístupy a návrhem řešení problému, druhá pak detaily realizace a testováním aplikace.

Abstract

This bachelor thesis focuses on interactive graphical presentation of large amounts of data, in a way which presents user with best possible view of wanted information. After research of possibilites and technologies focusing on this matter, the suitable tool was chosen, concept was created and then was projected the data visualizing aplication itself. This solution offers benefits such as clarity, speed of visualization, easy manipulation and intuitive view of data. First part of the work focuses on used technologies, approaches to the problem and design of solution. Second part contains details of realization and testing of application.

Klíčová slova

Interaktivní grafická reprezentace dat, Webová aplikace, Grafické uživatelské rozhraní, Vis.js, Online příspěvky, Resource Description Framework, Zobrazení vztahů, TimelineAnalyzer

Keywords

Interactive graphic data representation, Web application, Graphic user interface, Vis.js, Online posts, Resource Description Framework, Relation visualisation, TimelineAnalyzer

Citace

ŠKROBÁNEK, Kristián. *Interaktivní grafická reprezentace dat na WWW*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Interaktivní grafická reprezentace dat na WWW

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Kristián Škrobánek
12. května 2018

Poděkování

Chtěl bych poděkovat Ing. Radku Burgetovi, Ph.D. za ochotu, pomoc, rady a užitečné poznámky při vytváření aplikace a psaní této práce.

Obsah

Kapitola 1

Úvod

S rostoucím využitím a důležitostí sociálních sítí vzniká stále větší potřeba a zájem analyzovat jejich příspěvky a data s nimi spojená. Při sběru a uložení takovýchto údajů pak vyvstává problém jejich vizualizace.

Vizualizace velkého množství dat je záležitostí, nad kterou lze zaujmout různé pohledy. Sekvenční procházení a zobrazení jednotlivých příspěvků je pouze jedním z možných způsobů jak se na věc dívat a předmětem mé práce je najít způsob alternativní.

Při hledání alternativního způsobu byl kladen důraz na následující atributy. Možnost interaktivity grafického prostředí, možnost manipulace s prostředím a pojmutí problému zajímavým způsobem.

Při zamyšlení nad tím, jak nejlépe zobrazit data sociálních sítí je třeba brát v úvahu rozličné aspekty. Je třeba uvážit jaká forma zobrazení by nejvíce blížila intuitivnímu, snadno orientovatelnému prostředí, jaká volba vzhledu prostředí by byla uživatelsky nejpřijatelnější, jaký nástroj je schopen rychle a efektivně vyvolat zobrazení odpovídající daným nárokům, za podmínky, že bude výkonnostně dostatečný. Problémem není však pouze zobrazení dat samotných, ale také vztahů mezi těmito daty. Propojení jednotlivých příspěvků je další aspekt, který je třeba brát v úvahu.

Z tohoto vyplývá, že hledáme řešení, které poskytuje možnost interaktivní manipulace s daty příspěvků sociálních sítí, mezi kterými vidíme návaznost, vizualizaci vztahů, jasné logické umístění v grafickém zobrazení a zároveň intuitivní a uživatelsky přístupnou práci s celým zobrazením.

Součástí práce tedy, kromě samotného vymýšlení konceptu a následného programového řešení je průzkum a posléze studium možných nástrojů a technologií použitelných k danému účelu. Po vybrání adekvátního nástroje splňujícího určená kritéria, je nutné vypracovat návrh zobrazení dat, který vyhovuje požadavkům a posléze tento návrh implementovat.

V poslední fázi je vzhledem k použití výsledného produktu také nutno řádného testování a zjištění nutných dodatečných úprav v pozdější fázi projektu. K tomuto účelu je zapotřebí použít velké množství testovacích dat.

Kapitola 2

Možnosti a technologie interaktivní grafické reprezentace dat ve webových aplikacích

V první fázi probíhal průzkum mnoha různých možností, který spočíval v pozorování nabízené funkčnosti a základním zkoušení různých demo příkladů na oficiálních stránkách daných nástrojů. V této fázi bylo z desítek možných javascriptových knihoven vybráno do užšího okruhu pouze několik. Do toho okruhu byly vybrány právě na základě nabízených možností interaktivní vizualizace dat.

2.1 Cíle a kritéria výběru

Před průzkumem jednotlivých technologií a možností je nutné určit, dle jakých kritérií se budou dané přístupy posuzovat. Již bylo zmíněno, že se jedná o interaktivní vizualizaci. V této části si podrobněji popíšeme, co se od tohoto pojmu očekává a jaké další záležitosti jsou s ním spojené.

V první řadě chceme, aby bylo možné jednotlivě graficky reprezentovat dané příspěvky, neboli chceme, aby byli představovány nějakými prvky. Mezi příspěvky pak mohou být určité vztahy a tyto vztahy je opět nutno nějak graficky vizualizovat. Poté přichází otázka interaktivity neboli možnosti vyvolání nějaké reakce při manipulaci s daty. Mezi tyto reakce patří například možnost pohybu s prvky, vyvolání nějaké odezvy při kliknutí na prvek nebo reakce celkového zobrazení na nějakou uživatelem vyvolanou událost.

Taky je nutné zvážit, jestli je možné měnit vlastnosti zobrazených dat za běhu, tedy po iniciálním zobrazení nebo lze data ovlivňovat pouze před aktivováním funkcí dané knihovny.

Poté je zde otázka výkonnosti a času zobrazení. Je důležité prověřit, jak dlouho trvá dané technologii zobrazit určitá data, obzvláště v našem případě, kdy očekáváme práci s velkým objemem dat. Poté co jsou data zobrazena, nás pak zajímá vliv knihovny na výkonnost zobrazování v prohlížeči, při manipulaci se zobrazovací plochou. Je tedy nutné ověřit, do jaké míry manipulace se zobrazenou sítí dat ovlivňuje Frames Per Second, abychom jsme dosáhli co nejlepší uživatelské zkušenosti.

Dalším požadavkem je, aby daná technologie byla dobře zdokumentovaná, zpracovaná a zcela funkční. Je důležité, aby při práci s danou technologií a případném řešení potíží nebo zpracování implementačních úkolů bylo možné danou situaci vyhledat a efektivně vyřešit.

Pokud tedy sesumarizujeme hlavní požadavky, klademe důraz na možnost grafické reprezentace příspěvků a vztahů, schopnostech manipulace s daty a možnostmi interaktivních reakcí na podněty uživatele. Dále nás pak zajímají možnosti manipulace s daty po počátečním volání knihovny, rychlosti zobrazení a výkonnosti při práci se zobrazením. Na závěr pak samotnou kvalitou zpracování technologie.

2.2 Výběr zajímavějších technologií

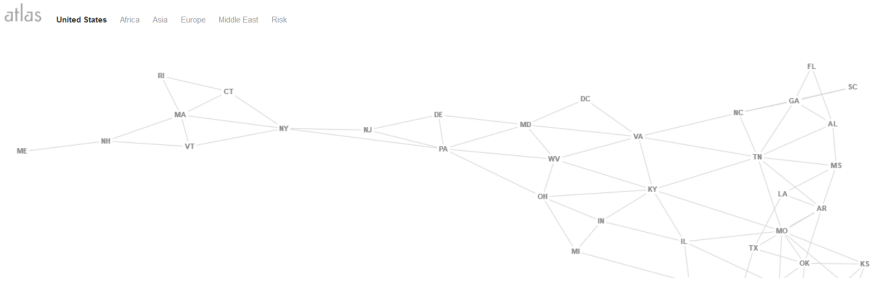
Do tohoto výběru byly vybrány následující knihovny.

Arbor.js Knihovna[1] prezentuje data v interaktivní formě, kdy lze s jednotlivými prvky manipulovat, hýbat a vyvolávat interaktivní reakce. Dynamika jednotlivých prvků je, ale příliš náhodná a uživatelsky nepříjemná. Při větším počtu dat se s prvky manipuluje přinejmenším obtížně a celkové zobrazení ztrácí na přehlednosti. Knihovna je tedy vhodná především pro použití při menších počtech prvků, které potřebujeme zobrazit a v situacích, kdy neklademe důraz na přehlednost a stálou pozici prvků.

Vlastnosti knihovny	Arbor.js
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ano
Interaktivita	Ano
Dynamická změna dat	Ano
Pohyb prvků	Ano

Tabulka 2.1: Tabulka vlastností knihoven

Dle tabulky kritérií sice knihovna splňuje jednotlivé požadavky, ale z dříve zmíněných důvodů není vhodné knihovnu zařadit do užšího výběru. Na ukázce vidíme příklad zobrazení dat.



Obrázek 2.1: Arbor.js demo

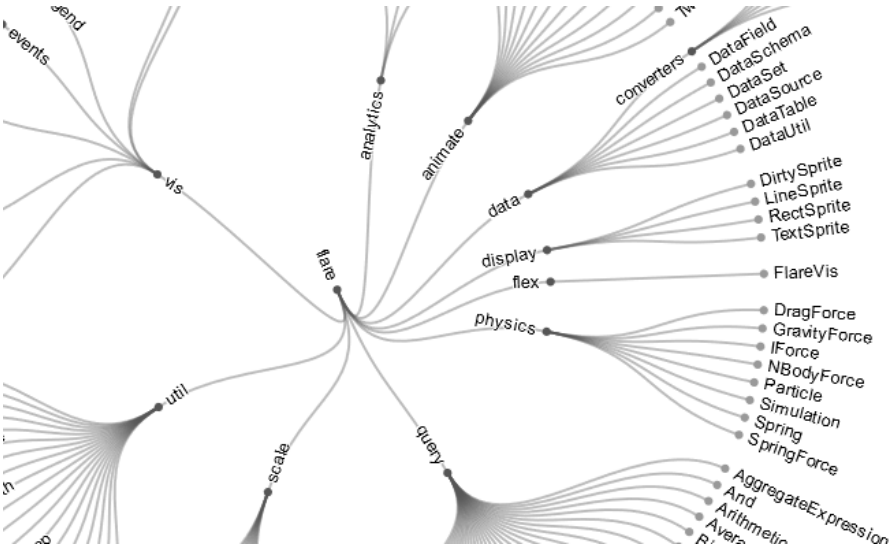
D3.js Rozsáhlá knihovna[2] pro vizualizaci dat, nabízející téměř neomezené možnosti. Její nevýhodou je nízká intuitivnost a nutnost produkce delších kódů pro dosažení požadovaných výsledků. Výhodou však je, že dokážeme napodobit i složitější vzorce požadovaného chování.

[1] Knihovna Arbor.js http://arborjs.org/
[2] Knihovna D3.js https://d3js.org/

Vlastnosti knihovny	D3.js
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ano
Interaktivita	Ano
Dynamická změna dat	Ano
Pohyb prvků	Ano

Tabulka 2.2: Tabulka vlastností knihovny

Dle tabulky můžeme vidět, že lze dosáhnout splnění požadovaných kritérií, nicméně využití knihovny D3 by znamenalo spíše vytvoření nového nástroje než využití nástroje existujícího, což není předmětem práce. Na ukázce můžeme vidět příklad vizualizace dat.



Obrázek 2.2: D3.js ukázka

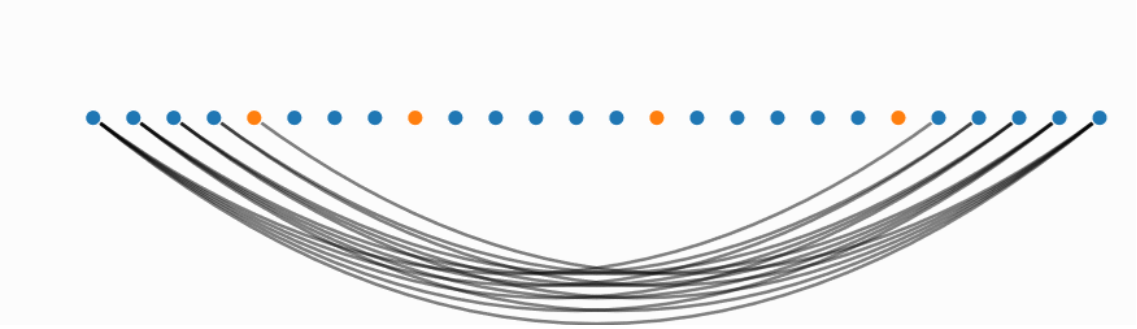
Candella Knihovna[3] vhodná k reprezentaci dat v podobě grafů. Kromě toho umožňuje znázornění dat v podobě uzlů grafu. Neposkytuje však možnost dynamického pohybu s prvky, proto pro náš účel není v tomto případě tato knihovna příliš vhodná.

Vlastnosti knihovny	Candella
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ano
Interaktivita	Ano
Dynamická změna dat	Ne
Pohyb prvků	Ne

Tabulka 2.3: Tabulka vlastností knihovny

Z výsledku tabulky kritérií a předešlých důvodů tedy není knihovna přidána do užšího výběru. Na ukázce můžeme vidět zobrazení dat v podobě grafu za pomoci využití sady Candella.

[3] Knihovna Candella https://candela.readthedocs.io/en/latest/index.html



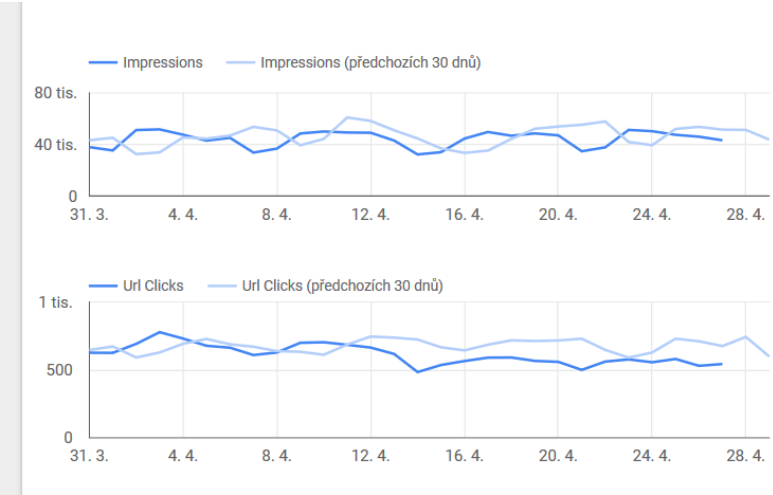
Obrázek 2.3: Candella ukázka

Data Studio Gallery Nástroj[4] společnosti google k vizualizaci dat. Poskytuje přehledné a intuitivní rozhraní k vytvoření přehledu nad určitými daty. Jedná se spíše o přehledy určené k obchodním aktivitám, tedy nabízí možnosti různých druhů grafů a statistických údajů určených k získání pojmu o datech, nad kterými je přehled vytvořen. V našem případě není vyhovující z důvodu, že nenabízí možnost libovolného zobrazení a dynamické manipulace s datovými prvky.

Vlastnosti knihovny	Data Studio Gallery
Reprezentovatelnost příspěvků jednotlivými prvky	Ne
Možnost znázornění vztahů	Ne
Interaktivita	Ano
Dynamická změna dat	Ne
Pohyb prvků	Ne

Tabulka 2.4: Tabulka vlastností knihovny

Z tabulky kritérií a zmíněných informací lze usoudit, že nástroj je pro náš účel nevhodný a není tedy zařazen do další fáze výběru. Na ukázce můžeme vidět příklad vyobrazení grafů pomocí tohoto nástroje.



Obrázek 2.4: Data Studio Gallery ukázka

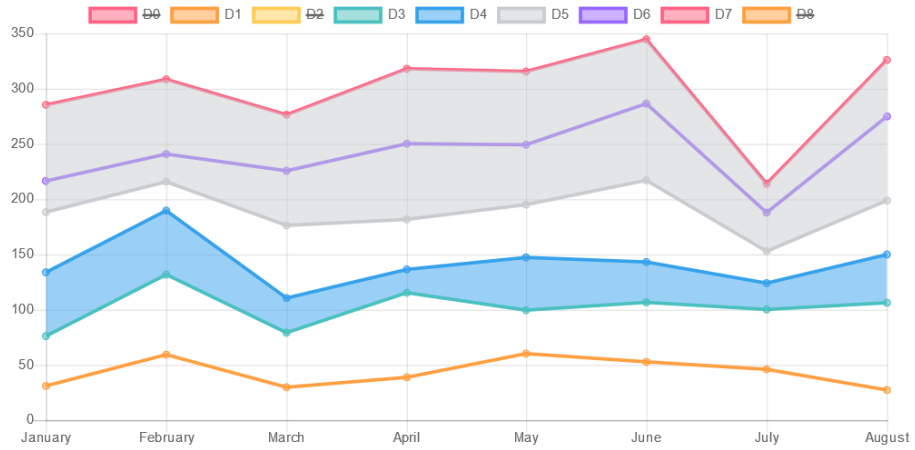
[4] Data Studio Gallery https://www.google.com/analytics/data-studio/gallery/

Chart.js Knihovna[5] určená především k zobrazení dat formou různých či kombinovaných grafů. Nabízí určitou dávku interaktivity prostřednictvím zobrazení dodatečných informacích o prvcích při kliknutí, avšak chybí zde funkcionalita možnosti s prvky pohybovat či je nějak ovládat.

Vlastnosti knihovny	Chart.js
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ne
Interaktivita	Ano
Dynamická změna dat	Ano
Pohyb prvků	Ne

Tabulka 2.5: Tabulka vlastností knihovny

Z tabulky kritérií a zmíněných informací vychází, že knihovna nesplňuje základní kritéria a tedy nevyhovuje našim účelům. Není tedy zařazena do užšího výběru. Na ukázce můžeme vidět příklad zobrazení dat.



Obrázek 2.5: Chart.js ukázka

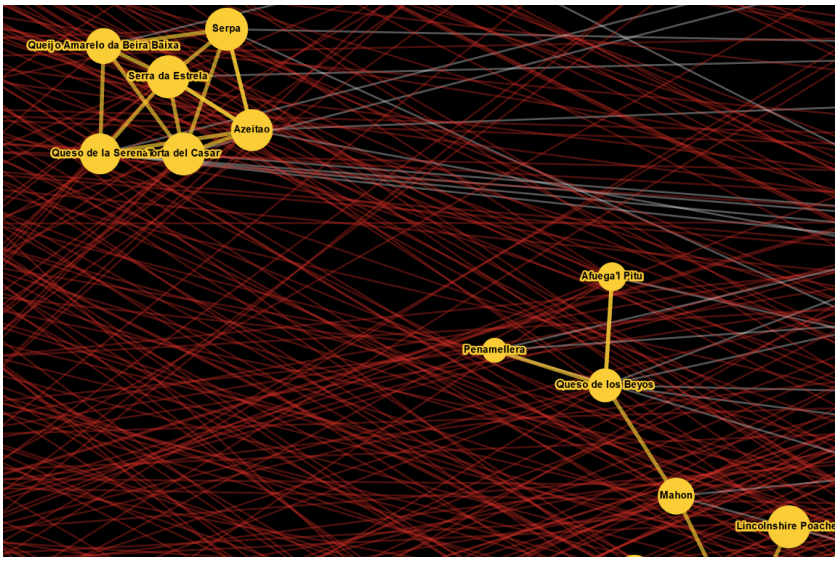
Cytoscape.js Knihovna[6] poskytující možnost zobrazení dat v pohyblivých prvcích, které je možno navzájem propojovat a ovlivňovat jejich dynamiku. Lze využít funkcí jako je změna pozice prvků celé propojené sítě manipulací s jedním prvkem nebo naopak uchovávat pozici prvku po její změně s tím, že změna pozice jednoho prvku neovlivní zbytek sítě. Prvky poskytují možnost interaktivního zobrazení dalších dat. Další z možností, kterou knihovna nabízí je animovaný přechod mezi přiblížením jednotlivých prvků, což je uživatelsky přátelská vlastnost, kterou lze při vizualizaci dat vhodně využít.

[5] Knihovna Chart.js http://www.chartjs.org/samples/latest/
[6] Knihovna Cytoscape.js http://js.cytoscape.org/

Vlastnosti knihovny	Cytoscape.js
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ano
Interaktivita	Ano
Dynamická změna dat	Ano
Pohyb prvků	Ano

Tabulka 2.6: Tabulka vlastností knihovny

Splnění kritérií v tabulce spolu se zmíněnými informacemi o knihovně je důvodem k zařazení knihovny do užšího výběru. Na ukázce pak můžeme vidět příklad zobrazení dat knihovnou Cytoscape.js.



Obrázek 2.6: Cytoscape ukázka

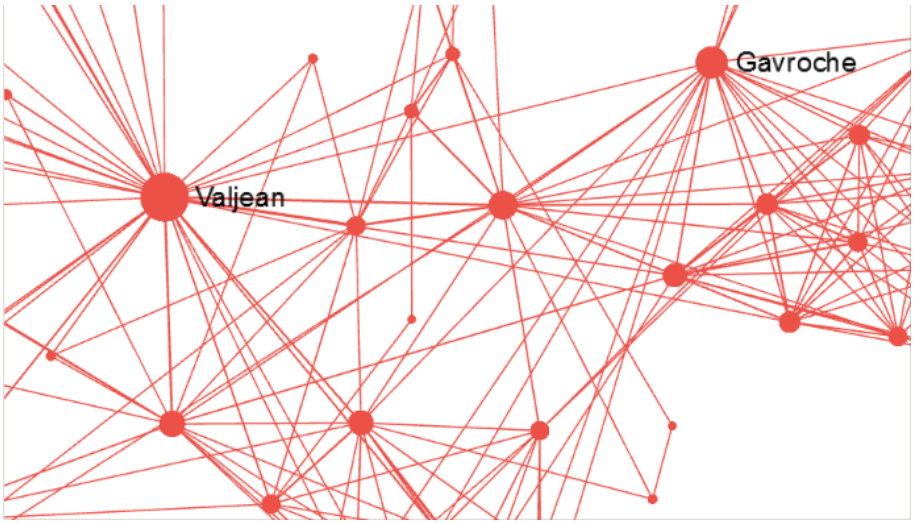
Sigma.js Knihovna[7] pro vizualizaci dat uložených ve formátu JSON. Knihovna vytváří sítě prvků na základě souboru JSON. Poskytuje možnost volání callback funkcí ovlivňujících vzhled sítě a také animací s možností libovolně měnit rozložení sítě. Výhodou této knihovny je vyšší výkonnost i při vyšším počtu dat. Neumožňuje pohyb s prvky nebo jiné dynamické chování mimo animaci.

Vlastnosti knihovny	Sigma.js
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ano
Interaktivita	Ano
Dynamická změna dat	Ano
Pohyb prvků	Ne

Tabulka 2.7: Tabulka vlastností knihovny

[7] Knihovna Sigma.js http://sigmajs.org/

Z tabulky můžeme vidět, že knihovna splňuje všechny kritéria, kromě pohybu prvků v síti. Tuto funkčnost lze však částečně nahradit funkcemi umožňujícími přesun prvků. Na základě zmíněných informací a splnění kritérií byla tabulka vybrána do užšího výběru. V ukázce můžeme vidět demo zobrazení dat pomocí této knihovny.



Obrázek 2.7: Sigma.js ukázka

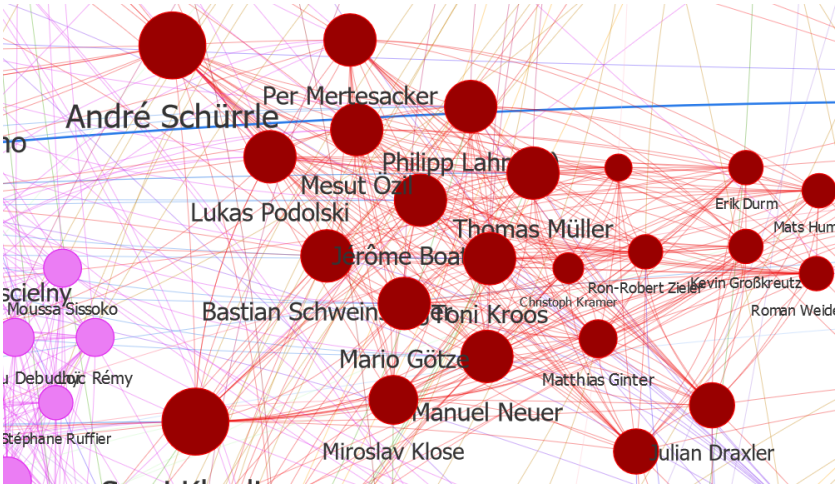
Vis.js Knihovna[8] umožňující několik různých způsobů interaktivní reprezentace dat. Mezi tyto způsoby patří interaktivní 2D a 3D grafy, rozložení dat v časové ose vytvořené knihovnou a určené nebo náhodné rozložení dat v uzlech, mezi kterými lze vytvářet grafická propojení. Knihovna poskytuje různé možnosti interaktvity pomocí funkcí reagujících na události v grafické ploše. Pohyby kamerou, zaměření na prvky, v určitých typech zobrazení i pohyb prvků. Rozličné možnosti vizualizace, interaktivity a kvalitní dokumentace, dělají z této knihovny jeden z nejzajímavějších návrhů na řešení.

Vlastnosti knihovny	Vis.js
Reprezentovatelnost příspěvků jednotlivými prvky	Ano
Možnost znázornění vztahů	Ano
Interaktivita	Ano
Dynamická změna dat	Ano
Pohyb prvků	Ano

Tabulka 2.8: Tabulka vlastností knihovny

V tabulce vidíme, že knihovna kromě již zmíněných pozitivních vlastností také splňuje základní kritéria výběru technologií a je tedy možné s ní počítat do užšího výběru. V ukázce pak vidíme jedno z mnoha možných zobrazení dat za pomoci knihovny Vis.js.

[8] Knihovna Vis.js http://visjs.org/



Obrázek 2.8: Vis.js ukázka

2.3 Užší výběr

V této fázi došlo k zhodnocování knihoven nejen na základě prezentačních demo příkladů, ale i bližšího studia dokumentace a zkoušení možností knihoven programováním. Do užšího výběru se dostali knihovny Cytoscape.js, Sigma.js a Vis.js, které nabízejí rozsáhlé možnosti vizualizace dat, intuitivitu a přijatelnou výkonnost. Vlastnosti knihoven užšího výběru popisuje následující tabulka.

Vlastnosti knihoven	Vis.js	Cytoscape.js	Sigma.js
Interaktivita	Ano	Ano	Ano
Pohyb prvků	Ano	Ano	Ne
Rozsáhlá dokumentace	Ano	Ano	Ne
Rozsáhlá sbírka demo příkladů	Ano	Menší sbírka	Velmi malá sbírka
Nabídka detailních funkcí	Ano	Ano	Méně funkcí
Intuitivnost	Ano	Méně intuitivní	Nutnost převodu dat do JSON

Tabulka 2.9: Tabulka vlastností knihoven

V tabulce 2.9 můžeme vidět srovnání jednotlivých vlastností knihoven. Knihovna Sigma.js v porovnání s ostatními nedisponuje základním pohybem prvků, má pouze menší počet knihovních funkcí nad zobrazením dat a její součástí je pouze menší sbírka demo příkladů se stručnější dokumentací. Na základě těchto informací i přes to, že má knihovna dobré výkonnostní výsledky a je v ostatních účelech vyhovující, není vybrána jako výsledný nástroj.

Knihovna Cytoscape.js pak nabízí i možnost pohybu s prvky a rozsáhlejší nabídku funkcí týkajících se možností manipulace s již zobrazenými daty a celým zobrazením. V porovnání s knihovnou Vis.js však působí méně intuitivně a nenabízí tak kvalitní prezentaci příkladů a využití knihovny.

Na základě těchto informací došlo po důkladném zvážení k výběru knihovny **Vis.js** jako nástroje vhodného k interaktivní grafické reprezentaci dat.

2.4 Knihovna vis.js

Knihovna nabízí kvalitní možnost reprezentace dat. Kromě rozsáhlé dokumentace a početných ukázkových příkladů s jejími rozličnými využitími je samotné programování s knihovnou intuitivní a dobře uchopitelné. Knihovna poskytuje kromě samotného základního zobrazení dat v prvcích a jejich vztahů pomocí křivek mnohé přizpůsobující možnosti. Například možnost animace a kamerových přechodů mezi prvky, dále jednoduchý způsob navigace přímo v zobrazovací ploše nebo možnost dynamického přidávání a odebírání prvků. Knihovna poskytuje možnost reakcí na události odehrávající se v zobrazovací ploše, například se může jednat o kliknutí, přejetí myší nebo přesun prvku. Kromě funkčních schopností je přítomna velká škála možností jak měnit vzhled prvků nebo křivek propojujících jednotlivá data. Lze měnit velikost, tvar i barvu. Jednou ze základních funkcí knihovny je možnost dynamicky ovlivňovat více prvků najednou a odvíjet jejich pohyb a umístění od centra sítě. Tato funkce se v knihovně nazývá physics. Nevýhodou však je, že pokud je tato funkce aktivní, nelze prvky rozmisťovat algoritmicky, ale rozmístění je ponecháno na knihovně samotné. Takovéto rozmístění je pak náhodné. Lze ho ovlivňovat změnami vlastností funkce physics, ale výsledek nebude plánované rozložení. Tento způsob rozmístění nemusí být vhodný pro všechny účely a tak lze zvolit možnost deaktivace této funkce a prvky rozkládat algoritmicky s tím, že pohybovat lze vždy pouze s jedním prvkem najednou. Následující tabulka ukazuje výsledky z testování výkonnosti.

Počet prvků	Průměrné FPS u Dynamické křivky	Průměrné FPS u Kontinuální křivky
25	59	59
100	50	59
200	40	45
300	32	37
400	25	33
500	15	30
600	14	25
1000	9	18

Tabulka 2.10: Tabulka FPS v prohlížeči u různých typů zobrazení křivek

Výsledkem průzkumu výkonnosti snímkování při manipulaci se sítí je fakt, že při počtech do pětiset navzájem propojených uzlů dosahujeme velmi příznivých hodnot snímků za sekundu. V případě, že chceme zobrazovat větší počet dat, je nutné zahrnout určitá výkonnostní opatření. Knihovna **Vis.js** je pro tento případ vybavena možnostmi jako je například přepínání zobrazování a nezobrazování hran při pohybu v síti, což výrazně pozitivně ovlivňuje plynulost zobrazování. Hrany je možno přepínat i manuálně, což poskytuje uživateli možnost přizpůsobit zobrazení dat podle situace. V případech extrémního objemu dat je možné vypínat i zobrazování uzlů, síť se pak zobrazuje pouze ve chvíli, kdy se v ní nepohybujeme. V závěru tedy lze říct, že knihovna **Vis.js** splňuje požadované kritéria a zároveň nabízí mnoho možností jak řešit potenciální výkonnostní potíže.

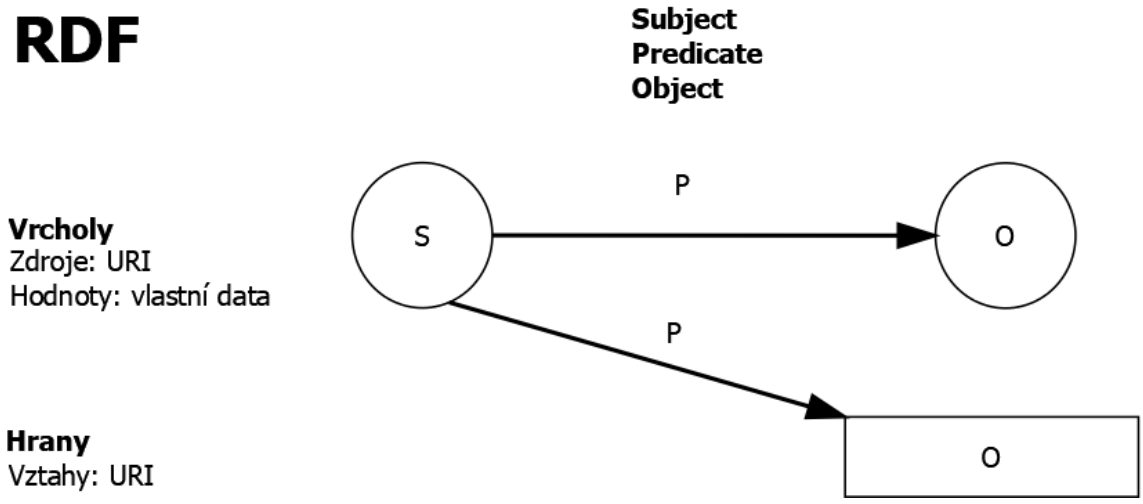
Kapitola 3

Sémantický web a jeho prvky

V této kapitole se budeme zabývat sémantickým webem a jeho prvky, neboť je na něm práce založena [14]. Jedná se o web, ve kterém jsou informace uloženy a strukturovány podle standardizovaných pravidel pro usnadnění jejich vyhledávání a zpracování. Na specifikacích pro sémantický web pracuje World Wide Web Consortium W3C. Sémantický web je založen na standardu Resource Description Framework.

3.1 RDF datový model

RDF neboli resource description framework je standardizovaný model pro výměnu dat na webu [7]. RDF má vlastnosti, které usnadňují spojování dat i když se základní schémata liší a konkrétně podporuje časový rozvoj schémat bez vyžadování změn spotřebitelů. RDF rozšiřuje propojovací strukturu webu k použití URI k pojmenování vztahů mezi prvky. Použití tohoto jednoduchého modelu umožňuje směšování a šíření strukturovaných a polostrukturovaných dat mezi různými aplikacemi. Spojovací struktura formuje orientovaný graf, kde hrany reprezentují pojmenované spojení mezi dvěma zdroji, representovanými grafovými uzly. Obecnou definicí lze říct, že RDF datový model je systém pro popis zdrojů, kdy konceptem je popsání zdroje pomocí trojice subjekt - predikát - objekt. Takový zdroj může být jakýkoliv zdroj, který můžeme v rámci webu popsat URI uniform resource identificator. Základní princip je uveden na následujícím grafu.



Obrázek 3.1: RDF data model

3.2 Ontology web language

W3C Web Ontology Language neboli OWL je sémantický webový jazyk stvořený k reprezentaci komplexní znalosti o prvcích, skupinách prvků a vztahů mezi prvky [6]. OWL je výpočetní, logicky založený jazyk, takový že znalosti vyjádřené v jazyce OWL mohou být zpracovány počítačovými programy, tedy ověřit, že znalosti jsou konzistentní nebo udělat z implicitních znalostí explicitní. OWL je jeden z jazyků, sloužících k popisu ontologií. Ontologie, mohou být publikovány na WWW, odkazovat nebo být odkazovány z jiných OWL ontologií. Ontologie jsou formální definice slovníku, které umožňují definovat nové komplexní struktury a nové vztahy mezi slovníkovými pojmy a mezi členy námi definovaných tříd. I bez definování velké a komplexní ontologie můžeme při vývoji sémantického webu použít malý počet tříd a vlastností OWL k přidání metadat do našich trojic.

Obecně lze říci, že velkou výhodou využívání RDF schématu a jazyku OWL je možnost vyjádření vztahů nebo informací o zdrojích bez nutnosti tyto informace explicitně zadávat.

3.3 Jazyk SPARQL

SPARQL je RDF dotazovací jazyk, pomocí kterého lze zasílat dotazy na databázi, která obsahuje data uložená v RDF formátu [11]. Data lze pomocí takovýchto dotazů manipulovat a získávat [1]. Dotaz je opět formulován podobou trojic. Základní funkčnost si lze ukázat na příkladu.

```
//Data
<http://example.com/fruits/orange> <http://myexample.org/content/colour> "Orange"
<http://example.com/fruits/banana> <http://myexample.org/content/colour> "Yellow"

//Querry
PREFIX pr: <http://myexample.org/content/>
SELECT ?fruit ?fruitcolour
WHERE { ?fruit pr:colour ?fruitcolour }
```

Výpis 3.1: Ukázka SPARQL querry na datech

V části dat můžeme vidět RDF reprezentaci dat pomocí trojice subjekt predikát objekt. Vezmeme-li v úvahu první trojici pak je v našem případě subjektem ovoce pomeranč, které má vlastnost jež je určena predikátem. Tato vlastnost je barva a hodnota barvy je pak určena objektem oranžová.

V části dotazu se pak chceme zeptat jaké ovoce má jakou barvu. Nejprve v klauzuli prefix určíme vhodný prefix pro usnadnění dotazování. Následně v části select zvolíme informace jež nás zajímají. Bude to subjekt neboli typ ovoce a objekt neboli jeho barva. Výsledek by pak byl následující.

Subjekt	Objekt
<http://example.com/fruits/orange>	Orange
<http://example.com/fruits/banana>	Yellow

Tabulka 3.1: Výsledek dotazu

Kapitola 4

Koncept zpracování problému

Po části výběru vhodného nástroje je nutné zamyšlení jak ho využít, neboli se zamyslet nad samotným konceptem zobrazení dat. Nejprve však musíme data nějakým způsobem získat. Následující část se věnuje obecnému řešení těchto dvou problémů.

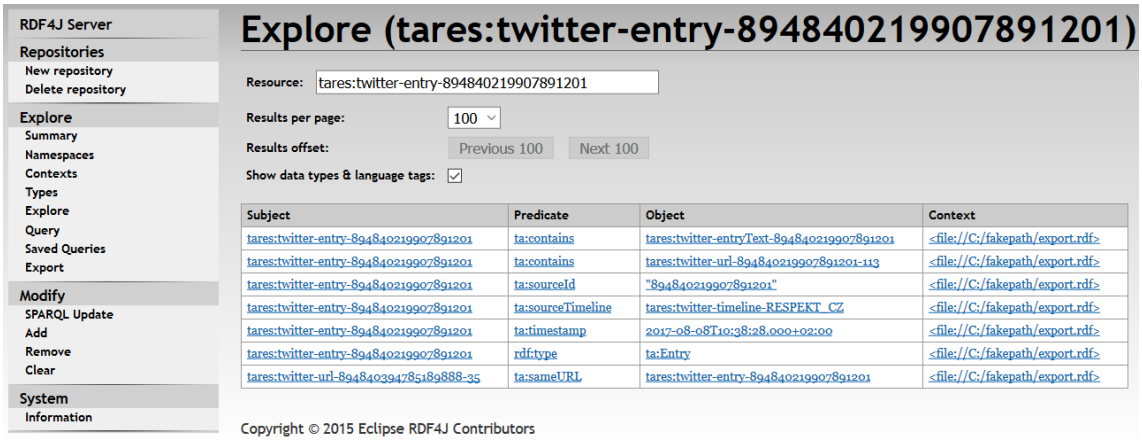
4.1 Zajištění dat ze sociálních sítí

Zajištěním dat ze sociálních sítí se zabývá související projekt TimelineAnalyzer[1], který implementuje získávání dat z různých zdrojů [2]. Výsledná získaná data jsou ukládána ve formátu RDF, kdy se využívá technologie sémantického webu. Společným uložištěm je framework RDF4J. Nad daty získanými jednotlivými softwarovými moduly od každého zdroje, kterým může být sociální síť, lokální profil a další, se provádějí jednotlivými analytickými nástroji operace, jejichž výsledkem jsou potencionálně zajímavé vztahy mezi daty, které jsou následně opět ukládány do společného uložiště. Ve výsledku, tedy dostáváme datový model 6.6, jež je předmětem vizualizace.

4.2 Získání dat pro zobrazení

Jak již bylo zmíněno, je nutné si říct, odkud a jak se data budou brát neboli co je datovým zdrojem. Odpovědí na tuto otázku je RDF4J framework [9] pro práci s RDF daty, který je zvolen právě z důvodu , že data jsou uložena ve formátu RDF. Konkrétně je pak zvolen RDF4J Server jako databázový server a kromě toho je použit RDF4J Workbench jako webové uživatelské rozhraní pro práci s databází [8]. Konkrétní úlohou RDF4J Serveru je poskytnout HTTP přístup k RDF4J repozitářům, tedy vystavit je jako SPARQL endpointy. S RDF4J serverem pak komunikuje klientská aplikace rdfj-client.js. Pro úspěšné zprovoznění RDF4J Serveru a RDF4J Workbenche je nutné využít nějakého Java Servlet Containeru. Pro tuto roli byl vybrán osvědčený a volně šiřitelný projekt Apache Tomcat verze Apache Tomcat/8.0.47. Na příkladu 4.1 pak můžeme vidět ukázku RDF dat pro konkrétní příspěvek.

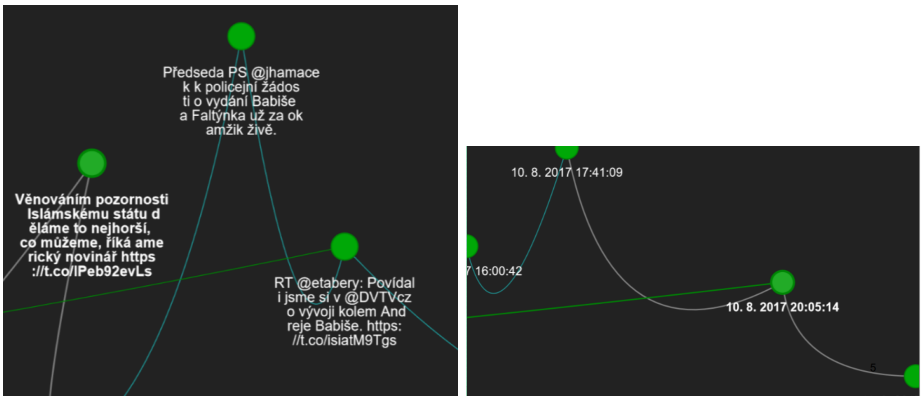
[1] Projekt TimelineAnalyzer https://github.com/nesfit/timeline-analyzer



Obrázek 4.1: Workbench explore příklad

4.3 Zobrazení dat

Jednou z možností je zobrazovat data přímo v zobrazovací ploše, což ale přináší řadu nepraktických a nežádaných vlastností. Při zobrazení v ploše může například dojít k překrytí dat, nepřehlednosti a celkovému nedůstojnému vzhledu zobrazení. Dalším problémem může být také negativní vliv na výkonnost, tedy rychlost zobrazení dat a taky rychlost snímkování při interaktivní manipulaci s daty. Jako lepší možnost se jeví reprezentovat příspěvky uzlovými prvky a hran mezi nimi, pomocí kterých lze vyjádřit návaznost a vztahy mezi daty. Samotná data příspěvku pak lze zobrazit v příslušném prostoru s dostatečnou intuitivitou a uživatelsky přátelskou jednoduchostí. Uzel představující příspěvek, je spojen s datem a po najetí na uzel se objeví náhled obsahu pro zobrazení při kliknutí. Na obrázcích 4.2 můžeme vidět příklad porovnání těchto přístupů.



Obrázek 4.2: Porovnání zobrazení

Samotné umístění uzlů je potom z důvodu uživatelsky přívětivého prostředí řízeno algoritmicky. Náhodné rozložení prvků sice poskytuje možnost využití funkce knihovny **Vis.js** pro manipulaci s více uzly najednou, ale takovéto uspořádání je neintuitivní a může působit matoucně. Překrývání prvků, promíchanost zdrojů a další negativa ovlivňují prostorovou orientaci, která je pro nás jeden z důležitých bodů pro kvalitní zobrazení velkého počtu dat. Dalším problémem, který může nastat při tomto typu zobrazení je negativní vliv na výkon-

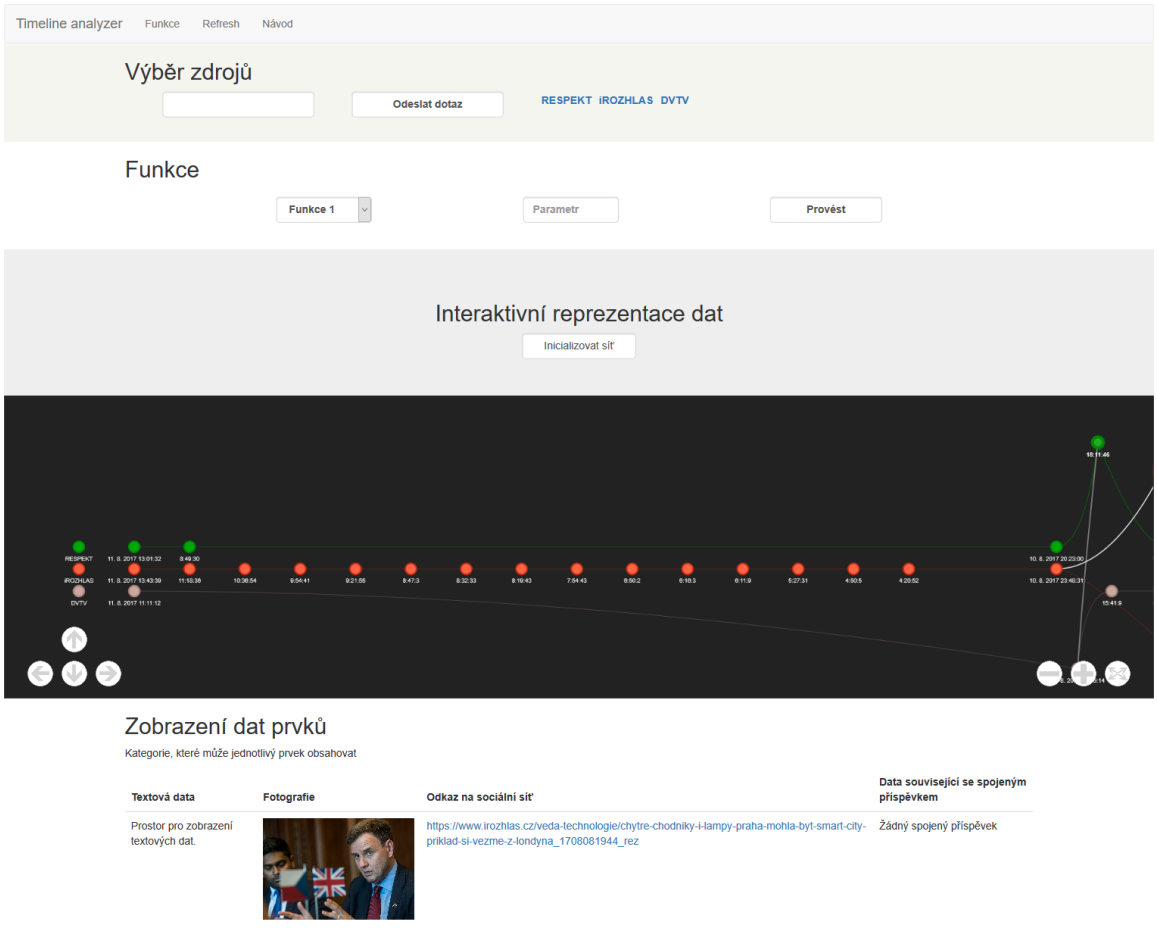
nost a plynulost zobrazení. Na druhou stranu zmíněné algoritmické zobrazení, není těmito problémy zatěžováno a jeví se tedy jako lepší volba. Prvky jsou pak seřazeny podle data a vzhledově uzpůsobeny, podle zdroje ke kterému náleží. Interaktivní zobrazovací plocha umožňuje uživateli s prvky manipulovat, přemisťovat je, přibližovat a oddalovat zobrazení a pohybovat se v ploše. Takovýto způsob zobrazení nabízí kvalitní a intuitivní možnost pohledu na data, jež chceme sledovat.

Další výhodou tohoto řešení je, že hrany mezi uzly mohou simulovat jednak jejich vzájemnou zdrojovou návaznost, ale hlavně mohou představovat vztahy a vzájemné odkazy mezi příspěvky. Dle zvýrazněné hrany lze jednoduše a rychle poznat, které prvky jsou propojené a lze naznačit případné informace o vztahu.

Místo klasické sekvenční reprezentace dat na www tedy máme možnost celkového pohledu na propojenou síť, ve které lze rychle a efektivně určit hustotu dat v daných časových obdobích, vztahy a propojení mezi daty, míru aktivity jednotlivých zdrojů a celkový rozsah dat za určité časové období.

4.4 Schéma uživatelského rozhraní

V této části bylo úkolem vymyslet návrh rozhraní, který by poskytl možnost pracovat s datovou zobrazovací plochou v prostředí prohlížeče. Návrh rozhraní zahrnuje 5 hlavních částí. První je jednoduché uživatelské menu s možností přesměrování na stránku s manuálem. Další je sekce s výběrem počátečních zdrojů pro zobrazení, kde uživatel určí o jaký obsah má zájem. Následuje část, která nabízí různé funkce k výběru a následné provedení funkce nad zobrazovací plochou. Jedná se například o možnost posunutí kamery nad uzel určitého data. Poté je část pro práci se zobrazovací plochou, rozdělená na 2 sekce, kdy v první je možnost inicializace plochy a jejího obnovení a v druhé je samotná zobrazovací plocha s uzly a hranami reprezentujícími příspěvky a jejich vztahy.V poslední části je pak prostor k zobrazení dat, kdy můžeme mít textová data, fotografie spojené s příspěvkem, odkaz na příspěvek z určité stránky nebo sociální sítě a data spojená se vztahem k jinému příspěvku.



Obrázek 4.3: Schéma GUI

Kapitola 5

Použité technologie

V rámci aplikace je potřeba získávat data od serveru a náležitě je zpracovat. Je tedy nutné využití technologií, které jsou schopné navázat komunikaci mezi klientskou a serverovou částí, požádat o data serveru a v rámci klientské aplikace je vhodně zpracovat. Následující část se věnuje těmto technologiím.

5.1 REST api

REST neboli representational state transfer je architektura rozhraní, navržená pro distribuované prostředí [10]. Rozhraní REST umožňuje jednotný a snadný přístup k webovým zdrojům použitím uniformních a predefinovaných souborů bezestavových operací. Zdroje mohou být data nebo stavy aplikace. Všechny zdroje mají svůj identifikátor URI. REST pak definuje čtyři základní přístupy k těmto zdrojům.

Tyto metody jsou známy pod označením CRUD neboli create pro vytvoření dat, retrieve pro získání dat, update pro úpravu dat a delete pro odstranění dat. Metody jsou implementovány pomocí příslušných metod HTTP protokolu [16].

GET Metoda pro získání zdroje. Požadavek na získání dat může vypadat následovně.

GET /informace/uzivatel.formát HOST: www.priklad.cz/

Data mohou být vrácena ve více formátech. Například XML,JSON,RSS. V našem případě používáme formát JSON.

POST Metoda pro vytvoření zdroje. V moment zaslání požadavku není známá URI zdroje. Je teprve vytvářen. Požadavek na vytvoření dat může vypadat následovně.

POST /questions HTTP/1.1 Host: www.example.com/

Zdroj je vytvořen a je navráceno jeho URL.

DELETE Metoda pro smazání zdroje. Požadavek na smazání dat může vypadat následovně.

DELETE /informace/uzivatel Host: www.priklad.cz/

Někdy může dojít k problému z důvodu, že formuláře jsou omezeny pouze na metody GET a POST. Taková situace se dá řešit například speciální formou metody POST, ve které je specifikováno, že se má ve skutečnosti použít metoda DELETE.

PUT Metoda pro úpravu zdroje. Požadavek na úpravu dat může vypadat následovně.

PUT /informace/novyuzivatel HOST: www.priklad.cz/

Metoda úpravy je podobná metodě vytvoření neboli POST. Rozdíl je takový, že nyní voláme konkrétní URI jistého zdroje, který chceme změnit. V těle pak předáme novou hodnotu. Na rozdíl od POST je u úprav zdroje jeho URI už známá, takže ji můžeme zadat. Metoda PUT může sloužit i k vytvoření zdroje.

PUT /informace/existujiciuzivatel HOST: www.priklad.cz/

5.2 JavaScript promise

Promise je zástupce pro hodnotu ne nutně známou, když je promise vytvořeno [5]. Dovoluje spojovat manipulátory s úspěchem nebo neúspěchem asynchronní akce. Toto dovoluje asynchronním metodám vrátit hodnoty jako synchronní metody. Místo okamžitého navrácení výsledné hodnoty, asynchronní metoda navrátí slib (promise), že hodnota bude vrácena někdy v budoucnu.

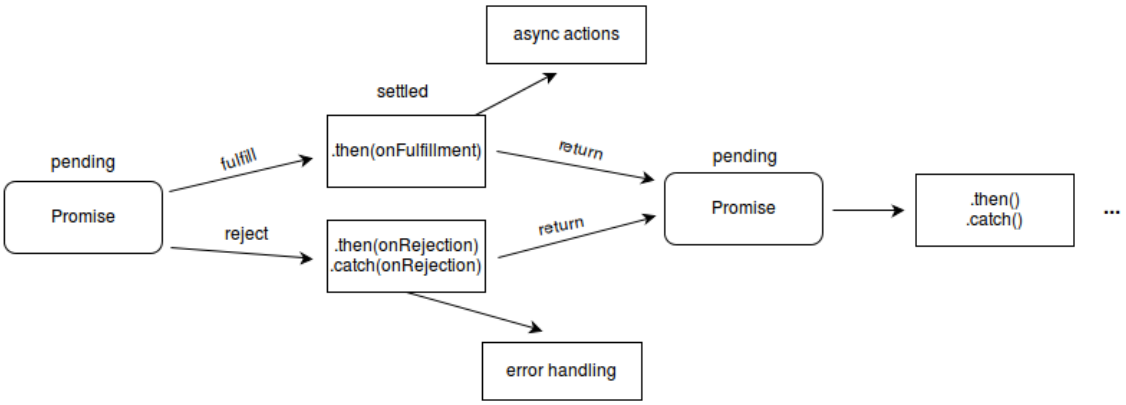
Promise má následující stavy.

Pending iniciální stav, nevyplněno ani nezamítnuto

Fulfilled znamená, že operace proběhla úspěšně

Rejected znamená, že operace proběhla neúspěšně

Pending promise se může stát fulfilled s hodnotou, nebo rejected s důvodem. Pokud nastane jedna z těchto možností, jsou zavolány spojené manipulátory seřazené metodou **then**. Metody **Promise.prototype.then()** a **Promise.prototype.catch()** mohou vrátit promise a mohou být řetězeny.



Obrázek 5.1: Příklad řetězení

Metody promise

Promise.all(iterable) Vrátí promise, které se buď naplní, pokud jsou naplněna všechna promise v posloupnosti nebo nenaplní pokud je alespoň jedno z promise v zadané posloupnosti nenaplněno. Pokud se vrácené promise naplní, pak je naplněno polem

hodnot vyplněných promise ve stejném pořadí jako v posloupnosti. Pokud je nenaplněno, pak nese důvod prvního odmítnutého promise z posloupnosti. Metoda je vhodná pro agregaci výsledků více promise.

Promise.race(iterable) Vrací promise, které se naplní nebo zamítne hned, jak je naplněno nebo zamítnuto, některé z promise posloupnosti. Naplněno nebo zamítnuto s hodnotou onoho promise.

Promise.reject(reason) Vrací promise objekt, který je zamítnut s daným důvodem.

Promise.resolve(value) Vrací promise objekt, který je vyřešen s danou hodnotou. Pokud má hodnota **Promise.prototype.then()** metodu, navrácené promise bude následovat tuto metodu.

Promise prototype

Promise.prototype.catch(onRejected) Připojí manipulátor zamítnutí k promise a vrátí nové promise řešící návratovou hodnotu callback funkce pokud je zavolána nebo původní hodnotu pokud je naplněno.

Promise.prototype.then(onFulfilled, onRejected) Připojí manipulátory naplnění a zamítnutí k promise a vrátí nové promise řešící návratovou hodnotu volaného manipulátoru nebo originální hodnotu z naplnění.

Promise.prototype.finally(onFinally) Připojí manipulátor k promise a vrátí nové promise, které je naplněno, když je původní promise naplněno. Manipulátor je volán, když je promise vyřízeno, ať naplněno nebo zamítnuto.

Příklad promise z programové části práce

```
//funkce zasilajici query na server. Vraci promise, ktere pri uspechu vraci
data ze serveru.
RDFClient.prototype.sendQuery = function(query) {
       var client = this;
       return new Promise(function(resolve, reject) {
              $.ajax({
                  url: client.url + '/repositories/' + client.repo,
                  type: 'POST',
                  data: query,
                  contentType: 'application/sparql-query',
                  dataType: 'json',
                  headrs: {
                      Accept: 'application/sparql-results+json'
                  },
                  async: true,
                  success: function(data) {
                      resolve(data);
                  },
                  error: reject
              });
```

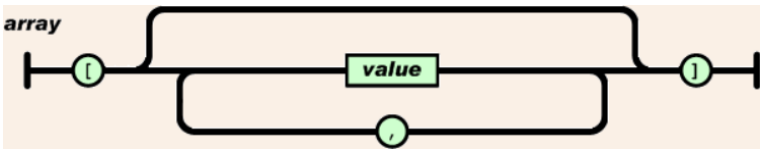
```
        });
};

//funkce pro ziskani datovych objektu, vraci promise, ktere pri naplneni
vraci data zpracovana funkci parseResponseObjects. Takoveto vyuziti promise
nam umoznuje asynchronni praci a tedy vyrazne urychleni behu aplikace.
RDFClient.prototype.getObjectsWhere = function(where) {
        var client = this;
        var w = '?s ?p ?o';
        if (where)
                w += ' . ' + where;
        var query = this.getPrefixes() + 'SELECT ?s ?p ?o WHERE {' + w + '}';

        return new Promise(function(resolve, reject) {
                var p = client.sendQuery(query);
                p.then(function(data) {
                        resolve(client.parseResponseObjects(data));
                }).catch(function(reason) {
                        reject(reason);
                });
        });
};
```

5.3 JSON

JavaScript Object Notation je odlehčená syntax pro ukládání a výměnu dat [15]. Json je textový na jazyce nezávislý formát psaný s JavaScriptovou objektovou notací. Je založen na dvou základních strukturách. První je kolekce párů název/hodnota. V jiných jazycích realizováno jako objekt, struktura, slovník, hash tabulka nebo asociativní pole. Další je seřazený seznam hodnot.Ten je jinde většinou realizován jako pole, vektor, list nebo posloupnost. Jedná se o univerzální datové struktury, které jsou v nějaké formě podporovány ve většině jazyků.

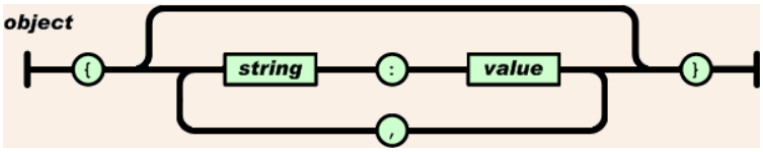
Následující konstrukce popisují jednotlivé struktury [12].

Pole Seřazená kolekce hodnot.



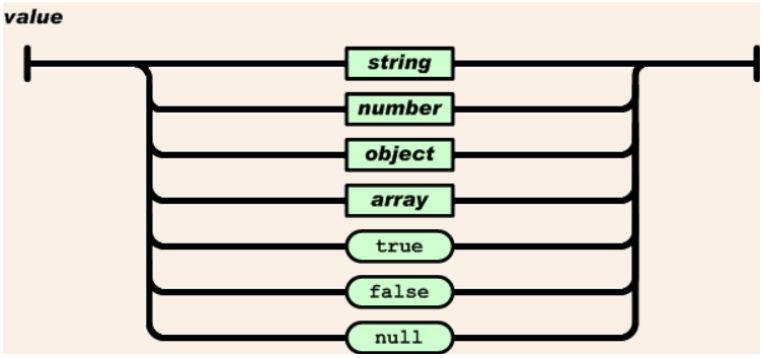
Obrázek 5.2: Pole

Objekt Neuspořádaná množina dvojic název a hodnota.



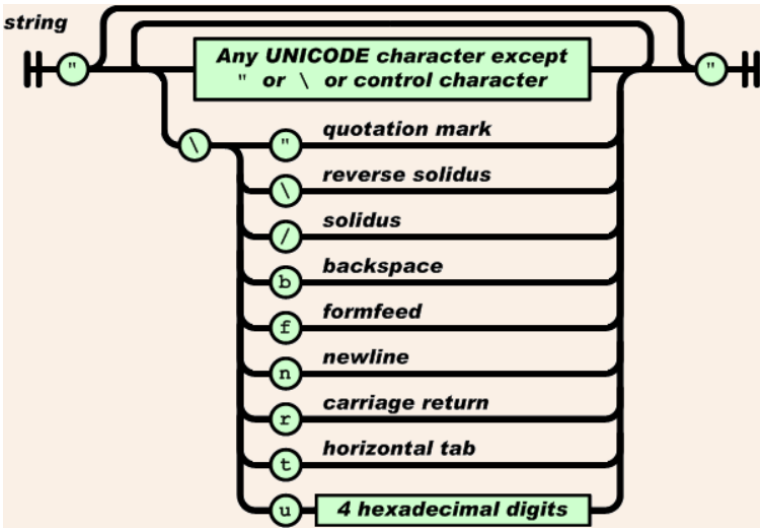
Obrázek 5.3: Pole

Hodnota Řetězec uzavřený do dvojitých uvozovek.



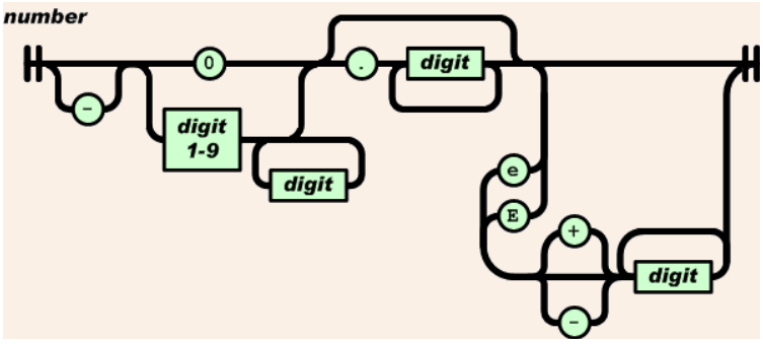
Obrázek 5.4: Pole

String Řetězcem je nula nebo více znaků kódování Unicode, uzavřených do dvojitých uvozovek a využívající únikových sekvencí s použitím zpětného lomítka.



Obrázek 5.5: Pole

Číslo Podobné číslům z C s vyjímkou nepoužívání hexadecimálního zápisu.



Obrázek 5.6: Pole

5.4 RDF4J Framework

RDF4J Framework je open source JAVA framework pro práci s RDF daty [9]. To zahrnuje zpracování, ukládání, odvozování a dotazování nad těmito daty. Nabízí snadno použitelné API, které může být připojeno na všechny RDF ukládací řešení. Umožňuje propojit koncové body SPARQL komunikace a vytvářet aplikace využívající možnosti propojených dat a sémantického webu.

RDF4J nabízí 2 databáze, in-memory store a native store, k tomu mnoho dalších řešení třetích stran. Framework umožňuje použití, velkého počtu vývojářských nástrojů k využití výhod RDF a přidružených standardů. RDF4J plně podporuje SPARQL 1.1 dotazy a upravující řeč pro vyjadřující dotazy. Dále nabízí transparentní přístup ke vzdáleným RDF uložištím používající stejné API jako pro lokální přístup. RDF4J podporuje všechny hlavní RDF souborové formáty jako například, XML, Turtle, N-Triples, N-Quads, JSON-LDm Trig a Trix.

5.5 Apache Tomcat

Apache Tomcat je volně šiřitelná softwarová implementace JAVA Servlet (JAVA program rozšiřující vlastnosti serveru), JavaServer pages, Java Expression Language a Java WebSocket tenchnologií [13]. Jinými slovy se jedná o open source webový server a JAVA Servlet kontejner.

Apache Tomcat software je vyvíjen v otevřeném a zůčastnitelném prostředí a veden pod Apache license verze 2. Apache Tomcat je využíván početnými rozsáhlými webovými aplikacemi s širokou škálou organizací.

5.6 JavaScript prototype

Prototypy jsou mechanismus pomocí, kterého JavaScriptové objekty dědí vlastnosti jeden od druhého [4]. Fungují jinak než v klasickém objektově orientovaném přístupu. JavaScript je často popisován jako prototypově založený jazyk, neboli každý objekt má prototypový objekt, který se chová, jako šablonový objekt od kterého se dědí metody a atributy. Tohle se často nazývá jako prototypový řetězec a vysvětluje, proč různé objekty mají přístup k metodám a atributům definovaných na jiných objektech.

Přesněji, atributy a metody jsou definovány v konstruktorových funkcích objektu, ne v jeho instancích.

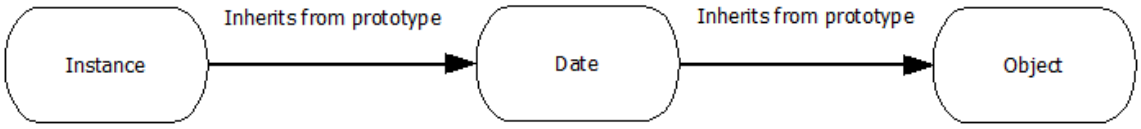
V klasickém Objektově Orientovaném Programování jsou definovány třídy. Ve chvíli kdy jsou vytvořeny instance objektu, všechny metody definované ve třídě jsou zkopírovány do instance. V JavaScriptu nejsou kopírovány, ale je vytvořeno propojení mezi instancí objektu a jejím prototypem. Atributy a metody jsou k dispozici při průchodu řetězcem prototypů.

Tento princip lze dobře ukázat na jednoduchém příkladu. Berme v úvahu následující kód.

```
function Date(first,last,microsecond,colour)
{
    this.firstName = first;
    this.lastName = last;
    this.micro = microsecond;
    this.eyeColor = colour;
}
var instance = new Date("Some", "Example", 50, "red");
var check = instance.valueOf();
```

Výpis 5.1: Ukázka vytvoření instance objektu

V první části můžeme vidět konstruktorovou funkci *Date*. Následovně pomocí ní vytvoříme instanci *instance*. Instance pak má zadané členské proměnné, jako *firstName*, *lastName* a tak dále, ale taky například *valueOf()*, která není definovaná v *Date* ale v prototypovém objektu *Date* , kterým je Object. Zde můžeme vidět princip prototypového řetězce.



Obrázek 5.7: Prototypový řetězec

Při vykonávání kódu, kdy se volá metoda, jež je definovaná ne v prototypu *Date*, ale v prototypu *Object* se stane následující. Prohlížeč zkontroluje zda má objekt *instance* k dispozici metodu *valueOf()*. Nemá, takže se podívá, zda ji má k dispozici prototypový objekt *Date*. Také nemá, takže se prohlížeč podívá, zda ji má k dispozici prototypový objekt *Object*. Ten ji definovanou má, takže ji lze zavolat.

Při prototypovém řetězci je třeba rozlišovat, kdy se metody skutečně dědí a kdy ne. Například vezmeme-li *Object*, tak budou děditelné pouze metody specifikované jako *Object.prototype.method*. Metody specifikované jako *Object.method* nebudou děditelné a tedy použitelné níže v prototypovém řetězci. Tyto metody budou přístupny pouze na konstruktoru *Object* samotném.

Další z možných využití tohoto konceptu je například definování metod pro prototyp. Pokud definujeme metodu pro prototyp *Date.prototype.newmethod*, budou ji pak schopny využívat všechny instance vytvořené kostruktorem s protypem *Date*.

5.7 Programování s třídami v JavaScriptu

V této části bych uvedl koncept programování s třídami užívaný v jazyce JavaScript, neboť je součástí práce.

K napodobení konceptu třídy je sice více způsobů, nicméně v práci použitý způsob je ten patřící mezi adekvátní, takže bude uveden k vysvětlení problému.

K vytvoření instancí třídy je použito konstruktoru a operátoru new. Konstruktorem přitom chápejme funkci, která je použita jako konstruktor [17].

```
this.client = new RDFClient(this.url, this.repo, this.ns);
```

Výpis 5.2: Vytvoření instance

Jak vidíme na ukázce 5.2 k vytvoření instance se zavolá operátor new s kontruktorem dané třídy. JavaScript v tuto chvíli udělá 4 kroky. Nejdříve vytvoří nový objekt. Poté nastaví constructor atribut objektu na RDFClient. Dále nastaví, aby objekt ukazoval na RDFClient.prototype.To znamená, že jakékoliv změny za pomocí RDFClient.prototype se promítnou v instanci. Například 5.3.

```
RDFClient.prototype.value = 10;
alert(this.client.value); // zobrazi deset
```

Výpis 5.3: Příklad

Na závěr se zavolá konstruktorová funkce, která nastaví atributy, dle parametrů v konstruktoru.

```
var RDFClient = function(serverUrl, repositoryName, namespaces) {
        this.url = serverUrl;
        this.repo = repositoryName;
        this.ns = namespaces;
        this.defaultNS = 'rdf';
};
```

Výpis 5.4: Konstruktorová funkce

Tento přístup nám pak umožní definovat instanční metody třídy RDFClient pro manipulaci s daty a další libovolné úkony.

Kapitola 6

Návrh softwarového řešení

Softwarové řešení našeho problému je řešeno, jak již bylo naznačeno pomocí jazyku JavaScript a knihovny vis.js. Jako zdroj dat je pak vybrán framework RDF4J, který využívá jako svůj JAVA Servlet container Apache Tomcat. Dalšími nedílnými součástmi jsou knihovny Bootstrap, pro stylizaci grafického uživatelského rozhraní a knihovna Jquerry pro rozličné úkoly.

6.1 Získání dat

Data příspěvků jsou získávána přes dotazy formulované jazykem SPARQL na RDF databázi. Dotazy jsou posílány technologií AJAX pro asynchronní zpracování obsahu webových stránek. Pro možnost využití asynchronního zpracování a tedy urychlení získání dat ze serverové databáze je použita technologie JavaScript promise. Při obdržování dat, týkajících se příspěvků a vztahů, dochází k jejich uložení v příslušných datových strukturách. Po obdržení všech dat jednotlivých zdrojů, může dojít ke zpracování dat v příslušných funkcích a k jejich následnému zobrazení v grafické ploše.

6.2 Rozložení

Rozložení uzlů reprezentujících jednotlivé příspěvky je prováděno pomocí funkce, jež rozdělí daný prostor na sektory, tak aby každý den měl dostatečnou velikost pro udržení maximálního počtu příspěvků v daném dni. Tímto způsobem jsme schopni pro každý den vidět podíl zdrojů na příspěvcích tohoto dne. Zároveň nám rozdělení prostoru na sektory reprezentující jednotlivé dny umožňují srovnat příspěvky zdrojů tak, aby příspěvky daného dne začínali na stejné xové souřadnici pro každý zdroj. Pro každý zdroj je pak dána nová ypsilonová souřadnice. Tímto dosáhneme intuitivního rozložení jednotlivých prvků. Rozložení dnů je dáno od nejmenšího po největší, dle zásad časové osy.

6.3 Zobrazení v grafické ploše

Zobrazení uzlů v grafické ploše má na starosti knihovna **Vis.js**, které jsou předány datové struktury obsahující informace o rozložení, obsahu, propojení a designu uzlů. Tyto struktury jsou upravovány na základě dat přijatých z databáze. Poté co je vyvolána inicializace grafické plochy, je potřeba získat data dle vybraných zdrojů z databáze, následně vytvořit požadované rozložení a poté může dojít k předání struktur knihovně. Ve chvíli kdy dojde k

zobrazení uzlů, lze s prvky interaktivně manipulovat. Prvky lze přesouvat, lze zobrazit náhled dat, lze se různě posouvat a přibližovat v prostoru zobrazovací plochy. Tuto funkčnost opět zařizuje knihovna **Vis.js**. Pro každý zdroj je určena jiná barva, pro vzájemné odlišení zdrojů v ploše. Vztahy jsou realizovány funkcí, která pracuje se strukturou pro uložení odkazů mezi uzly a následně přidá údaje o nutnosti propojení určitých uzlů a zvýraznění takovéhoto vztahu. V ploše je také zobrazena jednoduchá legenda pro rychlou identifikaci zdrojů. Nad grafickou plochou jsou realizovány určité funkce, které například pomocí funkcí knihovny **Vis.js** umožňují kamerový pohyb v ploše. Uzly jsou na první pohled identifikovány, dle konkrétního data příspěvku. V případě, že je den počátečním dnem časového sektoru, má celé datum i se dnem, v případě, že není prvním dnem má pouze označení hodin, minut a sekund.

6.4 Zobrazení dat

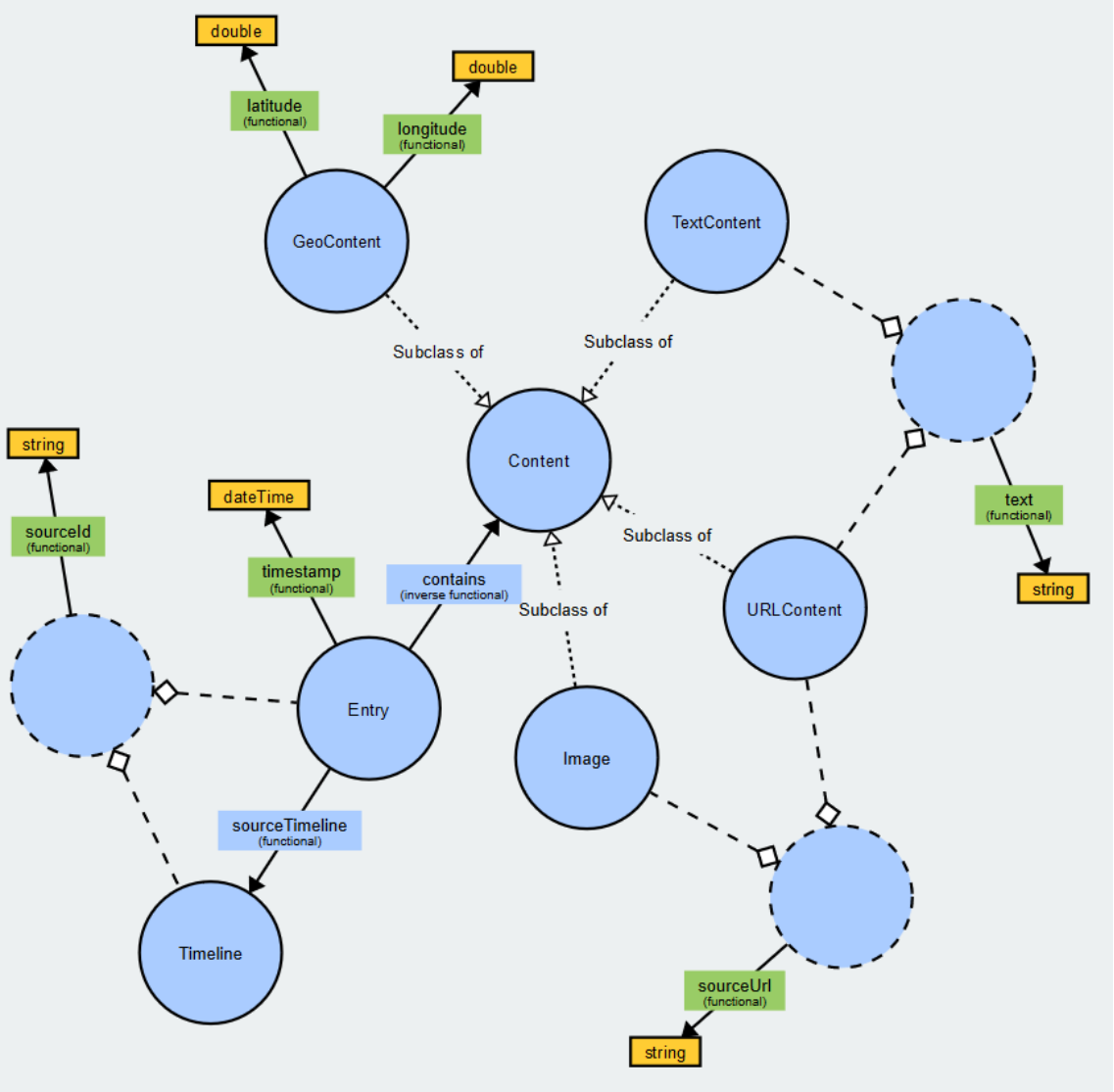
Zobrazení dat, je realizováno pomocí knihovní funkce, která vyvolává událost po kliknutí na uzel. Funkce informuje, o jaký uzel se jedná a na základě této informace, jsou od serveru získána data spojená s daným příspěvkem. Následně se zobrazí příslušná data v zobrazovacím okně. Zobrazovací okno obsahuje vlastní část pro každý typ dat spojených s příspěvkem. V jednotlivých částech jsme tedy schopni se rychle a jednoduše zorientovat a analyzovat zobrazená data.

6.5 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je vytvořeno pomocí technologií CSS, HTML5, knihovny Bootstrap a jazyku JavaScript. V první části je vytvořeno jednoduché menu. V druhé pak formulář pro výběr zdrojů, z kterého lze aktivně odebírat a přidávat prvky. Na základě vybraných prvků se pak zobrazují zdroje v grafické ploše. Následně byl po fázi testování zahrnut i formulář s výběrem časového omezení příspěvků. V další části je prostor s možností zavolání funkce s parametry nad datovou sítí. Jednou z funkcí je přiblížení kamery na uzel s určitým datem. Při opětovném volání se procházejí uzly stejného data. Dalším prvkem je samotná grafická plocha, ve které uživatel manipuluje se zobrazenými daty v podobě uzlů. Poslední částí je prostor, ve kterém se zobrazují data. Je vytvořen formou boxů, kdy každý představuje určitý typ dat. Celkově je grafické uživatelské rozhraní směřováno k účelnosti a jednoduchosti. Vzhled se drží jednoduchého barevného schématu a počet možností manipulace s prvky rozhraní nepřesahuje zbytečnou míru.

6.6 Datový model

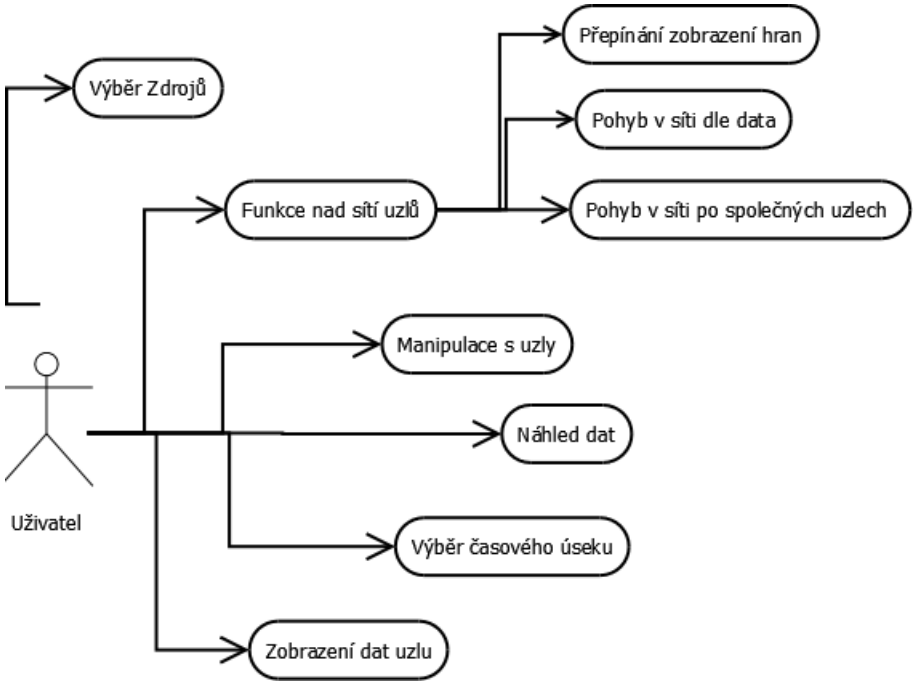
V aplikaci se pracuje s daty, které jsou reprezentovány datovým modelem vycházejícím z webové ontologie OWL. Třída Timeline představuje zdroj obsahující jednotlivé příspěvky. Přípěvky pak zastupuje třída Entry. Vztah mezi příspěvkem a zdrojem znázorňuje predikát sourceTimeline. Jak příspěvek, tak zdroj mají určité id. Každý příspěvek pak obsahuje časovou značku timestamp a obsah content. Vztah mezi obsahem a příspěvkem je reprezentován predikátem contains. Obsah pak může nabývat různých typů, které jsou znázorněny třídami URLContent, TextContent a Image. Podtřídy TextContent a URLContent obsahují určitý text. Třída URLContent ještě navíc spolu s třídou obsahují URL adresu sourceURL. Následující diagram popisuje tento datový model.



Obrázek 6.1: RDF data model diagram

6.7 Use case diagram

Tento typ diagramu reprezentuje, jak se systém chová z pohledu uživatele. Středem je role, v našem případě uživatel, který má možnosti (use cases) interakce se systémem. Následující diagram znázorňuje možnosti uživatele, při jeho práci s aplikací.

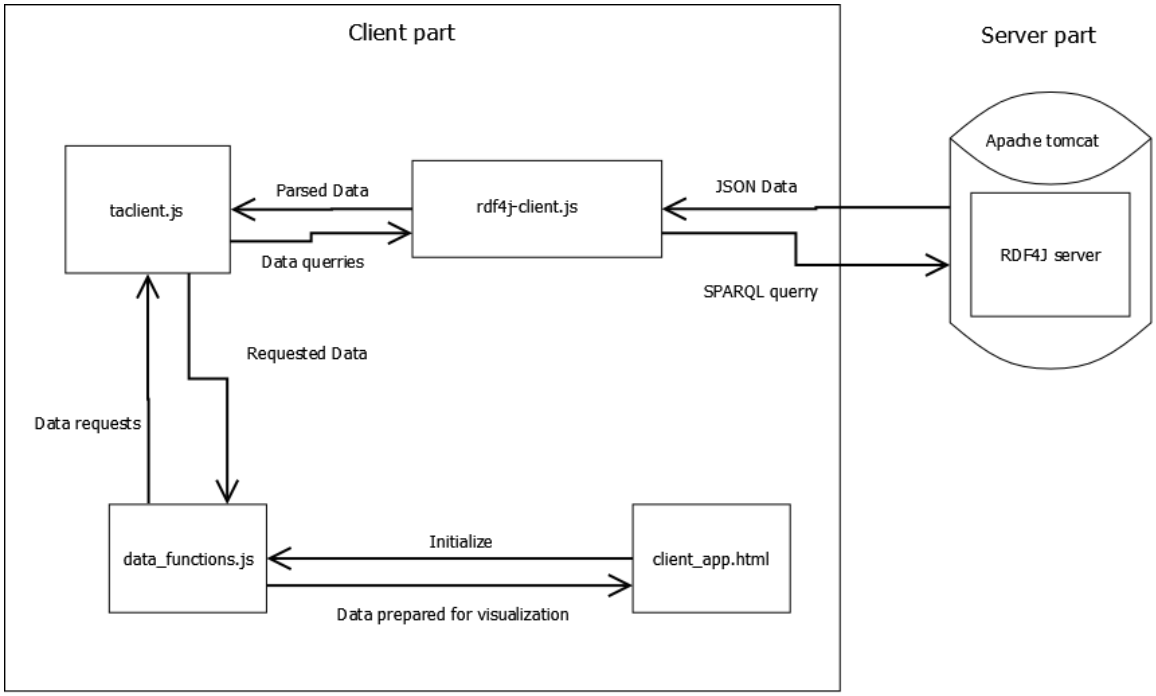


Obrázek 6.2: Use case

6.8 Struktura aplikace

Aplikace má následující strukturu, tvořenou klientskou částí a částí serveru. Klientská část komunikuje s webovým serverem Apache, který předává HTTP žádosti klienta Servlet containeru [18] Apache Tomcat [3]. Ten pak nechává odpovídat RDF4J Server. V HTTP žádostech jsou obsaženy SPARQL dotazy, které jsou RDF4J serverem zpracovány a následně odpovězeny. Žádosti jsou tvořeny pomocí AJAX metod, které zasílají serveru RESTFULL dotazy. Databáze odpovídá daty ve formátu JSON, která jsou pak zpracovávána klientskou aplikací.

Konkrétněji tedy nastává následující. Komunikace klienta se serverem je vyvolána uživatelem při interakci s grafickým uživatelským rozhraním v prohlížeči. Nejprve client_app.html po stisknutí tlačítka pro inicializaci uživatelem zavolá funkci knihovny data_functions.js. Funkce pak vyvolá proces žádostí o určitá data. Tyto žádosti jsou předány knihovně tac-lient.js, kde jsou na základě typu žádosti formulovány SPARQL dotazy. Dotazy jsou pak předány knihovně rdf4j-client.js, která dotazy odešle v HTTP žádosti pomocí technologie AJAX. Na straně serveru jsou dotazy vyhodnoceny a zpět jsou zaslána Data ve formátu JSON. Data jsou v rdf4j-client.js zpracována a v podobě datových objektů předána tac-lient.js. Ten je pak vrací jako odpověď na žádost data_functions.js, kde se provádí zpracování dat do podoby využitelné pro vizualizaci. V konečné části jsou pak data předána client_app.html, kde se provádí jejich zobrazení.



Obrázek 6.3: Struktura aplikace

Kapitola 7

Implementační detaily

Úkolem této kapitoly bude postupně rozebrat jednotlivé části implementace projektu.

7.1 Vytvoření RDF klienta a komunikace se serverem

Jedním ze základních prvků aplikace je získání samotných dat příspěvků z databáze RDF. Pro komunikaci se serverem je nutné vytvořit RDF clienta. Iniciace funkcí k vytvoření clienta proběhne následujícím kódem 7.1, kdy nejdříve je vytvořen Timeline Analyzer Client pro vytváření dotazů a práci s RDF clientem. Client musí znát serverovou URL, jméno repositáře a v případě RDF clienta i použité RDF jmenné prostory.

```
var TA = new TAClient("http://localhost:8080/rdf4j-server", "test");
```

Výpis 7.1: Vytvoření TAC a následně RDF clienta

Po vytvoření clienta, je možné zahájit komunikaci se serverem. Tato úloha je vykonávána pomocí technologie JQuery AJAX, kdy užitím AJAX metod pro komunikaci se serverem, zasíláme dotazy na server. Server pak odpovídá daty v podobě formátu JSON. Na ukázce 7.2 můžeme vidět funkci, která zasílá požadavek typu POST na určité URL s daty, které obsahují dotaz, jež bude na serveru vykonán. Očekávaná návratová data jsou formátu JSON.

```
RDFClient.prototype.sendQuery = function(query) {
        var client = this;
        return new Promise(function(resolve, reject) {
                $.ajax({
                    url: client.url + '/repositories/' + client.repo,
                    type: 'POST',
                    data: query,
                    contentType: 'application/sparql-query',
                    dataType: 'json',
                    headrs: {
                        Accept: 'application/sparql-results+json'
                    },
                    async: true,
                    success: function(data) {
                        resolve(data);
                    },
```

```
                error: reject
            });
        });
};
```

Výpis 7.2: Ukázka funkce vykonávající dotaz na serveru

7.1.1 Timeline Analyzer Client

Knihovna poskytující funkce komunikující s RDF4J clientem a definující dotazy na RDF serverovou databázi. Prostředník mezi knihovnou *data_functions*, která žádá data a RDF4J clientem, který je získává od serveru.

Počátečními úkony po samotném vytvoření clienta je například zjištění zdrojů webových příspěvků uložených v databázi, z kterých může uživatel následně vybírat. Tyto zdroje jsou získány metodou *getTimelines* .

Metoda předá data potřebná k vytvoření dotazu RDF4J clientovi, který v metodě *getObjectsWhere* dotvoří dotaz, nechá ho vykonat na serveru a pomocí další metody zpracuje vrácená data.

Další důležitou metodou je metoda *DateLimits* pro zjištění maximálního časového rozsahu dat uložených na serveru. Jinými slovy je potřeba zjistit datum prvního a posledního příspěvku z důvodu následného určování omezení výběru příspěvků dle času. Metoda vytvoří dotaz k odeslání na server a předá ho k vykonání metodě RDF4j clienta *getDateBounds*. Ta po vykonání dotazu a obdržení odpovědi od serveru předá zpracovaná data.

Následovně lze tedy použít metodu *setDateSpan*, která inicializuje hodnoty rozmezí časového úseku pro sběr dat z databáze.

TA Client má 4 klíčové metody pro získávání dat. Jejich důležitost spočívá ve zformování dotazu, který je předán RDF4J clientovy a následně vykonán na serveru.

TAClient.prototype.getEntries Metoda pro získání příspěvků daného zdroje od serveru. Spočívá ve vytvoření dotazu a jeho předání metodě RDF4j Clienta *getObjectArrayWhere*, která vykoná dotaz na serveru a předá navrácená data ke zpracování metodě *parseResponseObjectsToArray*. Výsledkem jsou data o příspěvcích použitelná k dalšímu využití.

```
TAClient.prototype.getEntries = function (timelineUri, src) {

        var filter = '';
        if (this.dateStart && this.dateEnd) {
                filter = ' FILTER (?time >= "' + this.dateStart.toISOString() +
        '"^^xsd:dateTime && ?time <= "' + this.dateEnd.toISOString() +
        '"^^xsd:dateTime)';
        }

        return this.client.getObjectArrayWhere('?s rdf:type ta:Entry .
    ?s ta:timestamp ?time . ?s ta:sourceTimeline <' + timelineUri + '>' +
    filter, 'ASC(?time)', src);
};
```

Výpis 7.3: Metoda getEntries

Na ukázce 7.3 můžeme vidět využití zmíněných okrajových časových hodnot *dateStart* a dateEnd pro ohraničení úseku sběru dat. Hodnoty jsou vloženy do filtru, který ve SPARQL query slouží jako omezující podmínka. Díky omezení ve filtru jsme schopni vybrat pouze příspěvky námi definovaného časového úseku. Dále můžeme vidět atributy, které nás budou zajímat. V zásadě dotaz říká, že nás zajímá každý subjekt, který obsahuje predikát rdf:type a objekt ta:Entry, dále predikát ta:timestamp s filtrem omezenou hodnotu objektu reprezentovanou proměnou ?time a predikát ta:sourceTimeline s hodnotou subjektu reprezentovanou proměnnou timelineUri, která říká, o který zdroj se jedná.

TAClient.prototype.loadEntryContents Metoda pro získání obsahu příspěvku. Dle dříve získaných dat o příspěvku, konkrétně podle URI, které jsou spojené s objekty uchovávajícími již konkrétní data o příspěvku, se volá metoda RDF4j clienta *getObject*, která vrací jednotlivé objekty pomocí metody *getObjectsWhere*. Tato metoda vykoná dotaz na serveru a následně nechá data zpracovat metodou *parseResponseObjects*. Výsledkem je pole objektů obsahujících data o konkrétním příspěvku.

TAClient.prototype.loadLinks Metoda pro získání dat spojených se vztahy mezi příspěvky. Využívá k tomu metodu RDF4J clienta *queryObjects*, které předá připravený dotaz. Metoda následně dotaz vykoná na serveru a po vrácení dat je předá ke zpracování metodě *bindingsToArray*.

```
var q = "SELECT ?p ?o ?label ?dtime ?text ?src ?srclabel"
            + " WHERE {<" + uri + "> ?p ?o . ?p rdfs:subPropertyOf
    ta:contextLink ."
            + " ?p rdfs:label ?label ."
            + " ?o ta:timestamp ?dtime ."
            + " ?o ta:contains ?textc . ?textc rdf:type ta:TextContent .
    ?textc ta:text ?text ."
            + " ?o ta:sourceTimeline ?src . ?src rdfs:label ?srclabel}";
    return this.client.queryObjects(q,finished);
```

Výpis 7.4: Metoda loadLinks

Na ukázce 7.4 můžeme vidět podobu celého query, jež se posílá na server. V trojici subjekt predikát objekt, má subjekt URI hodnotu daného příspěvku, pro který chceme data získat. Predikát *p* pak splňuje podmínku bytí subjektu pro predikát *rdfs:subPropertyOf* a objekt *ta:contextLink*. Takovýto predikát je pak *ta:sameURL*. V takovémto případě je pak objektem *o*, URI příspěvku, který je s původním příspěvkem spojený. Následující částí dotazu, již definují, které data nás budou zajímat a jak je získat.

TAClient.prototype.AllLinks Metoda pro získání dat týkajících se vztahů pro všechny příspěvky. Používá metodu RDF4J clienta *getAllLinks*, které předá dotaz, jež je následovně vykonán metodou na serveru. Po obdržení dat jsou data bez zpracování předána zpět metodě AllLinks.

```
var q = "SELECT ?sourceEntry ?destinationEntry"
            + " WHERE {"
```

```
                + " ?s ta:sameURL ?sourceEntry ."
                + " ?destinationEntry ta:contains ?s}";

        return this.client.getAllLinks(q);
```

Výpis 7.5: Metoda AllLinks

Na ukázce 7.5 můžeme vidět dotaz, jež se posílá na server. Zajímají nás vztahy příspěvků, neboli pro každý příspěvek *sourceEntry* potřebujeme příspěvky *destinationEntry* s kterými je spojený. Pro subjekt typu *ta:URLContent* máme predikát *ta:sameURL* určující společné URL, jehož objekt je *sourceEntry* . Takovýto subjekt je následně objektem subjektu *destinationEntry* s predikátem *contains* . Výsledkem takovéhoto dotazu pak jsou všechny URL zdrojových příspěvků a příspěvků s kterými mají společná data.

7.1.2 RDF4J Client

Klientská knihovna pro komunikaci s RDF serverem. Kromě přímé komunikace a zasílání dotazů na server zahrnuje i funkce pro předzpracování dat získaných od serveru. Prvním krokem RDF4J Clienta je jeho vytvoření, které je zahájeno TA Clientem. Vytvoření je provedeno pomocí kombinace klíčového slova new a konstruktoru *RDFClient*.

Jak bylo již, dříve zmíněno a znázorněno, client obsahuje funkce pro komunikaci se serverem. Kromě této funkčnosti ale obsahuje další klíčové funkce a to takové, které zpracovávají data získaná serveru do přijatelné podoby.

Jednou z klíčových metod třídy RDFClient pak je *parseResponseObjects*, která zpracuje data obdržená od serveru a dá jim podobu objektů dle RDF schématu. Jinými slovy pro určité URI (subjekt), které považujeme za id objektu, přidá pojmenování (predikát) a hodnotu (objekt).

Pro každý subjekt je tedy vytvořen objekt s id, které je hodnotou subjektu. Tomuto objektu je pak přidán predikát s určitou hodnotou. Pro získání predikátu je využita metoda *this.getPropertyName*, která vrací název predikátu na základě predikátové URI z OWL ontologie Timeline Analyzer. Výsledkem funkce je pak objekt, který obsahuje predikáty k němu vztažené a jejich hodnoty. Důležité predikáty zde jsou například predikáty *text* a *sourceUrl*.

Další důležitou metodou pro zpracování dat je *parseResponseObjectsToArray*, která zpracuje data RDF serveru obsahující informace o příspěvcích. Vrací pole objektů, kdy id objektů je URI příspěvku. Obsahem objektů jsou predikáty s určitou hodnotou. Pro získání predikátů je opět použita metoda *this.getPropertyName*. Důležitý predikát je predikát *contains* obsahující URI objektů s daty příspěvku. Hodnoty tohoto predikátu jsou totiž využity při volání metody *loadEntryContents*, kdy získáváme samotná data spojená s příspěvkem. Implementace metody je podobná té předchozí s tím rozdílem, že se nevrací objekt, ale pole objektů a taky je možnost, že predikát může obsahovat pole hodnot.

Metoda, která se zabývá získáním dat o vztazích, se nazývá *queryObjects*. Metoda nejprve vykoná dotaz na serveru a po získání dat zavolá metodu *bindingsToArray*.

Pro každý soubor dat, jež server navrátí, se vytváří objekt v poli objektů, jež metoda vrací. Nové objekty zachovávají atributy získaných dat, ale jako hodnoty ponechávají pouze hodnotu serverových dat s pojmenováním *value*.

7.2 Vyžádání dat pro zobrazení sítě

V této části se budu věnovat získávání dat pro zobrazení sítě uzlů. V prvé řadě je nutné si uvědomit, že vzhledem k objemu dat, které od serveru požadujeme, není vhodný přístup, který by vyžadoval načítání obsahu pro každý příspěvek již při zobrazení. Místo toho žádáme pouze o data spojená s informacemi o příspěvcích jednotlivých zdrojů a o data potřebná k zobrazení vztahů. Po obdržení těchto dat můžeme přejít k funkcím jež daná data zpracují a připraví je k předání knihovně vis.js, která je následovně zobrazí.

Část získávání dat začíná ve funkci *init*, kdy se nejdříve volá metoda třídy TA clienta *AllLinks*, která získá potřebná data pro pozdější zobrazení vztahů. Posléze je zavolána funkce *promise_processing*. V té se nejdříve pro každý uživatelem vybraný zdroj volá funkce TA clienta *getEntries*, která opět pomocí promise při úspěchu navrátí pole dat týkajících se jednotlivých příspěvků daného zdroje. Po získání těchto dat můžeme plnit strukturu reprezentující uzly, protože již máme k dispozici celkový počet příspěvků daného zdroje a časové údaje jednotlivých příspěvků. Po této fázi již můžeme přejít k části zpracování informací pro rozložení v rámci sítě.

7.3 Sběr dat a ukládání do příslušných struktur

V této sekci popíšu jak probíhá ukládání dat do jednotlivých struktur. Při kliknutí na uzel se ve funkci *display_data* zavolá funkce *fin_p2()*, která zažádá server o data spojená s daným příspěvkem a uloží je do příslušných struktur. Po získání pole hodnot v části kódu *Promise.all(TA.loadEntryContents(entry)).then(values)* dochází v cyklu k přidělení hodnoty do struktury, dle jejího typu. V následujícím seznamu jsou možné typy dat, kód pro jejich zpracování a vysvětlení použití daného kódu.

Textový obsah Text daného příspěvku. Do struktury pro textová data je přidán text a id, podle kterého lze identifikovat uzel, ke kterému text patří.

```
text_data.push({id: idk,content:val.text});
```

Výpis 7.6: Ukázka kódu pro získání textového obsahu

Grafický obsah Obrázky spojené s daným příspěvkem. Do struktury je přidán html obsah, pro pozdější zobrazení. Html kód obsahuje referenci na daný obrázek. Dále je opět přidáno id k identifikaci.

```
photo_data.push({id: idk,content:'<a href="' + val.sourceUrl + '">
<img style="' + "width:200px; height:auto;" + '" src="' + val.sourceUrl
+ '" alt=""></a>'});
```

Výpis 7.7: Ukázka kódu pro získání grafického obsahu

URL obsah Odkazy spojené s daným příspěvkem. Do struktury je přidán html obsah, pro pozdější zobrazení. Html kód referuje na jinou stránku. Opět je přidáno id k identifikaci.

```
URL_data.push({id: idk,content:'URL: <a href="' + val.sourceUrl + '">'
+ val.text + '</a>'});
```

Výpis 7.8: Ukázka kódu pro získání grafického obsahu

Linkový obsah Tento obsah je přidáván v jiné funkci a to ve funkci *links2*. Obsahuje data s informacemi o spojeném příspěvku. Například kdy byl příspěvek vytvořen a odkaz na něj. Do struktury je přidán html obsah, pro pozdější zobrazení. Znovu je přidáno id k identifikaci.

```
link_data.push({id:kk,content:'<dd>' +
'<span class="linkname">' + link.label + '</span> ' +
'<a href="#" class="proflabel">' + link.srclabel + '</a> ' +
'(<span class="etext">' + link.text + '</span>) ' +
'</dd>'});
```

Výpis 7.9: Ukázka kódu pro získání grafického obsahu

7.4 Funkce pro přípravu dat k zobrazení

V této části popíšu jak se data upravují, aby mohla být zobrazena. Po ukončení iniciálního sběru dat a zpracování všech zdrojů je zavolána funkce *process_info* , která pro každý zdroj vyhledá jeho prvky, seřadí je podle data, přiřadí jim barvu a následně je propojí. Po provedení těchto úkonů následuje funkce *positioning*.

Tato funkce má za úkol vytvořit vhodné rozložení prvků. Nejprve je nutno zavolat funkci *getDays*, ve které se vzestupně seřadí dříve získaná časové údaje a vypočítá se počet dnů, ve kterých se příspěvky vyskytují. Následovně se dle počtu dnů naplní struktura *day _space _struct* hodnotami pro určení xové souřadnice uzlů. Každá z těchto hodnot odpovídá jednomu dni. Každý den pak odpovídá jednomu sektoru zobrazení uzlů.

Pro každý zdroj se určí xová souřadnice jeho uzlů, tak že uzel má dostatečný prostor ze všech stran, aby byl dobře vidět a bylo jasné, do jakého dne patří. V cyklu se jednotlivým uzlům zdroje přiřazuje zvyšující se hodnota *inix*.

Pokud je uzel prvním uzlem dne, přidá se do zobrazovací struktury popis obsahující celé datum dne. Pokud tomu tak není, zobrazí se pouze čas v hodinách, minutách a sekundách. Pokud uzel obsahuje datum dalšího dne, do hodnoty *inix* se přiřadí hodnota pro další sektor, ze struktury *day _space _struct* a *inix* se nadále navyšuje od této hodnoty.

V případě že *inix* v daném sektoru překročí předpřipravenou hodnotu ve struktuře *day _space _struct* pro další sektor, je nutné tuto hodnotu posunout a rozložení přepočítat. Tento způsob nám umožňuje získat sektory odpovídající rozsahu počtu uzlů v jednotlivých dnech.

Pro každý zdroj se pak upravuje hodnota *ypos*, která udává ypsilonovou souřadnici uzlů. Po dokončení algoritmu dochází k naplnění struktury *nodes* zpracovanými daty. Struktura *nodes* je pak už přímo využívána knihovnou *vis.js*, k vyobrazení uzlů v grafické ploše.

Před sekcí zobrazení, se ale volá funkce *process_relations2*, která má za úkol zpracovat informace o vztazích a vhodně je zakomponovat do struktury *edges*, která pak opět slouží knihovně *vis.js* pro zobrazení.

V cyklu se prochází všechny vztahové informace dříve získané serverem a na základě jednotlivých údajů se přidávají záznamy do zmíněné struktury *edges*.

Po dokončení tohoto úkonu je vše připraveno k volání funcke invoke jež pracuje s knihovnou *vis.js* na samotném zobrazení dat.

7.5 Zobrazení dat v grafické ploše

K iniciaci zobrazení dochází ve funkci *invoke*, kdy se využijí data uložená v již zmíněných strukturách *nodes* a *edges*. Na začátku funkce se vytvoří pozice, vzhled a popis legendy, která slouží k rozeznání zdrojů v ploše a přidá se do struktury *nodes*.

Dále je třeba vytvořit proměnou *container*, která se vztahuje k HTML elementu *canv*, jenž určuje pozici a vlastnosti grafické plochy ve stránce.

```
var container = document.getElementById('canv');
```

Výpis 7.10: Vytvoření proměnné container

Následuje přidání dat do *Vis.js* struktur. Na základě těchto struktur se vytváří uzly a hrany v síti. Ve strukturách jsou obsaženy informace o pozici, vzhledu a vlastnostech prvků.

```
var data = {
          nodes: visnodes,
          edges: visedges
      };
```

Výpis 7.11: Předání dat

Poté můžeme nastavit rozličné parametry uzlů, hran a samotného chování sítě ve struktuře *options*.

Nastavení interakce, konkrétně zobrazení navigace a naslouchání tlačítkům klávesnice.

```
interaction: {
              navigationButtons: true,
              keyboard: true
          },
```

Výpis 7.12: Nastavení interakce

Defaultní nastavení uzlů. Uzly mohou být specifikovány, jak obecně v tomto nastavení, tak v rámci skupin nebo zcela jednotlivě.

```
nodes: {
              shape: 'dot',
              size: 30,
              font: {
                  size: 32,
                  color: '#ffffff'
              },
              borderWidth: 2,
              margin: 10
          },
```

Výpis 7.13: Nastavení uzlů

V této části lze definovat celou řadu různých nastavení sítě nebo požadovaného fyzického chování. Jedním z širokých prvků, které lze takto ovlivňovat je fyzické chování sítě. Tato funkčnost, ale již z dříve vysvětlených důvodů zůstává neaktivní, tedy není nutné ji nějako nastavovat. Jedním z chování, které nás ale zajímá z hlediska výkonnosti je vykreslování křivek. Standardně je aktivováno dynamické vykreslování křivek, které sice vypadá nejlépe, ale zároveň je největší zátěží pro vykreslení a tedy při větším počtu uzlů a křivek mezi

nimi má negativní vliv na výkonnost. Tento vliv se projevuje například poklesem Frames Per Second při práci se sítí nebo prodloužením času samotného vykreslení sítě. V části nastavení toto můžeme ovlivnit změnou vykreslování z dynamického na kontinuální, což se projeví menšími výkonnostními nároky a tedy lepším vzhledem sítě.

Samotné vytvoření sítě a zobrazení dat pak proběhne zavoláním knihovní funkce *vis.Network*.

```
network = new vis.Network(container, data, options);
```

Výpis 7.14: Vytvoření sítě

Síti je nutné dát vhodné rozměry a určit počáteční bod vzhledem ke stránce neboli souřadnice x rovno nule a y rovno nule. Tenhle úkon je proveden následujícím kódem.

```
network.moveTo({
          position: {x: 0, y: 0},
          offset: {x: -500, y: -150},
          scale: 1,
      });
```

Výpis 7.15: Pozicování sítě

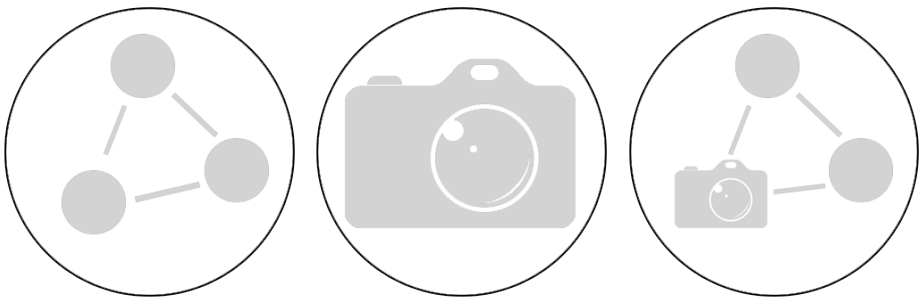
7.6 Vzhled dat v grafické síti

Vhodným prvkem při vizualizaci dat je odlišení jednotlivých typů dat v rámci sítě. Některé příspěvky mohou obsahovat fotografie, některé propojení s jinými příspěvky a tyto skutečnosti je vhodné náležitě graficky reprezentovat pro snadnou a rychlou orientaci v rozlehlé síti dat. Proto se ve chvíli, kdy uživatel klikne na nějaký uzel, kromě zobrazení dat mění i vzhled uzlu.

Problematika je řešena způsobem úpravy vzhledu uzlů, které obsahují různé typy informací. Nejdříve je zaznamenán typ dat při jejich získání ve struktuře *object_nodes* do atributu *content_type*. V takovéto situaci pak musíme rozlišit několik typů přístupů.

Například v situaci, kdy data příspěvku obsahují fotografii, je nutné zjistit, zda neobsahují ještě data spojená se vztahem k jiným příspěvkům. Na základě toho je pak určen výsledný typ obsahu a poté je dle něj přidělena grafická podoba ve funkci *design*.

Jak již bylo zmíněno, různý obsah dat je reprezentován různým grafickým zobrazením. Typy zobrazení jsou pak následující.



Obrázek 7.1: Design uzlů s určitým obsahem

První obrázek zleva znázorňuje informaci, že daný uzel je propojený s jiným uzlem. Další obrázek zleva znázorňuje informaci, že příspěvek reprezentovaný daným uzlem obsahuje

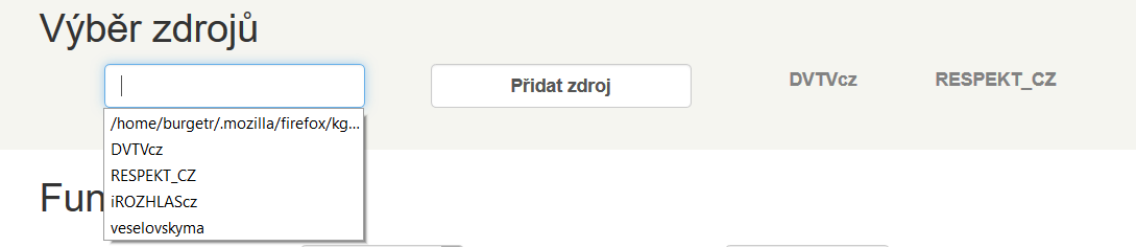
fotografie. Třetí obrázek zleva představuje situaci, která je kombinací obojího, tedy že uzel obsahuje fotografii a zároveň je nějak propojený.

7.7 Grafické uživatelské rozhraní a manipulace se sítí

V této sekci se věnuji zpracování uživatelského rozhraní a funkcím, které manipulují se sítí nebo zobrazují data v k tomuto účelu přizpůsobené části rozhraní.

První částí je klasické menu tvořené html prvky a vzhledově upravené pomocí knihovny *Bootstrap*.

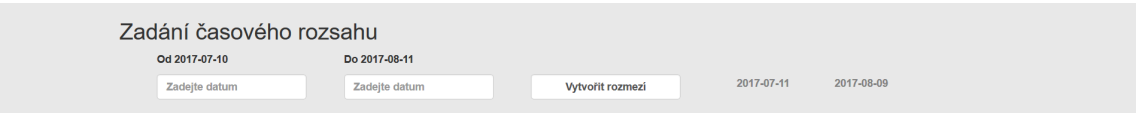
Další částí je sekce pro výběr zdrojů. Obsahuje formulář pro výběr zdrojů, o které má uživatel zájem. Pro ulehčení výběru je použit datalist, který nabídne možnosti zdrojů. Obsah datalistu je závislý na datech databáze. Nabídka jednotlivých zdrojů se získá tak, že se dotážeme serveru na zdroje, které obsahuje. Toho dosáhneme pomocí již dříve zmíněné metody *TAClient.prototype.getTimelines*, která nám tyto data zpřístupní.



Obrázek 7.2: Ukázka zdrojů

Po přidání zdroje se zavolá funkce *add*, která zdroj nejen zobrazí v pravé části okna přidáním do elementu *adder*, ale taky se přidá do struktury *li_sources*. Při vykonávání se volá funkce *source_parse*, která přidává dle zdroje o jaký se jedná do struktury *source_structure* timelineURI a zároveň kontroluje, zda je hodnota platná. V případě nutnosti odebrat zdroj z vybraných se volá funkce *delet*, která zdroj odebere jak z náhledu, tak ze struktury *source_structure*. Hodnoty v *source_structure* jsou důležité z hlediska pozdějšího využití při inicializaci.

Hned potom následuje část, ve které uživatel vybírá časové rozmezí příspěvků. Určuje zde datum prvního a posledního příspěvku, o který má zájem. Toto rozmezí je omezeno hranicemi, které jsou určeny dle data prvního a posledního příspěvku uloženého v databázi. Rozmezí se získá pomocí metody TA Clienta *DateLimits* , která vytvoří adekvátní dotaz a pomocí RDF4j Clienta ho vykoná na serveru. Následně je rozmezí zobrazeno v uživatelském rozhraní. Uživatel je pak žádán zadat rozmezí do formulářů a po potvrzení se hodnota ve funkci *timeadd* přidá do seznamu, který je opět zobrazen. Hodnota je náležitě kontrolována jestli je ve správném formátu a zda odpovídá mezím.



Obrázek 7.3: Ukázka výběru mezí

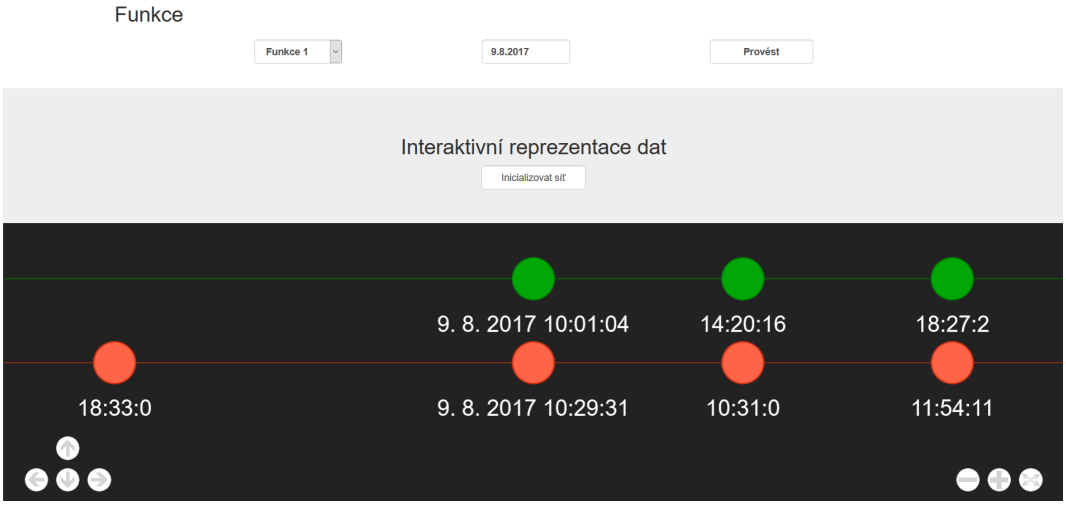
V další sekci máme formulář, který má za účel vybrat funkci nad sítí, kterou chceme provést a dodat jí požadovaný parametr. Mezi tyto funkce patří například možnost pohybu a zaměření kamery na zdroj určitého data. V kódu se pak při výběru této možnosti volá funkce *foc*. Nejprve se zkontroluje, zda vůbec uzel s daným datem je v síti přítomen. Pokud ano, vyhledá se pole uzlů s daným datem a kamera se zaměří na uzel z tohoto pole, kterým je vždy počáteční uzel dne. V případě, že funkci voláme opakovaně, kamera se zaměřuje postupně na uzly stejného data, od prvního až po poslední. Zaměření kamery je realizováno následujícím kódem.

```
var nodeId = result[ar_index].content.id;
            var options = {
            when focusing on nodes
                scale: 1.0,
                offset: {x:0,y:0},
                animation: {
                    duration: 1000,
                    easingFunction: 'easeInQuad'
                }
            };

            network.focus(nodeId,options);
```

Výpis 7.16: Kamerový přechod

Jsme schopni ovlivnit konkrétní pozici zaměření pomocí atributu *offset*, následně vlastnosti animace pomocí atributu *animation*. Konkrétně můžeme vidět dobu přechodu v atributu *duration* a animační funkci, která určuje, jak bude přechod vypadat.

Další funkcí je možnost posouvání se na prvky spojené s vybraným uzlem. Při opakovaném volání funkce se opět posouváme na všechny spojené prvky vybraného uzlu.



Obrázek 7.4: Ukázka zaměření

Další částí je pak sekce s formulářem pro inicializaci sítě. Po aktivaci tlačítka dojde k volání funkce *init* s uživatelem navolenými zdroji z již zmíněné struktury *source_structure*. Tímto se dostáváme k samotné zobrazovací ploše, která je reprezentována html elementem

div s id *canv*. V rámci plochy se lze libovolně pohybovat myší nebo klávesami. Orientaci taky usnadňují jednoduchá tlačítka pro pohyb, přiblížení a navrácení pohledu na celou síť. S uzly lze libovolně manipulovat, při najetí zobrazovat náhled dat a při aktivaci jejich data zobrazit.

Poslední sekcí pak je část pro zobrazení dat z uzlů. Každý typ dat má vyhrazený prostor. Zobrazení dat nastává při kliknutí na uzel. Kliknutí vyvolá událost v síti, která je řešena následující funkcí.

```
network.on("click", function (params) {
            params.event = "[original event]";
            if(this.getNodeAt(params.pointer.DOM) != undefined)
                    display_data(this.getNodeAt(params.pointer.DOM));
        });
```

Výpis 7.17: Onclick událost

Při vykonávání je volána funkce *display_data* s informací o tom, o který uzel se jedná. Na základě toho pak funkce vyhledá jednotlivá data spojená s daným uzlem a upraví html kód stránky.

Zobrazená data pak mohou být opět interaktivní. Například kliknutím na odkaz se lze dostat na původní příspěvek nebo zobrazit fotografii na externí stránce ve větším rozlišení.

Obrázek 7.5: Ukázka zobrazení

7.8 Ošetření výkonnosti

Při zobrazování velikého počtu dat, je třeba provést opatření zajišťující přijatelnou dobu provedení základního zobrazení a následně udržení dobrého počtu Frames Per Second při manipulaci se sítí.

Pro tento účel je nutné sledovat počet příspěvků a adekvátně na něj reagovat. V určitých mezích pak měníme vlastnosti sítě na základě tohoto počtu a přizpůsobujeme tak výkonnost zobrazení.

Konkrétně se daná problematika řeší ve funkci *performance_manage*, kdy se podle počtu příspěvků vybere vhodné nastavení sítě, které je následně aplikováno při zobrazování. V této funkci se rozhoduje například, jaký bude typ vykreslování křivek a zda se křivky budou vykreslovat při pohybu a manipulaci.

Kapitola 8

Testování

Testování je esenciální součástí každého projektu. Kromě testování samotného chodu aplikace neboli ověřování splnění požadavků na funkčnost je nutné testovat uživatelské prostředí. U uživatelského prostředí je pak nutné zjistit, zda je uživatelsky přívětivé, dá se v něm snadno a rychle orientovat a zda poskytuje přístup ke všem žádaným funkcím. Zároveň je třeba zjistit, zda aplikace dosahuje přijatelné rychlosti zobrazení a plynulosti práce s grafickým uživatelským rozhraním. Rozsáhlejší testování probíhalo na velkém vzorku dat, konkrétně pak obsahujícím 28 profilů, 14164 příspěvků a 184410 RDF trojic. Vzorek dat je ze sociálních sítí Facebook a Twitter.

8.1 Testování funkcionality

V této fázi se provádělo ověřování, zda aplikace zvládá všechny požadované funkce. Nejprve bylo nutné ověřit získávání dat neboli funkčnost komunikace mezi klientem a serverem. Tato část zahrnovala ověření správnosti dotazů, konfigurace serveru a správnost metod požadujících a zpracovávajících data v rámci Timeline Analyzer a RDF4J klienta.

Následně se testovalo žádání a zpracovávání dat v knihovně *data_functions.js*. Tuto část bylo na základě zkoušení nutno několikrát přepracovat, aby bylo dosaženo požadované funkčnosti a správné návaznosti na další zpracovávání.

Další částí bylo testování přípravy dat k vizualizaci. V této fázi bylo ověřováno, zda jsou data předaná touto částí knihovně vis.js, následně vizualizována dle požadovaného konceptu. Během testování postupně docházelo ke změnám algoritmu, dokud se nedošlo k hledanému výsledku.

Po fázi zobrazení bylo nutné kontrolovat správnost funkčnosti vizualizované sítě dat v podobě uzlů a hran. Testoval se pohyb a manipulace s uzly, pohyb v síti, přiblížení v síti a celkový vzhled zobrazení. Poté bylo nutné ověřit, zda se při interakci s uzly správně zobrazují požadovaná data v příslušném prostoru. V této části se nevyskytlo mnoho problémů, pouze bylo nutné zahrnout možnost zobrazení většího počtu stejného typu dat v určených prostorech.

8.2 Testování výkonnosti

U aplikace, jejímž cílem je vizualizace velkého množství dat je nutno brát ohled na výkonnost a čas, za který se daná data podaří zobrazit. V této části tedy bylo nutné otestovat dané ohledy.

Při zkoušení aplikace s větším množstvím příspěvků bylo zjištěno, že rozmísťovací algoritmus prvků je neefektivní a časově náročný. Z tohoto důvodu bylo nutné algoritmus přepracovat, tak aby byl méně výpočetně náročný. Po přepracování algoritmu a dalším testování byly provedeny další úpravy kódu, které měli za následek zrychlení běhu aplikace a tedy dřívější zobrazení požadovaných dat.

Dalším problémem, který byl odhalen u testování na větším vzorku dat bylo neefektivní získávání dat od serveru. Přístup, kdy se získávala všechna data příspěvků již při inicializaci, vedl k příliš pomalému, až nefunkčnímu chodu aplikace. Z tohoto důvodu bylo nutné získávání dat poupravit tak, aby se vlastní data jednotlivých příspěvků získávala až na vyžádání. Tento krok vedl k výraznému zrychlení aplikace a přijatelné odezvě i při velikém počtu příspěvků.

Kromě délky samotného zobrazení se testoval i průběh manipulace s již vizualizovanou sítí uzlů. Konkrétně tedy Frames Per Second při práci s uzly a pohybem v grafické ploše. V této fázi bylo na základě testů vyvinuto několik mechanismů, které v závislosti na velikosti dat mění nastavení chování funkcí knihovny vis.js, které v důsledku pozitivně ovlivňují výkonnost.

8.3 Testování uživatelského rozhraní

Dalším předmětem testování je samotné grafické uživatelské rozhraní, kdy je potřeba ověřit, zda je uživatelské rozhraní intuitivní, snadno pochopitelné a odpovídá potřebám uživatele.

Pro účel testování osobami byl zvolen jednoduchý seznam úkolů.

1. Zvolení a odstranění zdrojů
2. Manipulace s uzly
3. Procházení sítí
4. Zkoušení zaměřovací funkce
5. Sledování zobrazení dat při rozkliknutí příspěvku
6. Sledování návaznosti vztahů
7. Testování odkazů

Při testování bylo například odhaleno, že chybí možnost zvolení časového intervalu, ve kterém leží příspěvky, jež uživatele zajímají. Tato vlastnost musela být následně doimplementována a grafické rozhraní upraveno. Dále bylo zjištěno, že by bylo vhodné zakomponovat legendu mimo síť pro lepší orientaci ve větším množství dat. Legenda má podobu jednoduché tabulky barev s přiřazenými příslušnými zdroji. Dále byly vytvořeny další funkce nad sítí, které upravují uživatelskou zkušenost. Jedná se například o funkci procházení uzlů spojených s uživatelem vybraným uzlem. Při aktivaci se kamera posouvá po spojených uzlech tak, aby uživatel nemusel spojené uzly hledat ručně. Dále jde o funkci, která vypíná a zapíná zobrazení hran, což zvyšuje v určitých situacích přehlednost a také má pozitivní vliv na plynulost zobrazování. Finální podobu grafického uživatelského rozhraní lze vidět v příloze B.1.

Kapitola 9

Závěr

Úkolem této bakalářské práce bylo vytvořit webovou aplikaci pro zobrazení dat příspěvků ze sociálních sítí. Toto zobrazení muselo být ve formě, která by reprezentovala nejen samotná data příspěvků, ale i jejich časové rozložení a vztahové návaznosti mezi jednotlivými příspěvky.

V počátku byly definovány požadavky, podle kterých je nutné vyhledávat technologii pro realizaci daného zobrazení. V souladu s těmito kritérii byly zkoumány mnohé možnosti provedení zobrazení dat, kdy se postupně vybírali jednotlivé technologie do užšího výběru, kde byly porovnány na základě dalších kritérií. Po vyloučení nevhodných možností a zvážení výhod a nevýhod nástrojů v užším výběru byla následně vybrána vhodná technologie, kterou se ukázala být knihovna Vis.js.

Po výběru vizualizačního nástroje bylo třeba vybrat vhodný datový zdroj. Před tímto krokem bylo však nutné se nejdříve seznámit s existujícím systémem pro sběr událostí. Na základě informací o existujícím aplikačním rozhraní byl pak zvolen framework pro práci s RDF daty RDF4J, který nabízí RDF4J server jako SPARQL endpoint a RDF4J workbench jako uživatelské prostředí pro práci s RDF databází. RDF4J pak používá jako svůj JAVA Servlet container osvědčené řešení Apache Tomcat.

Po části zvolení technologií, frameworků a jazyka k implementaci aplikace bylo potřeba navrhnout samotný koncept zobrazení dat. Bylo nutné vzít v potaz uživatelská kritéria pro zobrazení dat a najít co možná nejlepší způsob, který by byl jak uživatelsky vyhovující, tak přijatelný z hlediska možností a výkonnostních omezení zvolené technologie. Výsledkem této fáze byl intuitivní a praktický koncept reprezentace příspěvků pomocí uzlů a vztahů mezi nimi pomocí hran. Samotné zobrazení dat jednotlivých příspěvků bylo uváženo jako reakce aplikace na podnět uživatele v samostatném prostoru mimo síť uzlů. V této části bylo také navrženo webové grafické uživatelské rozhraní pro práci s aplikací, ve kterém je umístěna zobrazovací plocha.

Po této fázi se pracovalo na vytvoření samotné aplikace, jež si zažádá o data, náležitě je zpracuje a pak je za využití knihovny vis.js zobrazí. Získání velkého množství dat bylo řešeno využitím technologie promise, která umožnila použití asynchronních žádostí, čímž vedla k výraznému urychlení získání dat. Dále byl navrhnut algoritmus pro rozřazení příspěvků v rámci zobrazovací grafické plochy dle vytvořeného konceptu zobrazení. Výsledným rozložením pomocí tohoto algoritmu je pak síť uzlů rozdělená na jednotlivé sektory, které reprezentují jednotlivé dny časové osy. Kromě backendové části bylo nutné vytvořit grafické uživatelské rozhraní za pomocí designové knihovny Bootstrap a jazyků CSS a HTML5.

Následně bylo nutné aplikaci otestovat v hlediscích funkčnosti, praktičnosti, výkonnosti zobrazení a práce s uživatelským rozhraním. K tomuto účelu bylo mimo jiné zapotřebí

velkého testovacího vzorku dat, kterým bylo 14164 příspěvků ze sociálních sítí Facebook a Twitter. Při testech byly odhaleny jak problémy správnosti řešení, tak nízké rychlosti zobrazení, které byly postupně vyřešeny a odstraněny. Například bylo třeba přepracovat algoritmus rozřazení uzlů v grafické ploše z důvodu vysoké časové náročnosti. Na základě zpětné vazby uživatelů došlo ke změnám grafického uživatelského rozhraní, kdy bylo nutné přidat určité prvky a upravit nedostatky týkající se designu. Postupně se tedy fází testování došlo k finální podobě aplikace, která je vyhovující počátečním požadavkům.

Během práce bylo nutné se naučit pracovat s novými nástroji, nastudovat teoretickou stránku použitých přístupů a následně ji prakticky využít. Konkrétně se jednalo například o koncept sémantického webu, technologii JavaScript promise nebo programování s třídami v prostředí JavaScript. Výsledkem mé práce je pak intuitivní, přehledná a účelům vyhovující aplikace, která nabízí kvalitní možnost průzkumu uživatelem požadovaných dat.

Literatura

[1] DuCharme, B.: *Learning SPARQL*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2011, iSBN: 9814350605.

[2] Ing. Radek Burget, Ph.D.: *Sociální sítě: Sběr a analýza dat v souvislosti s bezpečnostními incidenty*. 2017, [Online; navštíveno 29.02.2018].
URL http://www.fit.vutbr.cz/~burgetr/pubs.php?id=11573

[3] Kolektiv autorů: Apache Tomcat: *Apache Tomcat*. [Online; navštíveno 23.02.2018].
URL https://en.wikipedia.org/wiki/Apache_Tomcat

[4] Kolektiv autorů: Mozzila contributors: *Object prototypes*. [Online; navštíveno 23.02.2018].
URL https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes

[5] Kolektiv autorů: Mozzila contributors: *Promise*. [Online; navštíveno 22.02.2018].
URL https://developer.mozilla.org/cs/docs/Web/JavaScript/Reference/Global_Objects/Promise

[6] Kolektiv autorů: OWL Working Group: *Ontology web language*. [Online; navštíveno 23.02.2018].
URL https://www.w3.org/OWL/

[7] Kolektiv autorů: RDF Working Group: *Resource Description Framework*. [Online; navštíveno 23.02.2018].
URL https://www.w3.org/RDF/

[8] Kolektiv autorů: RDF4j: *RDF4j server a workbench*. [Online; navštíveno 23.02.2018].
URL http://docs.rdf4j.org/server-workbench-console/#_rdf4j_server_workbench_and_console

[9] Kolektiv autorů: RDF4j: *RDF4j úvod*. [Online; navštíveno 23.02.2018].
URL http://rdf4j.org/about/

[10] Kolektiv autorů: REST: *Representational State Transfer*. [Online; navštíveno 23.02.2018].
URL https://en.wikipedia.org/wiki/Representational_state_transfer

[11] Kolektiv autorů: SPARQL Working Group: *SPARQL*. [Online; navštíveno 23.02.2018].
URL https://www.w3.org/TR/rdf-sparql-query/

[12] Kolektiv autorů: Technical Committee 39: *JSON Syntax*. [Online; navštíveno 23.02.2018].
URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[13] Kolektiv autorů: Tomcat: *Tomcat úvod*. [Online; navštíveno 23.02.2018].
URL http://tomcat.apache.org/index.html

[14] Kolektiv autorů: W3C: *Semantic web*. [Online; navštíveno 23.02.2018].
URL https://www.w3.org/2001/sw/

[15] Kolektiv autorů: w3schools: *JSON úvod*. [Online; navštíveno 23.02.2018].
URL https://www.w3schools.com/js/js_json_intro.asp

[16] Martin Malý: *REST architektura*. [Online; navštíveno 23.02.2018].
URL https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[17] Pivotal Labs: *Javascript constructor, prototype a new*. [Online; navštíveno 23.02.2018].
URL https://content.pivotal.io/blog/javascript-constructors-prototypes-and-the-new-keyword

[18] Ryan Wang: *Servlet container*. [Online; navštíveno 23.02.2018].
URL https://dzone.com/articles/what-servlet-container

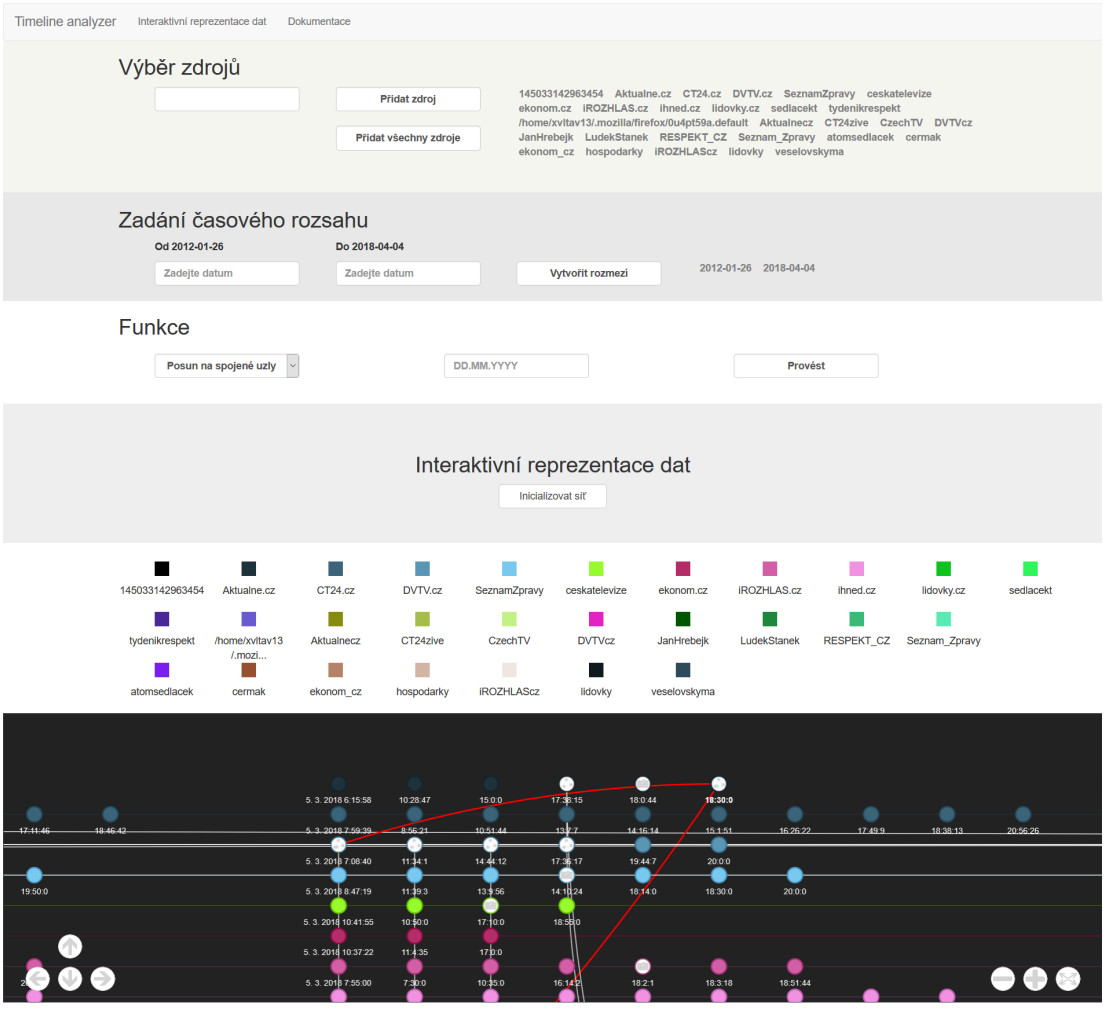
Příloha A

Obsah CD

- Technická zpráva (technicka_zprava.pdf)
- Zdrojové soubory (source_files.zip)
- Testovací data (test_data.zip)
- Popis k aplikaci (README)

Příloha B

Ukázky GUI



Obrázek B.1: Vzhled GUI