VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÉ UŽIVATELSKÉ ROZHRANÍ NÁSTROJE PRO EXTRAKCI INFORMACÍ

WEB USER INTERFACE FOR A INFORMATION EXTRACTION TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN POKORNÝ**
AUTHOR

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

BRNO 2017

Zadání bakalářské práce/19291/2016/xpokor67

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Pokorný Jan**
Obor: Informační technologie
Téma: **Webové uživatelské rozhraní nástroje pro extrakci informací**
Web User Interface for a Information Extraction Tool
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou extrakce informací z dokumentů a s nástrojem FITLayout pro analýzu dokumentů.
2. Prostudujte současné technologie pro tvorbu webových aplikací s klientem a serverovou částí na platformě JavaScript.
3. Navrhněte architekturu aplikace implementující webové rozhraní k serverovému nástroji pro extrakci informací z dokumentů.
4. Po dohodě s vedoucím implementujte navrženou aplikaci s využitím vhodných technologií.
5. Implementujte možnost přístupu více uživatelů a funkce pro tvorbu a správu extrakčních úloh.
6. Proveďte testování vytvořené aplikace a zhodnoťte dosažené výsledky.

Literatura:

- Swicegood, T.: Programming Node.js, O´REILLY, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

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 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

V této práci se můžete dočíst o návrhu a implementaci JavaScriptové aplikace, která slouží jako uživatelské rozhraní pro nástroj k extrakci dat. Aplikace nabízí prostředí, ve kterém si uživatel spravuje extrakční úlohy. Úlohy jsou vytvářeny pomocí interaktivních grafů. Této funkcionality je docíleno pomocí současných moderních trendů z oblasti JavaScriptových aplikací, které jsou v práci popsány. Zejména se jedná o knihovnu React a správce stavu Redux.

Abstract

In this work you can read about the design and implementation of the JavaScript application, which serves as a user interface for the data extraction tool. The application offers an environment in which the user manages extraction tasks. Tasks are created using interactive graphs. This functionality is achieved through the current modern trends in JavaScript applications that are described in the work. In particular, it is a React library and Redux state manager.

Klíčová slova

JavaScript, React, Redux, REST API, extrakce informací

Keywords

JavaScript, React, Redux, REST API, data mining

Citace

POKORNÝ, Jan. *Webové uživatelské rozhraní nástroje pro extrakci informací*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Webové uživatelské rozhraní nástroje pro extrakci informací

Prohlášení

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

.....
Jan Pokorný
29. května 2017

Poděkování

Rád bych poděkoval panu doktoru Burgetovi za vedení práce a za odborné rady. Také bych chtěl poděkovat báječné komunitě, která se kolem JavaScriptových aplikací pohybuje.

Obsah

Kapitola 1

Úvod

Směle bychom mohli tvrdit, že vývoj webových aplikací a technologie s tím spojené prožívají neustálé revoluce. Nové technologie a přístupy vznikají obdivuhodným tempem a technologie, které nejsou řádně zakořeněné, jsou zapomínány. Webový prohlížeč přestal být pouhým tenkým klientem pro prohlížení dat, ale začíná plnit roli hybridní platformy pro sofistikované aplikace, které byly dříve dostupné pouze jako klasický počítačový program.

Ruku v ruce s dynamickým vývojem webových technologií jde i problematika extrakce dat. Internet je stále zdrojem informací, které je třeba dále zpracovávat. Webové stránky bývají zpravidla navrženy tak, aby poskytovaly informace uživateli jakožto živé bytosti. Avšak nejsou to jen lidé, kteří potřebují efektivně získávat informace prostřednictvím internetu. Na mysli mám specializované nástroje – roboty, pro které tato činnost není snadná.

Tato práce pojednává o návrhu a implementaci moderní JavaScriptové aplikace, která bude sloužit jako rozhraní mezi uživatelem a nástrojem pro extrakci dat.

V následující kapitole si můžete přečíst krátký úvod do problematiky extrakce dat. Dočtete se zde o různých přístupech k extrakci dat a také o nástroji FITLayout, kterým je možné extrakce provádět. Z této kapitoly také vyplynou požadavky na výslednou aplikaci.

Kapitola třetí se zabývá různými technologiemi, jež s vývojem aplikace souvisely. Je zde zmíněn zejména poslední trend, který se na poli JavaScriptových aplikací odehrává.

Ve čtvrté kapitole vás seznámím s návrhem aplikace. Můžete zde nahlédnout na rozvržení server / klient a na role použitých knihoven. Naleznete zde i výstižný diagram celé architektury. Dále se zde také můžete dočíst o návrhu uživatelského rozhraní, o reprezentaci extrakčních úloh a také můžete nahlédnout na návrh databáze.

Další kapitola pojednává o implementaci. Zde si můžete přečíst popis implementace zajímavých a náročnějších částí aplikace. Naleznete zde podrobný popis implementace rozhraní pro tvorbu extrakčních úloh, řešení administrace a popis komunikace se serverem.

Předposlední kapitola je věnována vyhodnocení. Dočtete se zde o kompatibilitě aplikace napříč prohlížeči a o součinnosti aplikace s extrakčním nástrojem.

Poslední kapitola je tradičně věnovaná závěru.

Kapitola 2

Extrakce informací z WWW

Extrakcí informací se rozumí proces, při kterém se snažíme z informačních zdrojů vyjímat informace relevantní z hlediska obsahu dokumentů nebo z hlediska informačního dotazu [2]. Jinými slovy můžeme říct, že se jedná o proces, při kterém identifikujeme datové položky a následně je ukládáme do vhodného strukturovaného formátu. Tuto činnost dělá běžně každý uživatel internetu bez vynaložení většího úsilí. Oproti tomu extrakce informací pomocí programů je netriviální problém. Webové stránky jsou primárně navrženy tak, aby usnadnily práci uživatele. Jejich strojová čitelnost bývá zpravidla až druhotný úkol.

K problematice extrakce dat z dokumentů můžeme přistupovat hned několika způsoby. Jeden z nejtriviálnějších způsobů je využití regulárních výrazů. Při využití regulárních výrazů se zpravidla nejprve snažíme o lokalizaci jednotlivých částí dokumentu – hlavička, obsah, zápatí a jiné. Po nalezení relevantní části dokumentu aplikujeme sadu regulárních výrazů zaměřených již na konkrétní data. Při tvorbě regulárních výrazů využíváme často přítomnost značek jazyka HTML[1].

Další, o něco více sofistikovaný přístup je založen na práci s DOMem[2]. Základní internetový dokument je tvořen textem a značkami jazyka HTML. Tato kombinace utváří hierarchickou strukturu nazývanou DOM [33]. Je tedy možné navrhnout cestu DOMem, kterou se program musí vydat, aby našel potřebné data.

Oba výše zmíněné způsoby trpí zásadními nedostatky. U obou případů je nezbytné extrakční úlohy na míru specifikovat konkrétnímu dokumentu.Tyto metody také selžou v okamžiku, kdy se zdrojový dokument změní. Navzdory těmto problémům jsou tyto metody stále velmi používané, a to především díky své jednoduchosti.

O něco sofistikovanější je přístup, kdy na dokument nahlížíme i po vizuální stránce. Při této metodě dokument nejdříve vykreslíme a poté jej rozdělíme na vizuální segmenty. Segmentace může probíhat ve dvou směrech. Segmentace shora dolů nahlíží na dokument jako na celek a drolí jej dále na menší segmenty. Naopak segmentace zdola nahoru nejdříve zpracuje nejmenší segmenty, které dále spojuje do větších celků [3]. Jelikož se při této metodě využívá i grafická reprezentace dokumentu, můžeme tvrdit, že je tato metoda bližší lidskému zpracování. Výsledkem segmentace je hierarchický vizuální model. Z výsledného modelu se poté odfiltrují nepotřené části a ve zbylých oblastech se poté snažíme najít strukturu, která by odpovídala hledané informaci [5].

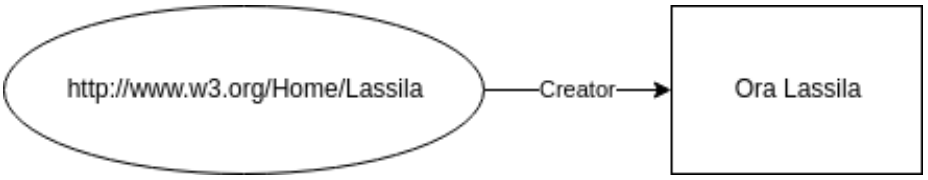
[1] HyperText Markup Language
[2] Document Object Model

2.1 Resource Description Framework (RDF)

RDF je framework spravovaný konsorciem W3C, který se zabývá popisem a výměnou dat. RDF je grafový model tvořen trojicemi: subjekt, predikát, objekt. Tyto trojice tvoří tvrzení. Univerzálním prvek RDF je zdroj, který je identifikovaný svým URI[3]. URI adresa je ukazatelem do ontologických slovníků a dává zdroji význam. Zdroj může být subjektem i objektem. Predikát je sledovaná vlastnost subjektu a objekt je její hodnota. Kromě zdroje může být objektem i primitivní datová hodnota [29].

Datový model RDF nemá pevně danou reprezentaci. RDF model můžeme reprezentovat graficky nebo pomocí serializačních jazyků, jako jsou například XML[4], JSON[5], Turtle a jiné.

Na obrázku 2.1 a v ukázce kódu 2.1 si můžete prohlédnout příklad užití RDF. Příklad byl převzat z práce pana Svátka [29].



Obrázek 2.1: Příklad RDF reprezentovaného grafem

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Ukázka kódu 2.1: Příklad RDF serializovaného v XML

Framework RDF bude využit pro specifikaci extrakční úlohy. Pomocí RDF budeme popisovat data, která mají být extrahována. Také nám popis v RDF stanoví strukturu dat, tedy jejich způsob uložení.

2.2 FITLayout – nástroj pro extrakci dat

FITLayout je aplikační rámec pro segmentaci a analýzu dat webových stránek. Tento nástroj byl vytvořen na Fakultě informačních technologií VUT v Brně [4]. Rámec nabízí rozhraní, ve kterém je možné implementovat různé segmentační algoritmy. FITLayout je schopný díky knihovně CSSbox vykreslit webovou stránku a provést zvolenou segmentaci. Rámec dále nabízí nástroje na analýzu a klasifikaci získaných segmentů. Celý nástroj je implementovaný v jazyce Java.

[3] Universal Resource Identifier, zobecnění běžně používaných URL (Universal Resource Locator)
[4] eXtensible Markup Language
[5] JavaScript Object Notation

2.3 Cíle práce

Po segmentaci webové stránky se v segmentech snažíme najít určitý vzorec dat. Předpis tohoto vzorce dat je v současné podobě definován přímo v kódu nástroje FITLayout. Uživatel musí nástroj FITLayout nainstalovat a jeho kód si přizpůsobit. Extrakční úloha (onen vzorec dat) je v systému reprezentovaná pomocí RDF.

Cílem práce je nabídnout tento nástroj skrze internetovou službu. Je potřeba navrhnout aplikaci, která bude bránou mezi uživatelem a nástrojem FITLayout. Tato aplikace musí především nabídnout uživatelsky přívětivé rozhraní pro správu extrakčních úloh – RDF výrazů. Jak již bylo zmíněno, RDF nemá konkrétní reprezentaci. Z hlediska uživatelského zážitku bude grafická reprezentace RDF nejpříjemnější. Aplikace tedy nabídne uživateli rozhraní, ve kterém si bude moct vytvářet grafy, jež budou reprezentací RDF.

Dále je potřeba navrhnout vhodný formát, kterým aplikace zadání extrakčních úloh předá nástroji FITLayout.

Výsledkem má být moderní aplikace dostupná skrze internetový prohlížeč. Z těchto důvodů je potřeba prostudovat poslední trend na poli JavaScriptových aplikací a následně pomocí vhodných technologií aplikaci implementovat.

Kapitola 3

Související technologie

Vývoj internetových aplikací je poměrně komplexní záležitost, která sahá hned do několika různých oborů informatiky. Proto při vývoji internetových aplikací je třeba znát hned několik technologií, programovacích jazyků, nástrojů, knihoven, principů a zvyklostí. Také zde obzvlášť platí pravidlo, že to, co již bylo jednou vymyšleno, není třeba vymýšlet znova.

V této kapitole se pokusím popsat z mého pohledu nejzajímavější a nejužitečnější technologie, které jsem při práci využil.

3.1 JavaScript a ECMAScript 6

Jazyku JavaScript se věnuje poslední dobou značná pozornost a jazyk sám prochází velkým vývojem.

Jazyk JavaScript je implementací standardu ECMAScript [22]. Standard ECMAScript je vydáván ve verzích, které obsahují vlastnosti (features). Verze JavaScriptu se tedy odvíjí od verze ECMAScriptu.

V červnu 2015 byla vydána šestá verze standardu ECMAScriptu s názvem ECMAScript 2015 [9]. Tato nová verze přinesla řadu nových konstrukcí a vlastností [10]. Mohli bychom tvrdit, že JavaScript velmi zmodernizovala.

Z mého pohledu nejmarkantnější je nová definice tříd. Objekty je nyní možné definovat pomocí tříd, podobně jako u jiných třídních jazyků. Stejně tak funguje dědičnost tříd. A to vše navzdory tomu, že JavaScript je jazyk prototypový [22].

Další zajímavou novinkou, viz ukázka 3.1, je nová syntaxe funkcí, která kromě názornějšího a kratšího zápisu nevytváří nový kontext.

```
1 const soucet = (a, b) => (a + b);
2 const pozdrav = () => { console.log('Hello'); }
```

Ukázka kódu 3.1: Nový syntax definice funkcí

V neposlední řadě přichází ES6 s *Promises*. Promise je objekt sloužící pro práci s asynchronními výpočty. Dříve bylo zvykem, že funkce, která se vykonává asynchronně, měla navíc dva parametry. První parametr byla funkce, která se volá s výsledkem po dokončení asynchronní části kódu. Druhá funkce se volá v případě selhání s informací o chybě. Výpočty tohoto typu se často řetězí a používají anonymní funkce. Výsledkem bývá nečitelný kód s velkým množstvím zanořených funkcí. S příchodem Promises již není nutné funkce

do sebe zanořovat a provolávat. Namísto toho se využívá klasického navracení hodnot, případně vyhazování výjimek. Řetězení je docíleno opakovaným voláním metody *then* Promise objektu, viz ukázka 3.2.

```
1 stahniData()
2   .then(rawData => zpracujData(rawData))
3   .then((data) => {console.log(data)})
4   .catch(e => {console.log('Chyba ....')});
```

Ukázka kódu 3.2: Zpracování asynchronního dotazu pomocí Promises

ES6 přináší řadu dalších neméně významných inovací, avšak jejich výčet a popis by si vyžádal samostatný dokument.

3.1.1 Překlad kódu do starších verzí ECMAScriptu

Jelikož nové standardy jsou implementovány pozvolna a nové vlastnosti jsou u vývojářů velmi žádané, je běžné, že vývojáři píší kód v nejnovějším standardu a kód poté překládají zpět do starších standardů ECMAScriptu.

Jeden z takových překladačů se nazývá *Babel*. V Babelu je možné si zvolit, jaké vlastnosti nebo skupiny vlastností mají být překládány [20]. Překlad můžeme také definovat výčtem prohlížečů, které mají kódu rozumět. Babel poté sám určí, co je třeba přeložit a co může zůstat v původní formě.

Babel provádí pouze transformaci syntaxe. Za účelem přidání nových funkcí (například zmíněné Promises) je nutné do výsledné aplikace přibalit polyfill[1], který Babel sám nabízí.

3.1.2 Jiné jazyky překládané do JavaScriptu

Je potřeba se také zmínit, že existují i jiné jazyky, které se překládají do JavaScriptu. Mezi ty nejznámější patří *TypeScript*, který je vyvíjen Microsoftem [21]. TypeScript má k JavaScriptu poměrně blízko, avšak zavádí (mimo jiné) dosud novou vlastnost, a to statické typovaní. Nutno podotknout, že statické typování je nyní možné použít i v klasickém JavaScriptu pomocí Babelu a modulu Flow.

Další možností je například jazyk Dart [14] nebo funkcionální jazyk Elm [7].

3.2 Pomocné nástroje pro vývoj aplikací v JavaScriptu

Při vývoji webové aplikace v JavaScriptu kombinujeme často řadu knihoven, kód se snažíme logicky strukturovat do souborů a preferujeme konstrukce a vlastnosti z nejnovějších standardů ECMAScriptu.

Naopak pro běh v běžném prohlížeči preferujeme co nejmenší počet a velikost načítaných souborů a co možná nejvyšší míru kompatibility.

Řešením těchto protichůdných požadavků bývá nástroj – bundler, který má za úkol zabalit jednotlivé soubory a provést případné předzpracování.

[1] JavaScriptový kód, který doplní funkcionalitu prohlížeče

3.2.1 Webpack

Současně velmi populární bundler je Webpack [19].

Primárním úkolem Webpacku je rekurzivně procházet zdrojový kód, hledat žádosti o načtení dalších souborů a provádět předzpracování. Do předzpracování můžeme zahrnout překlady standardů, filtrace ladících funkcí, optimalizace atd. Výsledkem jsou poté soubory nazývané *assests*, které jsou připraveny pro použití v prohlížeči.

Webpack nepracuje pouze se soubory JavaScriptu. Webpackem je také možné zpracovávat CSS a jeho nástavby, optimalizovat obrázky či jen kopírovat potřebné soubory.

Webpack také umožňuje *watch mód*, při kterém kontroluje změny v souborech a provádí svou činnost automaticky na pozadí.

Další velmi užitečnou součástí Webpacku je integrovaný vývojový server, který je vybaven technologií *hot reloading*. Hot reloading znamená, že změna ve zdrojovém kódu nemusí nutně vést k restartování celé aplikace. Pokud je to možné, systém sám za běhu nahradí starý kód novým a stav aplikace zůstává zachován. Nutno podotknout, že tato možnost nefunguje s libovolným kódem či knihovnou, ale je potřeba kód na tuto možnost předem připravit.

3.2.2 ESlint

Za úvahu také určitě stojí použití linterů, což jsou nástroje, které kontrolují kód a upozorňují na pochybení, která by nemusela být jinak objevena.

V JavaScriptu můžeme využít například ESlint [18]. S pomocí ESlintu se můžeme vyhnout problémům s konzistencí kódu, nejasným konstrukcím jazyka a jiným typickým potížím.

Pravidla hlídaná ESlintem je možné definovat pomocí konfiguračního souboru. Seznam hlídaných pravidel můžeme definovat sami, avšak je lepší se odkázat na již existující seznam, který je udržován komunitou.

3.3 React knihovna pro práci s DOMem

Knihovna React je vyvíjena vývojáři Facebooku a poprvé byla zveřejněna v roce 2013 [12]. V kontextu softwarové architektury *MVC* hraje knihovna roli *V*. React umožňuje tvorbu rozsáhlého interaktivního uživatelského rozhraní snadným, deklarativním způsobem.

Jádrem Reactu jsou komponenty. Myšlenkou Reactu je deklarovat co možná nejmenší znovupoužitelné komponenty a výslednou aplikaci tvořit kompozicí komponent.

Další myšlenkou je snaha o přesunutí vývoje kompletně do jazyka JavaScript a vyhnout se tak problémům spojeným s využíváním HTML šablon. React proto nabízí sadu elementárních komponent, které reprezentují základní HTML elementy, a také zavádí nový *syntax sugar* nazývaný *JSX*. Díky JSX je možné komponenty konstruovat a zanořovat stejným způsobem, jako to dělá jazyk HTML (viz ukázka 3.3).

```
const dom = (
  <Header background="blue">
      <h1>{lang === 'cs' ? : 'Ahoj' 'Hello'}</h1>
      <Menu />
  </Header>
);
```

Ukázka kódu 3.3: Ukázka zápisu JSX

Za účelem maximalizace efektivity práce s DOMem využívá React technologii virtuálního DOMu. React veškeré operace s DOMem provádí nejdříve v paměti a poté rozhoduje, které části skutečného DOMu je nutné v prohlížeči aktualizovat.

3.3.1 Komponenty

Základem komponenty v Reactu jsou *props*, *state* a metody životního cyklu.

Props jsou proměnné, které komponenta získává od svého rodiče, oproti tomu state jsou proměnné, které si komponenta spravuje sama. Tyto proměnné dohromady definují stav komponenty. Komponenta vždy reflektuje svůj stav. Je-li stav komponenty (props nebo state) změněn, komponenta na to reaguje svým překreslením.

Metody životního cyklu jsou metody, které se volají ve významných chvílích života komponenty. Například metoda *componentDidMount* – komponenta se vložila do DOMu, *componentDidUpdate* – komponenta se aktualizovala atd. Nejdůležitější metodou životního cyklu je metoda *render*. Metoda render slouží k vykreslení komponenty a volá se vždy, když se komponentě změní stav. Toto chování lze optimalizovat metodou *shouldComponentUpdate*, kde je možné rozhodnout, zda je nutné komponentu překreslovat.

Vedle klasických komponent existují i komponenty čisté (pure). Jedná se o komponenty s předem definovanou metodou *shouldComponentUpdate*, ve které probíhá kontrola stavu, a pokud nedošlo ke změně, komponenta se nepřekresluje. Jedná se tedy v podstatě o běžnou komponentu s integrovanou optimalizací.

Kromě klasických třídních komponent je možné definovat komponentu funkcí. Tyto komponenty postrádají metody životního cyklu a proměnné state. Jejich výhodou je menší paměťová náročnost a snadnější testování.

3.3.2 Alternativy Reactu

Za konkurenta Reactu by mohl být považován Angular 2 [13]. Angular je rozsáhlý framework, pokrývající různé aspekty aplikace. Oproti tomu React je knihovna, která se soustředí pouze na určitou část problematiky. React až ve spojení s jinými knihovnami vytváří takzvaný *devstack* – alternativu k frameworku. Z tohoto důvodu je velmi obtížné obě knihovny porovnávat. Volba mezi Reactem a Angularem je spíš otázka souhlasu s jejich filozofií.

Mezi výhody Angularu můžeme zařadit:

1. Méně prostoru pro nevhodná rozhodnutí – plyne z robustnosti frameworku
2. Ověřenost časem – Angular je starší a zavedenější
3. Kompatibilita s technologií Webcomponents
4. Využití a přímá podpora jazyka TypeScript

Další, často diskutovaný konkurent Reactu je VueJs. VueJs sdílí s Reactem podobné myšlenky [34] jako například: virtuální DOM, kompozice komponent či zaměření se pouze na jádro knihovny. Na rozdíl od Reactu využívá VueJs klasické HTML šablony namísto JSX.

3.4 Správce stavu Redux

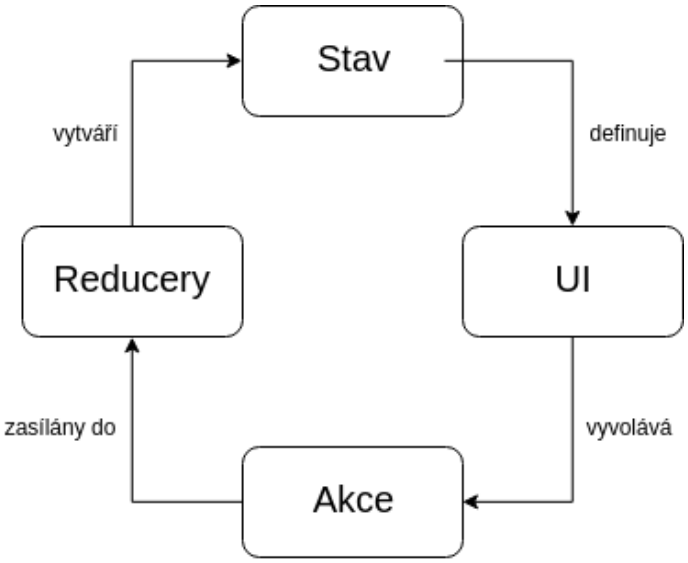
Z kapitoly o Reactu je patrné, že klíčovou roli aplikace hraje její stav (stav komponent). Proto je dobré stanovit si nějaký návrhový vzor, kterým je stav aplikace řízen, nebo využít knihovnu, která správu stavu řeší.

Jedním z takových správců stavu je *Redux* [1]. S Reduxem přichází tři nové termíny: *akce*, *reducer* a *kontejner*. Tok řízení stavu je možné vidět na obrázku 3.1.

Akce je jediný způsob jak změnit stav aplikace. Samotná akce je reprezentovaná jednoduchým objektem, který vždy obsahuje informaci o názvu akce a případná data.

Reducer je funkce, která spravuje část stavu aplikace. Akce jsou zasílány reducerům a ty na ně reagují. Pokud reducer obdrží akci, jež se týká stavu, který spravuje, tak na základě akce vytvoří nový stav. Důležitou vlastností je, že se stav nemění, ale vytváří se nový.

Kontejner je komponenta, které je předána část stavu aplikace. Kontejner poté tento stav distribuuje dál do zanořených komponent.



Obrázek 3.1: Tok řízení stavu aplikace

Myšlenkou Reduxu je zbavit jednotlivé komponenty stavu a mít pro celé jádro aplikace pouze jeden globální stav, který je jen pro čtení a aktualizovat jej lze pouze akcí.

3.4.1 Alternativy Reduxu

V první řadě je potřeba se zamyslet, zda správce stavu potřebujeme. Zavedení správce stavu znamená psaní podstatně více kódu, a určité omezení lehkosti celého projektu. Pro určité typy projektu nemá správce stavu význam, naopak by byl přítěží.

Za předchůdce Reduxu by mohl být označen Flux [11]. Flux byl vydán vývojáři Facebooku právě jako odpověď na otázku, jak spravovat stav u komplexních projektů. Redux se od Fluxu zásadně neliší, myšlenka Reduxu i Fluxu je v podstatě stejná [1]. Avšak Redux se jeví transparentnější a rychlejší na pochopení.

Další zajímavý správce stavu je MobX, který do problematiky přináší prvky reaktivního programování. Také nabízí přímé řešení problému, jak efektivně aktualizovat data, která jsou od stavu odvozená [15].

3.5 Server na platformě NodeJs pomocí knihovny Express

Knihovna Express slouží pro tvorbu serverového rozhraní [23]. Základní jednotkou knihovny je *middleware*. Middleware jsou funkce, které se řetězí a zpracovávají serveru položený dotaz. Middleware je tedy funkce se třemi parametry: *request*, *response*, *next*. Request reprezentuje dotaz, který server obdržel. Response reprezentuje odpověď, kterou server odešle. Next je funkce, kterou middleware zavolá, jakmile je hotov, a předá tak běh programu dalšímu middleware. Middleware je možné zavádět plošně, tedy pro všechny dotazy, nebo je možné jej specifikovat pro konkrétní požadavek. V praxi se poté serverová aplikace skládá z plošných middleware, definic párů cesta – HTTP metoda a middlewarů pro konkrétní páry. Poslední middleware posílá odpověď a uzavírá dotaz viz, ukázka 3.4.

```
app.use(globalMiddleware);
router.get('/user/:id', authMiddleware, function(req, res){
  if (!req.user.isLoggedIn) {
      res.status(401).send('Error you are not logged in!');
    }else{
      res.send('Requested id is '+ req.params.id);
    }
});
```

Ukázka kódu 3.4: Příklad použití knihovny Express

3.6 ORM knihovna Sequelize

ORM neboli objektově relační mapovaní je programátorská technika, kdy se snažíme spojit objektově orientovaný přístup s relační databází. Cílem ORM je vytvořit abstrakci nad relační databází tak, aby odpovídala objektově orientovanému návrhu. Programátor poté pracuje s objekty, kterými reprezentuje reálné entity, a techniky ORM se poté starají o jejich perzistenci. Při využití ORM knihovny nejsme závislí na konkrétním typu relační databáze. Také není potřeba tvořit složité SQL dotazy, knihovna je efektivně tvoří sama. Dále nám knihovna může usnadnit návrh databáze. Nutno podotknout, že využití ORM knihoven má i svoje nevýhody a dalo by se říct, že dělí programátory do dvou skupin, a to na příznivce a odpůrce. Avšak v naši skromné aplikaci je využití ORM knihovny nesporná výhoda.

V jazyce JavaScript nalezneme hned řadu ORM knihoven, z nichž se nejlépe jeví knihovna Sequelize. Knihovna Sequelize podporuje hned několik databází, a to MySQL, MariaDB, SQLite, Postgres a MS-SQL [28]. Knihovna pracuje se dvěma pojmy, *model* a *instance*. Model reprezentuje entitu, odvíjí se od něj podoba tabulek a slouží k dotazování databáze. Instance reprezentuje konkrétní záznam. Instance může být perzistentní, čili odpovídá konkrétnímu záznamu v databázi, nebo neperzistentní – existuje pouze v paměti.

3.7 Serverová architektura REST

REST (Representational State Transfer) je architektura rozhraní orientovaná na zdroje. Každý zdroj má svůj jednoznačný identifikátor a definovanou sadu CRUD[2] operací.

Architektura REST jasně odděluje klienta od serveru. Klient i server jsou v podstatě nezávislé aplikace, které si mezi sebou vyměňují data v nějakém serializačním jazyku, z pravidla jazyk JSON nebo XML. Server tak může obsluhovat více klientů, přičemž každý může fungovat na úplně jiné platformě. Například jeden server může sloužit pro webovou i mobilní aplikaci. Stejně tak klient může komunikovat s více servery. Například kdyby šlo o aplikaci filmový katalog, může aplikace získávat filmy z více serverů najednou.

Zajímavou vlastností REST serveru je bezestavost. Každý požadavek od libovolného klienta musí obsahovat kompletní informaci potřebnou k jeho obsloužení. To vede k úspoře prostředků serveru, jelikož si server nemusí držet kontext s každou aplikací.

REST server zpravidla běží na protokolu HTTP. V tom případě jsou URL adresy využity jako identifikátory zdrojů a HTTP metody jsou využity jako CRUD operace (GET – read, POST – update, PUT – create, DELETE – delete).

3.7.1 Autentizace vůči REST serveru

Z absence kontextu na straně serveru plyne, že pokud nějaký zdroj vyžaduje autorizaci, požadavek na tento zdroj musí vždy obsahovat informace nezbytné pro autentizaci. Pro tyto účely lze využít hned několik standardů.

Za nejjednodušší můžeme považovat HTTP-Basic. HTTP-Basic přidává do hlavičky komunikace uživatelské jméno a heslo. Tyto přístupové údaje jsou v hlavičce zakódovány pouze pomocí Base64. Přihlašovací údaje nejsou tedy nijak zabezpečeny a bez šifrování na nižší komunikační vrstvě jsou velmi snadno zneužitelné [17].

Problém s přihlašovacími údaji v otevřené podobě řeší vylepšený protokol HTTP-Digest, který pracuje na principu *výzva-odpověď* a posílá pouze otisk přihlašovacích údajů [27].

Další možností je autentizace pomocí tokenů. Klient si na začátku komunikaci autentizuje například přihlašovacím formulářem a server mu přidělí unikátní token. Další požadavky na serveru obsahují v hlavičce onen token, který slouží k prokázání identity.

[2] create, read, update, delete

Kapitola 4

Návrh

S použitými technologiemi jsem bohužel neměl předchozí zkušenosti. Proto jsem se vždy musel nejdříve s danou technologií seznámit, udělat návrh a poté provést implementaci. Často se při implementaci ukázalo, že můj prvotní návrh není optimální a že si s určitými problémy neporadí. Z těchto důvodů bylo třeba návrh aplikace provádět iterativně.

V této kapitole vám přiblížím finální návrh aplikace.

4.1 Architektura aplikace

Cílem práce je vytvořit takzvanou single page aplikaci (SPA). SPA je charakteristická tím, že se snaží spojit výhody desktopových aplikací a dostupnost webových stránek. Jinými slovy se SPA snaží nabídnout uživatelský zážitek desktopové aplikace skrze webový prohlížeč. Tento přístup nabourává staré konvence, kdy webová aplikace byla otázkou především serveru a webový prohlížeč sloužil pouze jako zprostředkovatel. U SPA je přístup zcela opačný. Vývoj je soustředěn na klientskou část a server je zpravidla pouze zdrojem dat.

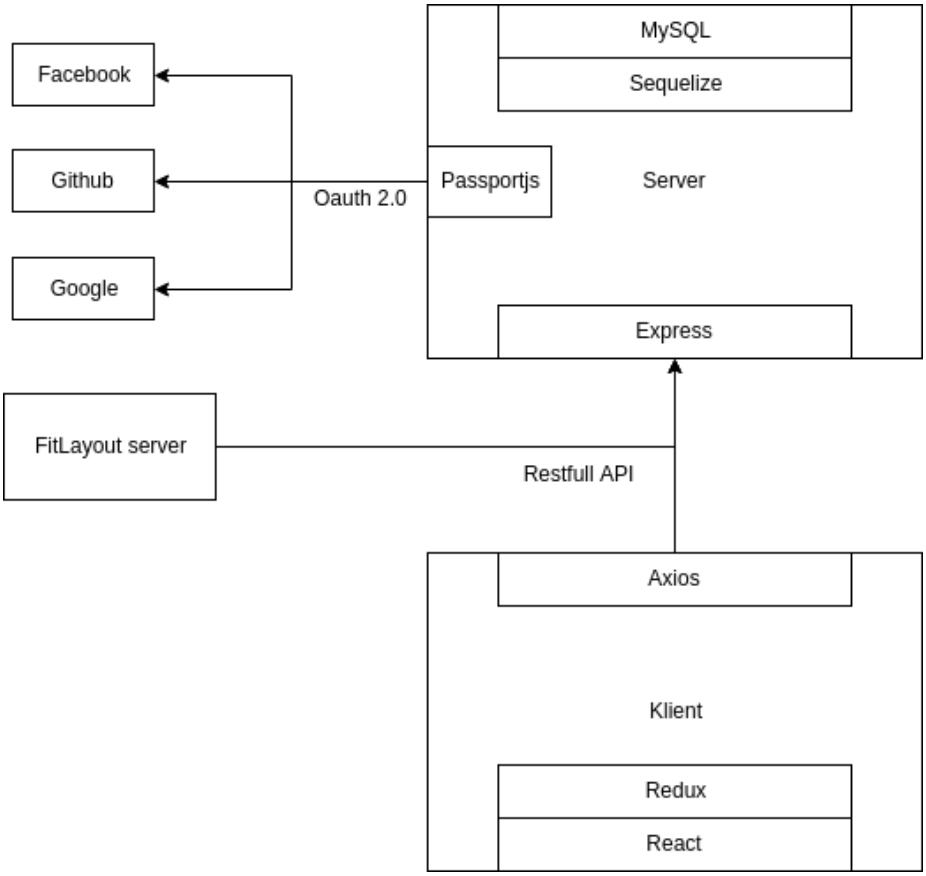
V našem případě tedy využijeme jednoduchý server na platformě NodeJs s využitím knihovny Express. Server bude mít dva úkoly. Prvním úkolem je distribuce statických souborů, tím je myšleno prvotní načtení webové stránky. Druhým úkolem je poskytnutí REST rozhraní pro přenášení a ukládání dat. Pro ukládání dat použijeme databázový server MySQL, se kterým budeme komunikovat pomocí ORM knihovny Sequelize. Abychom zbytečně uživatele nezatěžovali registrací a přihlašováním, implementujeme možnost registrace a přihlášení skrze sociální sítě. Kromě standardního formuláře nabídneme uživateli registraci skrze Facebook, Github nebo Google. S implementací nám pomůže knihovna Passport [16].

Na straně klienta vytvoříme aplikaci pomocí knihovny React a správce stavu Redux. Stav aplikace rozdělíme do několika částí a ke každé části napíšeme příslušné akce. Komunikaci se serverem skrze API nám usnadní knihovna Axios [35].

Z praktických důvodů bude klient i server samostatný projekt. V produkčním i vývojovém režimu bude server sloužit jako datový sklad. Avšak při vývoji bude distribuce aplikace prováděna skrze vývojový nástroj Webpack. Pro produkci bude aplikace sestavena a zkopírována do projektu se serverem.

Podrobný nákres architektury aplikace si můžete prohlédnout na obrázku 4.1.

Úkolem aplikace je nabídnout uživateli rozhraní pro správu extrakčních úloh. Typický scénář užití vypadá následovně:



Obrázek 4.1: Diagram architektury a použitých knihoven

Uživatel přistoupí na webovou stránku, tím si stáhne potřebné statické soubory a spustí se aplikace. Uživatel prokáže svou identitu, na základě toho dostane přístupový token. Poté se pohybuje v aplikaci, spravuje extrakční úlohy, které se průběžně odesílají na server. Po dokončení práce uživatel kontaktuje speciální server, na kterém běží nástroj FITLayout. Tento server kontaktuje náš server, jenž poskytne informace o zadání extrakční úlohy, kterou si předtím uživatel v aplikaci vytvořil.

4.2 Reprezentace extrakční úlohy

Jak již bylo zmíněno, extrakční úlohu specifikujeme pomocí RDF a RDF reprezentujeme grafem. Graf se skládá z uzlů a orientovaných hran (šipek).

Uzel je buď RDF subjekt, nebo objekt. Každý uzel má svůj unikátní identifikátor a skládá se z názvu, seznamu URI adres a informaci zda reprezentuje objekt nebo primitivní datová hodnota. V rozhraní je zobrazen jako obdélník s okrajem. Vnitřek obdélníku slouží k posunu uzlu po plátně a je zde zobrazen název uzlu. Okraj uzlu slouží jako zdroj a cíl hran.

Hrana je RDF predikát. Hrana reprezentuje vztah mezi uzly. Hrana je tvořena identifikátory zdrojového a cílového uzlu, názvem, seznamem URI adres a kardinalitou vztahu.

Hrany můžeme rozdělit na dvě kategorie. Hrany, které spojují dva různé uzly, a hrany, které pojí ten samý uzel. Hrana dvou různých uzlů je reprezentována úsečkou zakončenou šipkou. Hrana dvou různých uzlů musí být unikátní. Hrana pojící stejný uzel je reprezentována smyčkou. U obou typů hran se dále vizualizuje název a kardinalita vztahu.

Hranu i uzel můžeme označit až čtyřmi URI adresami. Pro usnadnění práce se společná část URI adres zkracuje a vzniká tak prefix, který se posléze doplňuje sufixy.

4.3 Návrh uživatelského rozhraní

Aplikace má za úkol poskytnout uživateli rozhraní pro správu extrakčních úloh. Extrakční úlohu definujeme grafem. Úkolem je tedy navrhnout rozhraní pro správu specifických grafů. Prvotní úvahy směřovaly k využití již existující knihovny zabývající se práci s grafy. Později se ovšem ukázalo, že námi potřebné grafy jsou příliš specifické a existující knihovny naopak příliš obecné. Došel jsem tedy k názoru, že přizpůsobení existující knihovny našemu problému by bylo náročnější než vymyslet vlastní řešení. Také by hrozilo, že by v průběhu práce došlo k objevení požadavku, který by knihovna nedokázala zpracovat.

Co se týče zbytku aplikace, využijeme již hotovou sadu grafických komponent. To nám značně ušetří práci a uživateli nabídne prostředí, které je mu blízké. Nabízí se zde dobře známá knihovna Bootstrap [26], ale nakonec byla vybrána knihovna implementující principy Material Designu [6], a to z důvodů vyšší kompatibility se zbytkem aplikace.

4.3.1 Drátěný model a jeho popis

Na obrázku 4.2 je zobrazen drátěný model stránky se správou grafu.

Záhlaví slouží k navigaci v aplikaci.

Seznam úloh, jak název napovídá, poskytuje uživateli přehled o jeho extrakčních úlohách. Uživatel zde může přepínat mezi jednotlivými úlohami nebo vytvořit úlohu novou.

Formulář pro správu úlohy slouží ke konfiguraci parametrů úlohy. Zvolit zde můžeme název úlohy a URL adresu zdrojového dokumentu. Formulář dále nabízí tlačítka pro uložení či smazání úlohy, tlačítko pro automatické ukládání a v neposlední řadě zobrazuje URL adresu, kde je dostupný výsledek extrakce.

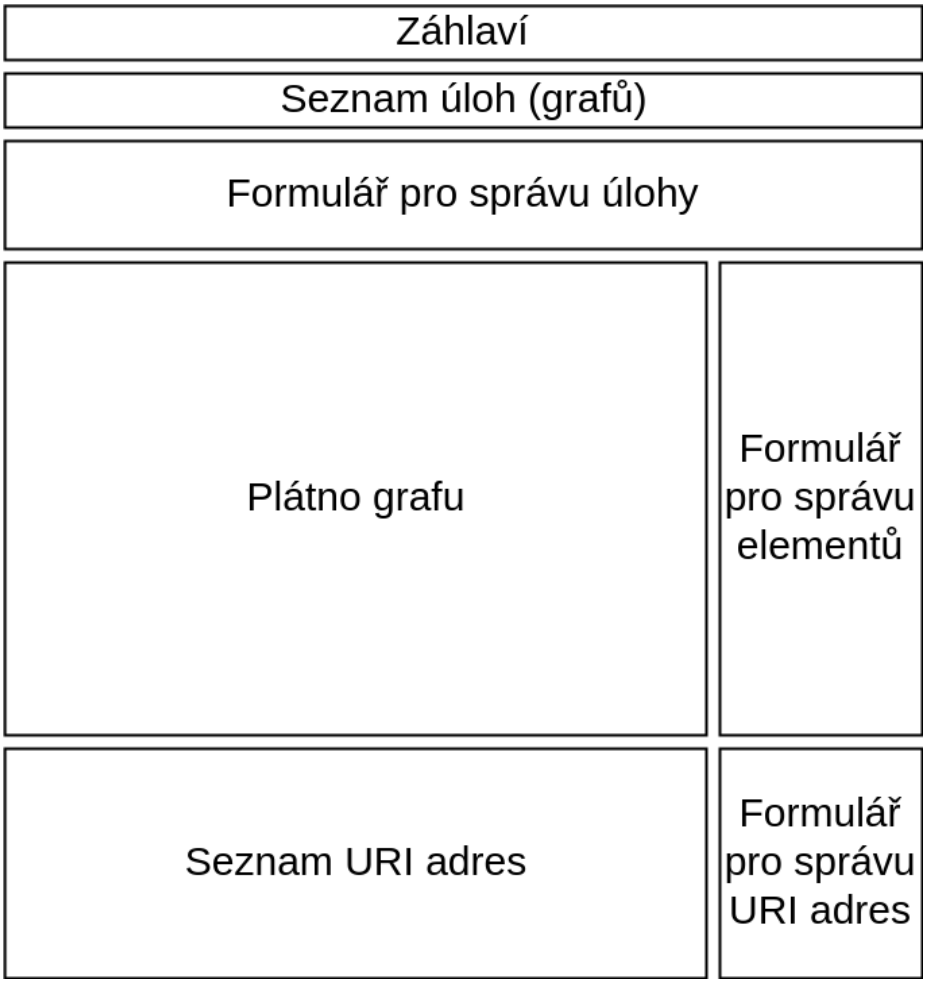
Plátno grafu slouží k formulaci extrakční úlohy – tvorbě RDF výrazů. Uživatel zde vkládá jednotlivé uzly a ty spojuje pomocí hran. Táhnutím myší může uživatel uzly po plátnu přesouvat. Kliknutím myši na uzel nebo hranu uživatel daný prvek označí.

Formulář pro správu elementů se mění v závislosti na označeném elementu (uzel nebo hrana) a slouží ke specifikaci parametrů elementu.

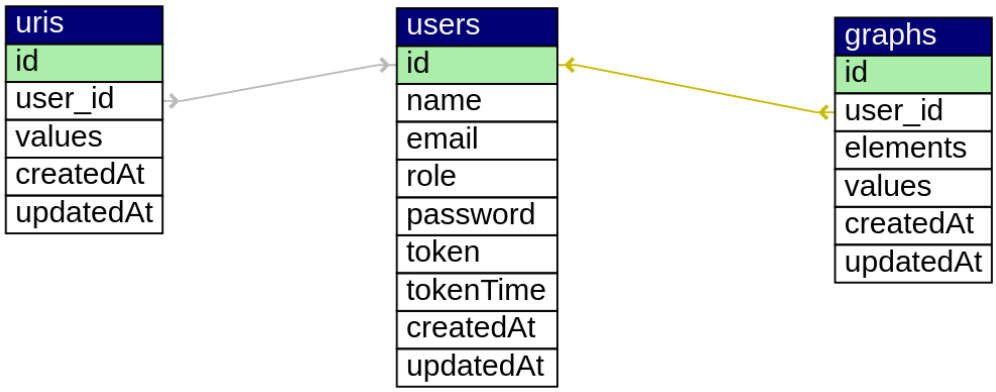
Seznam URI adres je nezávislý na konkrétní extrakční úloze a je pro všechny úlohy společný. Seznam zobrazuje uživatelem zadané URI adresy a jejich prefixy. Formulář pro správu URI adres slouží k jejich editaci a vytváření.

Původní návrh počítal s responzivní verzí, avšak kvůli technickým komplikacím byla velikost plátna stanovena fixně. Velikost plátna byla zvolena tak, aby bylo možné zároveň zobrazit plátno, formulář pro správu grafů a formulář pro správu úlohy již při základním rozlišení displeje 1366x768px.

Finální podobu stránky je možné vidět v příloze A na obrázku A.1.



Obrázek 4.2: Wireframe stránky se správou grafu



Obrázek 4.3: Databázové schéma

4.4 Návrh databáze

Na základě předchozích zkušeností byla zvolena relační databáze MySQL. Návrh databáze je ve výsledku velmi prostý, a to díky nativní podpoře datového formátu JSON, která je v MySQL přítomna od verze 5.7.8 [25]. Díky formátu JSON lze vytvořit velmi pružný návrh, který je možné v průběhu vývoje snadno měnit. Nevýhodou přímého ukládaní JSONu je absence integritních omezení a nemožnost specifikovat na tyto data SQL dotaz.

Finální návrh je možné vidět na obrázku 4.3. V tabulce *graphs* si můžeme všimnout položek *elements* a *values*, které jsou právě ve formátu JSON a celý návrh databáze tak zjednodušují. Stejný případ je položka *values* v tabulce *uris*. Výslednou podobu schématu si spravuje ORM knihovna Sequelize na základě specifikace v kódu.

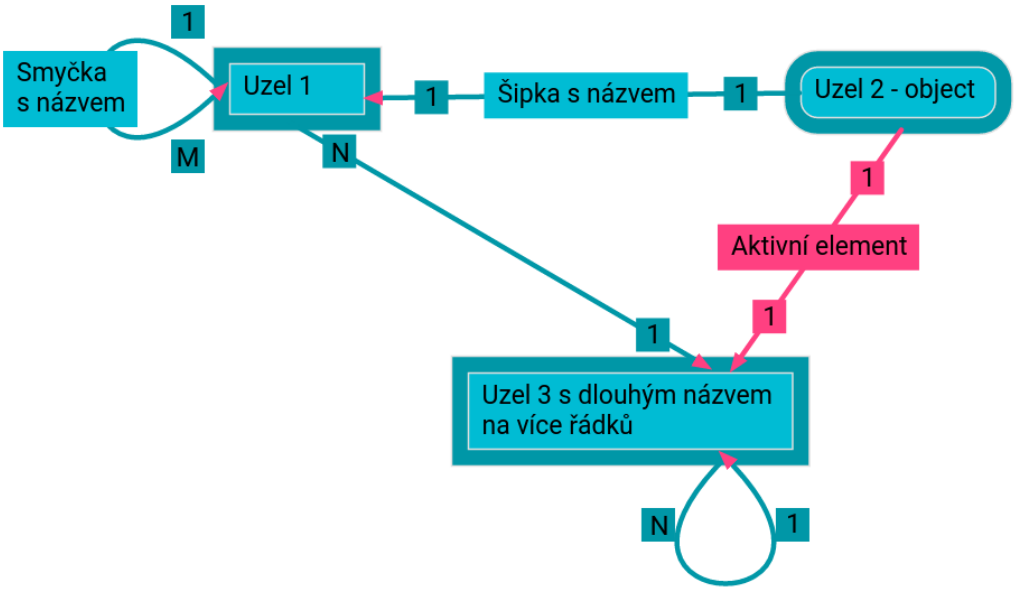
Kapitola 5

Implementace

V této kapitole naleznete podrobnější popis implementací zajímavých a náročnějších částí aplikace.

5.1 Vykreslování grafu

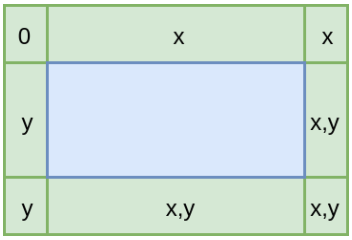
Výslednou podobu grafů je možné vidět na obrázku 5.1



Obrázek 5.1: Příklad grafu

Prvotní návrh počítal s vykreslováním grafů pouze pomocí klasického HTML a CSS. Později se ukázalo, že bude velmi obtížné tímto způsobem vykreslovat hrany. Jako řešení tohoto problému se nabízí formát SVG. Formát SVG je již široce podporován napříč prohlížeči [8] a umožňuje tak práci s vektorovou grafikou jakožto součástí běžného HTML DOMu.

Využití SVG má i svoje nevýhody. Jeden z problémů při práci s SVG je absence možnosti explicitně určit překrývání objektů. Objekty se překrývají podle pořadí vykreslení.



Obrázek 5.2: Schéma rozdělení uzlů na jednotlivé oblasti

Dalším problémem je práce s textem. SVG nepodporuje automatické zalamování textu. Pokud je potřeba text zalomit, je třeba si jej rozdělit na jednotlivé řádky a ty vykreslit za sebou s patřičným rozestupem. V SVG také není možné docílit automatického přizpůsobení velikosti objektů vůči textu.

5.1.1 Datová reprezentace a vykreslování uzlů

Datovou reprezentaci uzlu je možné vidět v ukázce kódu 5.1.

```
{
  "id": 1489765715524,
  "cords": {
    "x": 216,
    "y": 142
  },
  "width": 100,
  "height": 50,
  "values": {
    "uris": [
      {
        "id": 1,
        "sufix": "name",
        "prefix": "http://xmlns.com/foaf/spec/index.rdf"
      }
    ],
    "object": true
  }
}
```

Ukázka kódu 5.1: Příklad datové reprezentace uzlu

Každý uzel obsahuje svůj unikátní identifikátor, souřadnice umístění na plátně, velikost a hodnoty popsané v kapitole 4.2.

Na obrázku 5.2 je zobrazeno schéma uzlu. Uzel je rozdělen na dvě základní části. Zelená část uzlu slouží pro práci s hranami. V modré oblasti je možné uzel uchopit myší a posouvat po plátně. V modré oblasti je také umístěn název uzlu.

Výsledný uzel je vytvořen pomocí dvou obdélníků a textového elementu.

Jelikož technologie SVG neumožňuje vnoření textového elementu do elementu obdélníku, není velikost uzlu automaticky přizpůsobena velikosti textu. Z těchto důvodů se při

každé změně názvu uzlu zjistí velikost vykresleného textu a vyvolá se akce, která provede korekci velikosti uzlu.

5.1.2 Datová reprezentace a vykreslování hran

Datovou reprezentaci hrany je možné vidět v ukázce kódu 5.2.

```
{
  "id": 1489849243492,
  "dst": {
    "id": 1489765715524,
    "cords": { "x": 302, "y": 187 },
    "resizable": { "x": true, "y": true }
  },
  "src": {
    "id": 1489849241959,
    "cords": { "x": 480, "y": 320 },
    "resizable": { "x": false, "y": true }
  },
  "width": 23,
  "height": 28,
  "values": {
    "uris": [],
    "title": "Hrana 1",
    "cardinality": { "dst": true, "src": false }
  },
  "direction": null
}
```

Ukázka kódu 5.2: Příklad datové reprezentace hrany

Hrana je definovaná počátečním a koncovým bodem. Tyto body obsahují souřadnice, identifikátor uzlu a informaci o citlivosti na změnu velikosti uzlu. Dále hrana obsahuje šířku a výšku (zde míněno pro obdélník s názvem hrany), hodnoty popsané v kapitole 4.2 a položku *direction*, která udává směr u hran odkazujících na stejný uzel.

Citlivost na změnu velikosti udává, zda se má hrana posunout, dojde-li ke změně velikosti uzlu. Na obrázku 5.2 je zobrazeno osm zelených oblastí, kde může být umístěn počáteční nebo koncový bod hrany. Oblast *0* znamená, že změna velikosti uzlu nemá na pozici hrany vliv. Oblast x znamená, že pokud dojde ke změně *šířky* uzlu, tak i souřadnice x koncového bodu hrany musí být o stejný díl posunuta. Oblast y znamená, že pokud dojde ke změně *výšky* uzlu, tak i souřadnice y koncového bodu hrany musí být o stejný díl posunuta. Oblast x,y značí citlivost jak na *výšku*, tak *šířku* uzlu.

Nová hrana je vytvořena po kliknutí právě na jednu z osmi zelených oblastí. Souřadnice počátečního i koncového bodu nově vzniklé hrany jsou souřadnice kliknutí a identifikátor počátečního bodu se rovná identifikátoru zdrojového uzlu. Identifikátor koncového bodu je nulový. V této chvíli se nacházíme v situaci, kdy uživatel klikl a tlačítko myši stále neuvolnil. Následný pohyb myši vyvolává akce, které mění souřadnice koncového bodu nově vzniklé hrany. V případě, že je tlačítko uvolněno a souřadnice koncového bodu neodpovídají poloze

některého z přítomných uzlů, je hrana odstraněna. Pokud jsou souřadnice umístěny na ploše uzlu, je označen cílový uzel a souřadnice jsou upraveny tak, aby došlo k takzvané magnetizaci a šipka ukazovala opět do jedné z osmi zelených oblastí.

Hrana dvou různých uzlů

Hranou dvou různých uzlů se rozumí hrana, která má různý identifikátor koncového a počátečního bodu.

Tyto hrany jsou unikátní, není možné spojit dva uzly více než dvěma šipkami.

Hrany tohoto typu jsou reprezentovány úsečkou a její název je umístěn na střed. Pozice kardinality je počítána z vektoru úsečky tak, aby se vykreslila ve správném směru a v konstantní vzdálenosti od uzlu.

Hrana na jednom uzlu

Hranou na jednom uzlu se rozumí šipka, která vychází a ukazuje na stejný uzel. Tyto hrany jsou reprezentovány smyčkou a jejich počet je omezen na čtyři pro každý uzel.

Název hrany i kardinalita jsou v tomto případě umístěny v konstantní vzdálenosti od počátku hrany.

5.2 Administrace

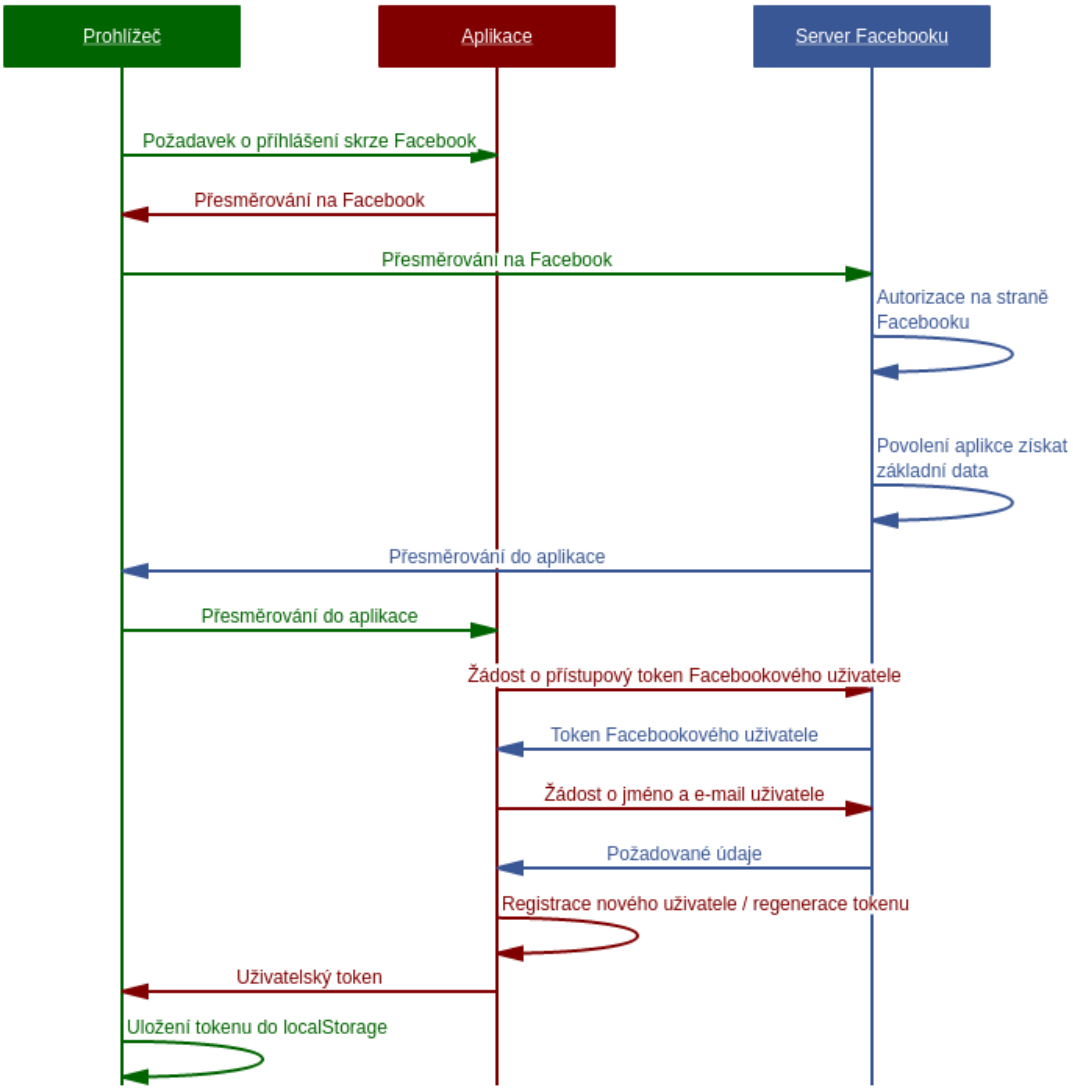
Do aplikace bylo třeba zakomponovat i nástroje pro administraci. Základní požadavek byl, aby si administrátor mohl prohlížet seznam uživatelů a jejich grafy. Při návrhu serveru se s administrací počítalo, nebylo tedy třeba API serveru upravovat. Při návrhu klientské části se na tuto možnost také myslelo, avšak důraz byl kladen na funkčnost aplikace samotné a administrace byla brána jako druhotný úkol. Výsledný návrh aplikace bohužel spojil získávání dat a jejich správu do stejných komponent. Komponenta, která data spravuje, si je i sama získává, a to na základě právě přihlášeného uživatele. Pokud bychom chtěli řešit administraci klasickou cestou, bylo by třeba získávání dat od komponent oddělit. To by byl ovšem poměrně velký zákrok, a tak jsem přišel s alternativním řešením.

V aplikaci nyní figurují dvě entity: přihlášený uživatel a aktivní uživatel. Pokud se v aplikaci pohybuje klasický uživatel – ne administrátor, je přihlášený uživatel shodný s aktivním. Pokud se v aplikaci pohybuje administrátor, může si na stránce s výpisem uživatelů zvolit jiného uživatele, za kterého se bude vydávat. Administrátor tedy může prohlížet grafy jiných uživatelů a dělat veškeré operace jako uživatel sám za sebe.

5.3 Autentizace a komunikace se serverem

Komunikace se serverem je realizovaná asynchronně pomocí technologie AJAX. Autentizace je realizovaná pomocí tokenů. Token uživatel získá skrze přihlašovací formulář nebo přihlášením přes třetí stranu (Facebook, GitHub, Gmail). Získaný token je uložen do *localStorage* prohlížeče. Token je vystaven na neomezenou dobu, avšak s každým přihlášením je obnoven. Není tedy možné být přihlášen (standardní cestou) na více zařízeních najednou.

Na obrázku 5.3 je možné nahlédnout na diagram autentizace pomocí Facebooku. Při této autentizaci je uživatel nejdříve přesměrován na stránky Facebooku. Na stránce s Face-



Obrázek 5.3: Diagram autentizace pomocí Facebooku

bookem se musí uživatel přihlásit a potvrdit aplikaci přístup ke svému profilu. Pokud je uživatel již přihlášen a povolení v minulosti již udělil, je tento krok pro uživatele v podstatě nepostřehnutelný. Uživatel je následně přesměrován do aplikace a ta zažádá Facebook o token, na základě kterého dostane přístup k profilu uživatele. Pomocí získaného tokenu si aplikace stáhne základní informace o uživateli. Na základě získaných údajů vyhledá aplikace uživatele ve své databázi. Pokud uživatele nenajde, automaticky jej zaregistruje. V obou případech uživateli přidělí nový token, který se uloží do prohlížeče.

Kapitola 6

Vyhodnocení

Vývoj aplikace probíhal v prostředí operačního systému Fedora 25 s prohlížečem Google Chrome verze 55. Aplikace byla, v kooperaci s panem doktorem Burgetem, úspěšně propojena s nástrojem FITLayout.

Aplikaci je důležité testovat v produkčním sestavení, ve kterém je provedena řada optimalizací. Testovací úloha zněla: registrace, sestavení grafu, odeslání úlohy na server, ověření výstupních dat. Při sestavování grafů je testována odezva rozhraní a celkový dojem z plynulosti práce.

6.1 Kompatibilita a výkon napříč prohlížeči

Vzhledem k použití relativně nových technologií a přístupů nelze očekávat vysokou míru kompatibility napříč spektrem prohlížečů, obzvláště pak u neaktuálních verzí.

Aplikace byla laděna primárně pro internetový prohlížeč Google Chrome verze 55, ve kterém funguje bez problémů. Stejně dobrých výsledků bylo dosaženo i v prohlížeči Opera 42. Google Chrome i Opera jsou postaveny na stejném jádře Blink s JavaScriptovým jádrem V8 [30][31][24], což bude důvod stejně dobrých výsledků.

Testování v prohlížeči Safari verze 9.1.1 dopadlo také bez problému.

Při testování v internetovém prohlížeči Firefox verze 51 byly objeveny problémy s výkonem. Problém s výkonem se projeví při posouvání prvků grafu po plátně. Zprvu jsem se domníval, že je problém spojen s využitím SVG, avšak posléze se ukázalo, že se projeví, kdykoliv je aplikace zaplavena akcemi. Bohužel se mi nepodařilo přesně určit v čem tkví jádro problému, avšak domnívám se, že problém pramení z *immutable* přístupu. Při práci s Reactem a Reduxem se dbá na to, aby stavové proměnné nebyly měněny, ale byly pokaždé vytvářeny nové. Tento přístup vyžaduje intenzivní recyklaci paměťového prostoru a právě ve chvílích, kdy je spuštěn garbage collector[1], aplikace ztrácí výkon. Tento předpoklad vznikl na základě profilování aplikace ve vývojářském režimu.

Podobné problémy s výkonem byly pozorovány i v prohlížeči Internet Explorer 11, kde se navíc objevili i drobné vizuální chyby.

[1] Nástroj, který automaticky uvolňuje již nepotřebné bloky paměti

6.2 Součinnost s nástrojem FITLayout

V první řadě bylo potřeba ověřit, zda je výstupní formát extrakční úlohy vhodně navržen. Toto testování probíhalo výměnou textových souborů s příklady extrakčních úloh. Jakmile byla potvrzena finální podoba formátu, přešlo se do fáze, kdy si nástroj FITLayout potřebná zadání stahoval přímo ze serveru skrze API rozhraní.

Propojení s nástrojem FITLayout bylo úspěšné. Nástroj je schopen si stáhnout zadání extrakční úlohy, toto zadání zpracovat a provést extrakci. Příklad formátu dat je možné vidět v ukázce 6.1.

```
{
  "graph":{
    "id":14,
    "values":{ "title":"Example", "url": "..."  },
    "nodes":[
      {
        "id":1490096524441,
        "values":{
          "uris":[
            "http://data.semanticweb.org/ns/swc/ontology#/
               Paper",
            "http://xmlns.com/foaf/0.1/Document"
          ],
          "title":"Paper\nswe:Paper\nfoaf:Document",
          "object":true
        }
      }
      ...
    ],
    "edges":[
      {
        "srcId":1490096774809,
        "dstId":1490096524441,
        "values":{
          "uris":[
            "http://purl.org/dc/elements/1.1/creator"
          ],
          "title":"dc:creator",
          "cardinality":{ "dst":true, "src":true }
        }
      }
      ...
    ]
  }
}
```

Ukázka kódu 6.1: Příklad datové reprezentace hrany

6.3 Další možný vývoj

Aplikaci je samozřejmě možné dále vylepšovat. Rád bych zde zmínil pár, dle mého názoru, nejzajímavějších vylepšení.

6.3.1 Implementace funkcí zpět a vpřed

Při sestavování extrakční úlohy se může stát, že omylem vyvoláme akci, kterou jsme nezamýšleli, a rádi bychom udělali krok zpět. Implementace této funkce by měla být díky architektuře aplikace snadná. Jak již bylo zmíněno rozhraní, je projekcí stavu. Pokud bychom chtěli cestovat časem zpět a vpřed, museli bychom jednotlivé nové stavy ukládat do kolekce a spravovat si ukazatel na aktuální stav. Čili místo jednoho objektu reprezentujícího stav grafu bychom měli celou kolekci s ukazatelem na právě aktuální. Akce zpět by poté modifikovala onen ukazatel tak, aby ukázal na stav dřívější. Avšak bylo by potřeba vymyslet systém, který by rozhodoval, které akce vytvoří nový stav do historie a které ne. Například akce, jež mění pozici uzlu na plátně, jsou nepřípustné, jelikož jich je velké množství ve velmi krátkém intervalu. Vhodné by bylo historii pozměnit pouze poslední akcí z celé sekvence.

6.3.2 Server-side rendering

V současné podobě aplikace funguje následujícím způsobem. Uživatel navštíví webovou stránku a stáhne si minimalistickou kostru HTML a JavaScriptový kód. Teprve až po interpretaci JavaScriptu se uživateli zobrazí relevantní obsah. Jelikož internetové vyhledávače jen zřídka kdy interpretují JavaScript, je pro ně obsah webu nedostupný. Avšak vzhledem k účelu naší aplikace to není až tak zásadní problém. Pokud bychom chtěli tento nedostatek odstranit, musíme přistoupit k takzvanému *server-side renderingu*. Server-side rendering spočívá v tom, že se aplikace interpretuje již na straně serveru a klientovi se posílá HTML kód po interpretaci. Webový prohlížeč tedy dostává smysluplný HTML kód, který si mohou vyhledávače indexovat [12]. Současně s HTML kódem je potřeba aplikaci poslat stav, který vznikl po interpretaci. Tento stav je poté zaveden do aplikace a běh aplikace pokračuje tam, kde na serveru skončil. Jak již z popisu vyplývá, je potřeba mít možnost JavaScript na straně serveru interpretovat. Vzhledem k využití platformy NodeJS to není problém. Interpretaci je dokonce možné provést i na serveru poháněném PHP, a to díky rozšíření V8Js [32].

Kapitola 7

Závěr

Navrhl jsem a implementoval aplikaci, která slouží jako uživatelské rozhraní k nástroji pro extrakci dat.

Při vývoji aplikace jsem použil nejnovější trendy z oblasti JavaScriptových aplikací. Pro tyto účely jsem prostudoval řadu nových technologií, knihoven a přístupů. Zejména jsem se soustředil na knihovnu React a její ekosystém.

Abych mohl uživatelské rozhraní vhodně navrhnout, seznámil jsem se se základy extrakcí informací z dokumentů a s nástrojem FITLayout. Na základě prostudovaných dokumentů jsem se rozhodl, že základem uživatelského rozhraní bude interaktivní graf. Dále jsem navrhl formát a rozhraní, přes které bude nástroj FITLayout s mojí aplikací komunikovat.

Aplikaci jsem otestoval napříč spektrem moderních webových prohlížečů. Ve spolupráci s panem doktorem Burgetem jsme úspěšně propojili aplikaci s nástrojem FITLayout.

Literatura

[1] Abramov, D.: *Redux*. [Online; navštíveno 6.3.2017].
URL http://redux.js.org/

[2] Berka, P.: *Inteligentní systémy*. Oeconomica, 2008, ISBN 978-80-245-1436-9.

[3] Brychta, F.: *Rámec pro extrakci informace z WWW*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2008.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=8263

[4] Burget, R.: *FITLayout: Web Page Analysis Framework*. [Online; navštíveno 28.3.2017].
URL http://www.fit.vutbr.cz/~burgetr/FITLayout/

[5] Burget, R.: Layout Based Information Extraction from HTML Documents. In *9th International Conference on Document Analysis and Recognition ICDAR 2007*, IEEE Computer Society, 2007, ISBN 0-7695-2822-8, s. 624–629.
URL http://www.fit.vutbr.cz/research/view_pub.php.en?id=8403

[6] Call-Em-All and contributors: *Material-UI:A Set of React Components that Implement Google's Material Design*. [Online; navštíveno 24.4.2017].
URL http://www.material-ui.com/#/

[7] Czaplicki, E.: *Elm: A delightful language for reliable webapps*. [Online; navštíveno 6.3.2017].
URL http://elm-lang.org/

[8] Deveria, A.: *Can I use... Support tables for HTML5, CSS3, etc*. [Online; navštíveno 11.4.2017].
URL https://caniuse.com/#search=svg

[9] Ecma International: *ECMAScript® 2015 Language Specification* . [Online; navštíveno 30.3.2017].
URL http://www.ecma-international.org/ecma-262/6.0/index.html

[10] Engelschall, R. S.: *ECMAScript 6: New Features: Overview & Comparison*. [Online; navštíveno 6.3.2017].
URL https://babeljs.io/docs/plugins/

[11] Facebook Inc.: *FLUX:APPLICATION ARCHITECTURE FOR BUILDING USER INTERFACES*. [Online; navštíveno 24.4.2017].
URL https://facebook.github.io/flux/

[12] Facebook Inc.: *React: A JavaScript library for building user interfaces*. [Online; navštíveno 6.3.2017].
URL `https://facebook.github.io/react/`

[13] Google Inc.: *Angular: One framework*. [Online; navštíveno 6.3.2017].
URL `https://angular.io/`

[14] Google Inc.: *Dart:webdev*. [Online; navštíveno 6.3.2017].
URL `https://webdev.dartlang.org/`

[15] Gregory Shehet, B. G. M. R. M. W., Andy Kogut: *MobX:Simple, scalable state management*. [Online; navštíveno 24.4.2017].
URL `https://mobx.js.org/`

[16] Hanson, J.: *Passport:Simple, unobtrusive authentication for Node.js*. [Online; navštíveno 24.4.2017].
URL `http://passportjs.org/docs`

[17] J. Reschke, greenbytes.: *The 'Basic' HTTP Authentication Scheme*. RFC 7617, Září 2015.
URL `https://tools.ietf.org/html/rfc7617`

[18] JS Foundation: *ESLint:The pluggable linting utility for JavaScript and JSX*. [Online; navštíveno 24.4.2017].
URL `http://eslint.org/`

[19] Koppers, T.: *Webpack:Module bundler*. [Online; navštíveno 24.4.2017].
URL `https://github.com/webpack/webpack`

[20] McKenzie, S.: *Babel is a JavaScript compiler*. [Online; navštíveno 15.3.2017].
URL `http://es6-features.org/`

[21] Microsoft: *TypeScript: Javascript that scales*. [Online; navštíveno 6.3.2017].
URL `https://www.typescriptlang.org/`

[22] Mozilla Developer Network and individual contributors: *MDN*. [Online; navštíveno 6.3.2017].
URL `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

[23] Node.js Foundation: *Express:Fast, unopinionated, minimalist web framework for Node.js*. [Online; navštíveno 24.4.2017].
URL `http://expressjs.com/`

[24] Opera Software ASA: *Dev.Opera*. [Online; navštíveno 17.4.2017].
URL `https://dev.opera.com/`

[25] Oracle Corporation and/or its affiliates: *Changes in MySQL 5.7.8 (2015-08-03, Release Candidate)*. [Online; navštíveno 18.3.2017].
URL `https://dev.mysql.com/doc/relnotes/mysql/5.7/en/news-5-7-8.html`

[26] Otto, M.: *Bootstrap:Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.* [Online; navštíveno 24.4.2017].
URL http://getbootstrap.com/

[27] R. Shekh-Yusef aj.: *HTTP Digest Access Authentication.* RFC 7616, Září 2015.
URL https://tools.ietf.org/html/rfc7616

[28] Sequelize contributors: *Sequelize:The Node.js / io.js ORM for PostgreSQL, MySQL, SQLite and MSSQL.* [Online; navštíveno 24.4.2017].
URL http://docs.sequelizejs.com/en/v3/

[29] Svátek, V.: *Ontologie a WWW.* 2002.
URL http://nb.vse.cz/~svatek/onto-www.pdf

[30] The Chromium Project: *Blink.* [Online; navštíveno 17.4.2017].
URL https://www.chromium.org/blink

[31] The Chromium Project: *Chrome V8.* [Online; navštíveno 17.4.2017].
URL https://developers.google.com/v8/

[32] The PHP Group: *V8 Javascript Engine for PHP: This PHP extension embeds the Google V8 Javascript Engine.* [Online; navštíveno 17.4.2017].
URL https://github.com/phpv8/v8js

[33] W3C: *Document Object Model (DOM) Level 3 Core Specification.* [Online; navštíveno 7.4.2017].
URL https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

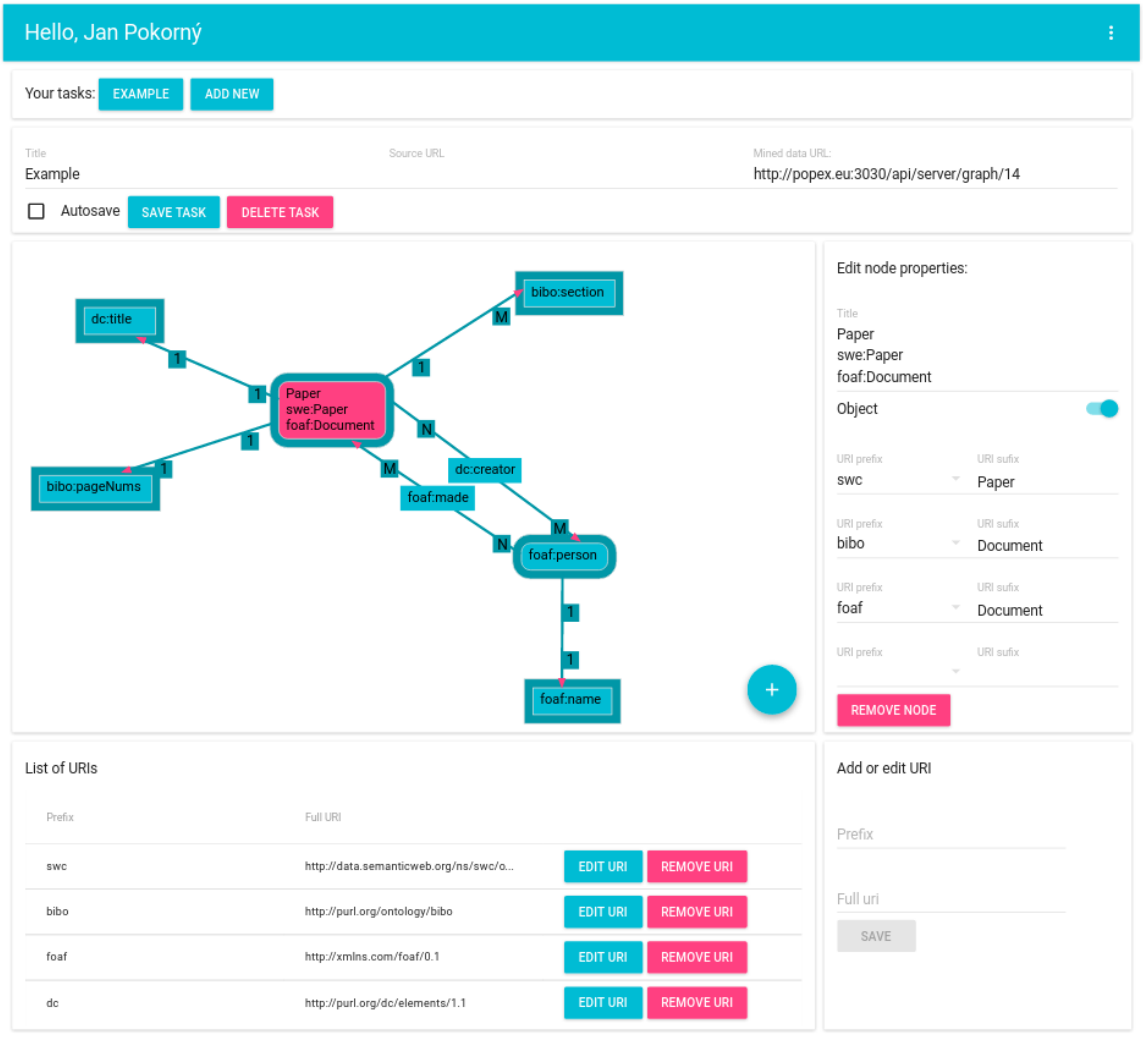
[34] You, E.: *VueJs: The Progressive JavaScript Framework.* [Online; navštíveno 6.3.2017].
URL https://vuejs.org/

[35] Zabriskie, M.: *Axios:Promise based HTTP client for the browser and node.js.* [Online; navštíveno 24.4.2017].
URL https://github.com/mzabriskie/axios

Přílohy

Příloha A

Snímek obrazovky aplikace



Obrázek A.1: Screenshot stránky se správou grafu

Příloha B

Obsah přiloženého CD

```
.
|-- backup_frontend_with_packages.zip // záloha s nainstalovanými balíčky
|-- backup_server_with_packages.zip // záloha s nainstalovanými balíčky
|-- docs // dokumentace
|-- frontend // klientská aplikace
|   |-- .babelrc
|   |-- dist // výsledné soubory po sestavení
|   |-- .eslintrc
|   |-- index.html
|   |-- node_modules
|   |-- package.json
|   |-- server.js
|   |-- src // zdrojové kódy
|   |-- webpack.config.js
|   `-- webpack.config.prod.js
`-- server // server
    |-- config.js // konfigurační soubor serveru
    |-- db.js
    |-- .eslintrc
    |-- node_modules
    |-- package.json
    |-- passport.js
    |-- public // statické soubory
    |-- routes
    |-- server.js
    `-- tools
```

Příloha C

Instalační manuál

C.1 Prerekvizita

Pro chod aplikace je třeba mít k dispozici MySQL databázi verze 5.7.8 a vyšší. Dále je nutné mít nainstalovaný JavaScriptový interpret NodeJS a balíčkovací nástroj Npm.

C.2 Sestavení frontendu

Pracujeme ve složce *frontend*. Nejdříve otevřeme a upravíme konfigurační soubor *src/config.js*.

Po editaci si stáhneme potřebné knihovny a aplikaci sestavíme:

```
[frontend] $ npm install
[frontend] $ npm run build
```

Po sestavení vygenerované soubory nakopírujeme do složky se serverem:

```
[frontend] $ cp dist/* ../server/public/static/
```

C.3 Spuštění serveru

Přesuneme se do složky *server*, kde opět provedeme editaci konfiguračního souboru *config.js*.

Nainstalujeme potřebné knihovny. Nastavím proměnnou *NODE_ENV* pro produkci. Nakonec server spustíme.

```
[server] $ npm install
[server] $ export NODE_ENV="production"
[server] $ npm start
```

Nyní si můžeme vytvořit administrátorský účet.

```
[server] $ npm run makeAdmin --email="" --password=""
```

C.4 Vývojářský režim

Pokud chceme aplikaci vyvíjet, spustíme si server obdobně jako při produkčním módu, avšak bez proměnné *NODE_ENV*. Poté ve složce *frontend* spustíme vývojový server.

```
[frontend] $ npm start
```