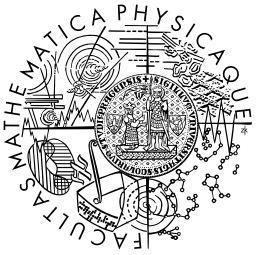


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Štěpán Stenchlák

Vizuální browser grafových dat

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové a datové inženýrství

Praha 2020

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

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

V ..... dne ....

Podpis autora

Poděkování.

Název práce: Vizuální browser grafových dat

Autor: Štěpán Stenchlák

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D., katedra

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Visual browser of graph data

Author: Štěpán Stenchlák

Department: Name of the department

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., department

Abstract: Abstract.

Keywords: key words

Obsah

Úvod

Na internetu je dnes možné najít téměř cokoli, kupříkladu stav počasí, encyklopedické informace, odborné publikace, jízdní řády a podobně. Tyto informace jsou uloženy jako webové stránky systému WWW (World Wide Web) a kterýkoli uživatel internetu má k nim přístup a může z nich čerpat.

Pro lidi je systém WWW vyhovující zdroj informací, nicméně narazíme na problém, pokud chceme tyto informace číst strojově. WWW totiž nepopisuje, jak by měly být informace na internetu prezentovány, každý publikovatel si je může zveřejnit jinak, například tabulkou, odrážkovým seznamem, nebo ve větách. Tyto informace jsou stále snadno čitelné pro člověka, ale obtížně čitelné pro stroj.

Možností řešení tohoto problému je zveřejňovat data i ve strojové podobně. Nabízí se například tabulky ve formátu CSV, nebo komplikovanější data ve formátu JSON a XML. Zde je již snadné číst data a dál je zpracovávat, ale programátor je stále nucen pochopit a přizpůsobit program na rozhraní těchto dat.

Konečným řešením se nabízí RDF.

0.1 Resource Description Framework

Resource Description Framework zkráceně RDF je rodina specifikací (tedy sada pravidel) která se používá na popis informací na internetu. RDF popisuje data jako graf, konkrétně vrcholy grafu jsou entity, jež ztvářňují nějaké věci (fyzické předměty, díla, myšlenky) a hrany tyto entity propojují a dávají jim vztahy.

Základem je takzvaný **statement** (česky trojice). Statement se skládá v tomto pořadí ze subjektu, predikátu a objektu přičemž subjekt a objekt jsou vrcholy (anglicky **nodes**) a predikát je orientovanou hranou jdoucí od subjektu k objektu.

Vrcholem v RDF může být IRI, literál, nebo prázdný uzel.

Jako vrchol IRI (Internationalized Resource Identifier) se rozumí vrchol jež přiřazuje entitě nějaký identifikátor ve tvaru IRI. Tímto jsme schopni v rámci WWW identifikovat různé entity a pracovat s nimi napříč různými zdroji. Ku příkladu Karel Čapek je v rámci Wikipedie jednoznačně identifikován jako `https://www.wikidata.org/wiki/Q155855`. IRI kromě identifikátoru nenese žádné další informace.

Literálem je pak uzel nesoucí nějakou hodnotu. Může se jednat o číslo popisující věk osoby, její jméno atp. V rámci RDF kromě hodnoty má literál i svůj typ, jež je opět vyjádřen pomocí IRI. Uveďme kupříkladu `http://www.w3.org/2001/XMLSchema#integer` jež vyjadřuje obecně celé číslo. Speciálním typem je `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString` který vyžaduje ještě laguage tag a popisuje text v nějakém konkrétním jazyce.

Literály nám dávají možnost přidat IRI uzlům hodnoty a podporují právě i multijazyčnost.

Nakonec, hrana je opět popsána pomocí IRI jež reprezentuje typ vztahu.

0.1.1 Turtle jazyk

RDF graf může být popsán pomocí Turtle jazyka[1]. Ten je využit při zápisu konfigurací a metakonfigurací, jež jsou popsány dále v tomto dokumentu.

Trojice (statementy) se zapisují jako tři slova oddělená mezerami a zakončená tečkou. Chceme-li zapsat IRI, musíme jej dát do špičatých závorek `<>`. Na začátku dokumentu lze kromě dalších věcí definovat i prefixy, které pak umožňují zkrátit zapisované IRI do namespacu a zbytku `namespace:rest`.

Pokud se nám u statementů opakuje subjekt a predikát, můžeme jednotlivé objekty oddělovat čárkou `,`. Obdobně, pokud se nám opakuje jen subjekt, můžeme dvojice predikát-objekt oddělovat středníkem `;`.

Uveďme několik příkladů RDF grafu.

Příklad. Statement „Karel Čapek se narodil 9. ledna 1890“ vyjádřený v rámci wikidat.

```
@prefix wd: <https://www.wikidata.org/wiki/> .
@prefix wdt: <https://www.wikidata.org/wiki/Property:> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

wd:Q155855 wdt:P569 "1890-01-09"^^xsd:dateTime .
```

Kód popisuje jeden statement kdy se Karlu Čapkovi `https://www.wikidata.org/wiki/Q155855` přiřadí přes property datum narození `https://www.wikidata.org/wiki/Property:P569` literál s jeho datem narození, jež má typ `http://www.w3.org/2001/XMLSchema#dateTime`.

Jak lze vidět z příkladu, statementy nemusí odkazovat jen na svůj dataset, ale mohou odkazovat i mimo něj. To nám dovoluje stavět již na existujících datasetech a jednoduše se na ně okazovat přes IRI. Můžeme tak mít vlastní knihovní databázi jež ke knize přiradí autora z datasetu Wikidat. Tímto jsme propojili dva datasety, což nám umožní se nad oběma dotazovat. Příkladem takového dotazu by mohlo být „Chci seznam knih v naší knihovně, jež byly napsány českými autory.“ Takový dotaz pak současně prohledá dva různé datasety a vrátí očekávané výsledky.

Uveďme ještě příklad statementu, jež se odkazuje na další entity.

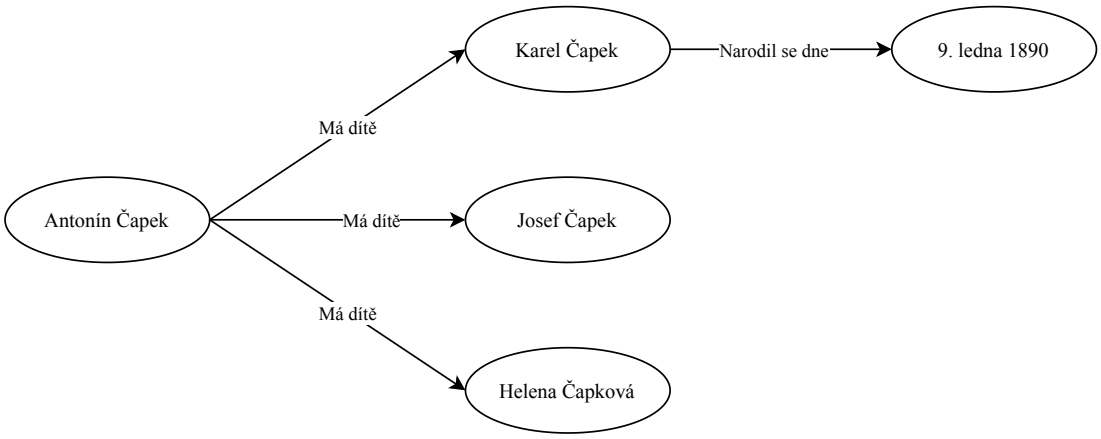
Příklad. Statement „Děti Antonína Čapka jsou Karel, Josef a Helena“ vyjádřený v rámci wikidat.

```
@prefix wd: <https://www.wikidata.org/wiki/> .
@prefix wdt: <https://www.wikidata.org/wiki/Property:> .

wd:Q6657059 wdt:P40 wd:Q155855 ,
                    wd:Q454568 ,
                    wd:Q4532606 .
```

Pro úplnost zmiňme ještě například následující graf popisující vztahy mezi lidmi, které jsou vyjádřeny ontologií FOAF (friend of a friend). Pokud by všechny zdroje popisující lidi využívali tuto ontologii, měli bychom jednotný interface jak přistupovat k lidským vstahům.

[1] `https://www.w3.org/TR/turtle/`



Obrázek 1: Příklad grafu který získáme z předešlých dvou ukázek

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix example: <http://example.org/> .

<http://example.org/Jan-Novák> foaf:name "Jan Novák" .
<http://example.org/Ondřej-Novák> foaf:name "Ondřej Novák" .

example:Pavel-Novotný foaf:name "Pavel Novotný" ;
                      foaf:knows example:Jan-Novák ,
                                 example:Ondřej-Novák .
```

1. Vue.js framework

Klientská část aplikace je postavena nad Vue.js frameworkem, jež je populární JavaScriptový framework na stavbu uživatelských rozhraní. Protože některé z jeho funkcionalit byly použity v klíčových částech aplikace, je nutné čtenáře obeznámit alespoň se základním principem fungování frameworku.

1.1 Vuex

Vue.js framework, podobně jako konkurenční React[1] (Facebook) nebo Angular[2] (Google), využívají principu sledování stavu aplikace (jejich dat) pro automatickou změnu DOMu webové stránky. V praxi to znamená, že programátor může velice snadno napsat kód, který generuje uživatelské rozhraní na základě dat, která mohou být libovolně měněna bez nutnosti řešit problém, zda ke změně vůbec došlo a které části aplikace mají být o změně stavu informovány. Ve Vue tuto funkcionalitu zastává právě Vuex[3], jež je možný používat samostatně.

Vuex drží stav aplikace jako jeden objekt (tedy slouží jako centrální úložiště dat pro celou aplikaci). Tento objekt se nazývá **store**. Změny ve storu mohou být sledovány Vuexem pro vykonání libovolných akcí, například překreslení textu na stránce, jež byl vykreslen Vue frameworkem.

Vrátíme-li se k původnímu příkladu, programátorovi stačí přiřadit do proměnné, jež je spravovaná Vuexem, novou hodnotu a Vuex se postará o zavolání všech komponent, které tuto proměnnou využívají a tyto komponenty na stránce překreslí původní hodnotu na novou. Překreslení přitom proběhne až poté, co skončí průběh aktuální funkce. Tohoto je docíleno pomocí `Window.requestAnimationFrame()`. Díky tomuto můžeme stav v rámci průběhu jedné funkce modifikovat vícekrát se skoro nulovým dopadem na celkový výkon aplikace.

1.1.1 Computed properties

Kromě této funkcionality Vuex nabízí takzvané **gettery**, jež jsou ve Vue frameworku nazývány jako **computed properties**. Jedná se o funkce, které využívají data ze storu pro výpočet dat nových. Výhoda takovýchto getterů je ta, že Vuex dokáže výsledky těchto funkcí cachovat a přepočítává je pouze tehdy, změní-li se data původní. Interně gettery fungují tak, že při zavolání klientské funkce Vuex sleduje které části storu byly dotázany a ty pak sleduje na změnu jež invaliduje cache konkrétního getteru. Při příštím požadavku na hodnotu se pak klientská funkce volá znovu a celá operace se opakuje.

Tyto computed properties jsou v aplikaci využívány často. Kupříkladu funkce, která počítá, zda je sousední uzel vybrán. Na takovouto hodnotu se v aplikaci mohu ptát libovolně krát, ale počítá se pouze tehdy, když se množina sousedních uzlů vrcholu změní, nebo se změní právě označení uzlu z množiny.

[1] `https://reactjs.org/`
[2] `https://angularjs.org/`
[3] `https://vuex.vuejs.org/`

1.1.2 Watchers

Ve Vue lze využívat i **watch**ery, které umožňují registraci callbacku na změnu určité proměnné ve stavu. Watchery má smysl využívat tam, kde již data přestávají být spravována Vue frameworkem, tedy u knihoven třetích stran. Watchery, podobně jako překreslení komponent, jsou volány až po skončení probíhající funkce.

1.1.3 Změna stavu

Jak již bylo zmíněno, stav lze měnit přiřazením do proměnné, popřípadě voláním metod jako `.push()` na poli. Vuex dokáže tyto změny sledovat nahrazením původní proměnné (máme na mysli položku objektu) JavaScriptovým setterem. U polí dojde k obalení metody `.push()` jež registruje změnu stavu.

Tento přístup má několik nevýhod, které se promítly i při vývoji klientské aplikace:

- Vue není plně kompatibilní s novými **ES6 kontejnery** `Map` a `Set` a proto jsou v aplikaci používány jen v rámci lokálních proměnných a mapa je nahrazena klasickým objektem.
- Protože je v JavaScriptu nemožné sledovat **vytvoření nové property objektu**, musí programátor v tomto případě volat ručně `Vue.set(target, propertyName/index, value)` a `Vue.delete`, popřípadě vytvořit nový objekt který nahradí ten původní.
- Je důležité mít na paměti, že data ve storu již nejsou původními objekty v pravém slova smyslu, ale veškeré fieldy a metody u polí byly nahrazeny, jak je zmíněno výše. Proto předání objektů a polí ze storu knihovnám třetích stran je nutné ošetřit **oklonováním objektu**, jinak může dojít k zaseknutí aplikace.
- Instance tříd **knihoven třetích stran může zaseknout aplikaci**, pokud ji uložíme do storu. Bohužel, tohoto je velmi snadné ve Vue frameworku dosáhnout omylem. Problém je rozebrán v následující kapitole.

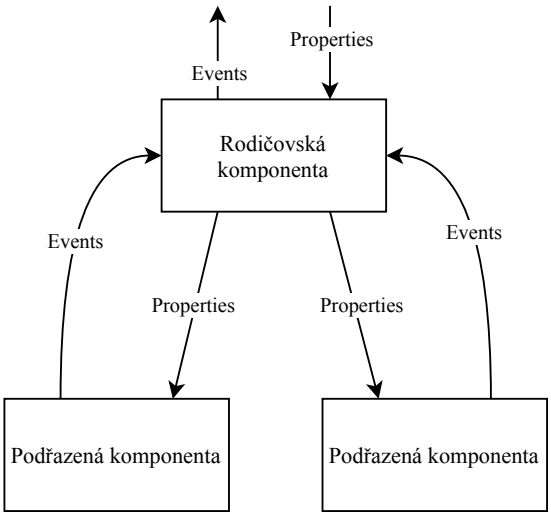
1.2 Vue framework

Vue framework využívá takzvané komponenty. Komponentou se rozumí prvek na stránce se kterým má smysl pracovat samostatně. Každá komponenta má vlastní HTML, CSS a JS. Komponenty se mohou do sebe zanořovat a vytvářet tak větší komponenty z menších. Příkladem může být komponenta *seznam* jež dokáže na stránku vykreslit senznam prvků. Takováto komponenta by pak mohla mít podkomponenty jako *prvek seznamu*.

Každé komponentě lze předat data formou **properties** přičemž komponenta na základě těchto dat může vyrobit pod sebou další komponenty a předat jim část dat, která dostala.

Tyto předávána data jsou právě data ze storu. Vue framework doporučuje, aby data co koponenta dostane formou properties neupravovala přímo, ale místo toho posílala události rodičovské komponentě, která data upraví. Ve své práci jsem se

rozhodl toto doporučení ignorovat, neboť by tímto vzrostla náročnočnost na správu aplikace.



Obrázek 1.1: Doporučený způsob komunikace mezi komponentami ve Vue frameworku

1.3 Loaders

Kód Vue komponent se zapisuje do souborů s příponou `.vue`. Při sestavování aplikace se pak použije `vue-loader` který ze souboru vyextrahuje zvlášť CSS, JS a HTML a ty předá dál na zpracování. HTML kód komponent není ve skutečnosti pravý HTML. Jedná se nadstavbu umožňující psát speciální značky, jež rozhodují kolikrát a jestli vůbec se tag na stránce vyrenderuje. Tato HTML nadstavba je pak předána `vue-template-compiler` který vyrobí optimalizovaný JS kód jež renderuje HTML na základě stavu komponenty.

Příklad. Ukázka jednoduché Vue komponenty `IndexedList` která má parametr `list` očekávající pole stringů. Tato komponenta vypíše pole v odrážkovém seznamu ve formě `index: hodnota`. Komponenta se sama stará o překreslení DOMu, když se změní data. Komponentu můžeme v jiné komponentě použít vložením `<indexed-list :list="inputData"/>` kde `inputData` je proměnná obsahující pole stringů.

```
<template>
  <ul>
    <li v-for="(item, index) in list" :key="index">
      {{index}}: {{ item }}
    </li>
  </ul>
</template>

<script lang="ts">
  import {Component, Prop} from "vue-property-decorator";
  import Vue from "vue";
  @Component
  export default class IndexedList extends Vue {
    @Prop() private list: string[];
  }
</script>

<style scoped lang="scss">
  ul {
    color: red;
  }
</style>
```

Scoped styly

Můžeme si povšimnout `scoped` stylů v ukázce. Vue má mechanismus, že styly které zde nastavíme se aplikují jen na tuto komponentu. Nastavením červené barvy na seznam jsme tedy skutečně nastavili červenou barvu jen této komponentě a ostatní seznamy jsou netknuté. Toto má nespornou výhodu pokud pracujeme s velkým množstvím komponent a hrozilo by, že bychom museli používat složitě pojmenované css třídy aby nedošlo ke kolizi.

Pokud bychom chtěli ovlivnit styly vnořených komponent a máme nastavené scoped styly, musíme použít pseudoselektor `::v-deep`, kupříkladu `.actions ::v-deep .v-input-selection-controls`. Tohoto je hodně používáno poukud je potřeba upravit styly Vuetify frameworku (viz dále).

2. Požadavky na systém

Tato kapitola shrnuje původní a později přidané požadavky na systém a jeho základní funkcionalitu.

Cílem bylo vyrobit webovou aplikaci jež by uměla procházet data v RDF databázích podle předem navolenýcyh konfiguracích. Procházením se myslí postupné objevování nových uzlů grafu s pomocí již existujících uzlů. Konfigurací se pak rozumí konkrétní způsob, jak mohou být data procházena a jak jsou vizualizována. Konfigurace jsou také uloženy v RDF databázi a je tedy možné používat konfigurace z jiných datových zdrojů a naopak.

2.1 Konfigurace

V následujícím vysvětlení jsou použity tyto RDF namespacy:

Prefix	IRI
browser	`https://linked.opendata.cz/ontology/knowledge-graph-browser/`
dct	`http://purl.org/dc/terms/`

2.1.1 Metakonfigurace

Metakonfigurace je skupina pro další konfigurace a metakonfigurace. Díky tomuto uživatel může procházet desíkty různých konfigurací uspořádaných do složek obdobně jak v souborovém systému na počítači. Metakonfigurace má tyto vlastnosti:

- `dct:title` Název metakonfigurace *(je možné zadat ve více jazycích)*
- `dct:description` Širší popis, co daná metakonfigurace obsahuje *(je možné zadat ve více jazycích)*
- `browser:image` Odkaz na URL soubor s obrázkem reprezentujícím metakonfiguraci. Obrázek je pak zobrazen v aplikaci při procházení konfigurací.
- `browser:hasMetaconficuration` Metakonfigurace, které spadají pod tuto metakonfiguraci.
- `browser:hasConfiguration` Konfigurace, které spadají pod tuto metakonfiguraci. *(popsáno dále)*

Metakonfigurací může být například „Wikidata“ nebo „Otevřená data ČR“.

2.1.2 Konfigurace

Konfigurací se rozumí uzel v RDF grafu který popisuje, jak by měla aplikace procházet datasety. Aktuálně může mít konfigurace tyto vlastnosti:

- `dct:title` Název konfigurace *(je možné zadat ve více jazycích)*
- `dct:description` Širší popis, čeho je možné s konfigurací dosáhnout *(je možné zadat ve více jazycích)*
- `browser:hasVisualStyleSheet` Určuje, jak mají uzly v aplikaci vypadat. *(popsáno dále)*
- `browser:startingNode` Doporučený uzel nebo uzly, kde začít s procházením grafu.
- `browser:resourceUriPattern` Regulární výraz popisující, jak by mělo vypadat IRI uzlu. Používá se v aplikaci jako nápověda uživateli, zda zadal správné IRI ještě než se pošle požadavek.
- `browser:hasViewSet` View sety. *(popsáno dále)*
- `browser:autocomplete` JSON soubor se seznamem RDF uzlů podle kterých probíhá hledání. *(popsáno dále)*

Příkladem konfigurace může být kupříkladu procházení slavných osobností na Wikidatech. Takováto konfigurace pak umožňuje uživateli chodit po uzlech reprezentující slavné osobnosti a dotazovat se například na filmy které natočily a knihy, které napsaly.

2.1.3 ViewSet

View set reprezentuje skupinu pohledů. Pravý smysl pohledů pochopí čtenář dále v textu. View set má následující vlastnosti:

- `dct:title` Název view setu *(je možné zadat ve více jazycích)*
- `browser:hasView` Pohledy které patří pod tento view set. *(popsáno dále)*
- `browser:hasDefaultView` Výchozí pohled ze seznamu výše.
- `browser:hasCondition` todo
- `browser:hasDataset` todo

2.1.4 View

View (česky pohled) je způsob, jak můžeme pohlížet na konkrétní uzel v RDF grafu. Uzel totiž může mít obecně více vlastností současně, obdobně jako jedna osoba může být současně spisovatel, režisér a herec. V takovém případě bychom mohli mít tři různé pohledy na jeden uzel a uživatel si může vybírat, jestli ho zajímá jeho herecká, nebo spisovatelská kariéra. View má následující vlastnosti:

- `dct:title` Název pohledu *(je možné zadat ve více jazycích)*
- `dct:description` Popis pohledu *(je možné zadat ve více jazycích)*
- `browser:hasExpansion` Odkaz na expanzi - určuje jaké uzly lze získat z daného uzlu *(popsáno dále)*
- `browser:hasPreview` Odkaz na preview - určuje jaká data se mají získat pro ostylování konkrétního uzlu *(popsáno dále)*
- `browser:hasDetail` Odkaz na detail - určuje která data se zobrazí v detailu konkrétního uzlu *(popsáno dále)*

2.1.5 Expansion

Expansion popisuje jak lze daný uzel expandovat, tedy jedná se o operaci kdy se stahují nové uzly jež jsou nějak příbuzné expandovanému uzlu. Expanze vrací graf, tedy expandované uzly nemusí být přímými sousedy expandovaného. Jako expanzi si můžeme představit například „Zobraz všechny knihy co napsala daná osoba“. Expanze formálně partří k pohledu (view).

- `dct:title` Název expanze *(je možné zadat ve více jazycích, aktuálně se nepoužívá)*
- `browser:hasDataset` Popisuje dataset vůči kterému se dotazuje na data. *(popsáno dále)*
- `browser:query` Popisuje SPARQL dotaz který bude spuštěn na endpointu datasetu.

2.1.6 Preview

Preview popisuje která data v rámci daného pohledu (view) popisují daný uzel. Popisem se myslí taková data, kretá mají na svědomí stylování uzlu. V aplikaci je možné mít uzly různých barev a tvarů, to právě popisuje preview. Obdobně jako expanze, preview patří ke konkrétnímu pohledu.

Preview má stejné vlastnosti jako expanze. `browser:query` v tomto případě popisuje SPARQL dotaz, který vrátí graf obsahující daný uzel a jeho literály a z těchto literálů bude sestaven preview.

Konkrétně pro preview je predikát `browser:class` považován na třídy uzlu a ty jsou nastaveny jako třídy v knihovně Cytoscape[1], která je využíváná v klientské části na krelsení grafů.

2.1.7 Detail

Detail je poslední z trojice a poskytuje dodatečné informace k uzlu. Může se jednat o literály které nemá smysl v dané konfiguraci vykreslit do grafu jako uzly, proto budou zobrazeny v bočním panelu aplikace.

Detail má stejné vlastnosti včetně `browser:query` jako preview.

[1] `https://js.cytoscape.org/`

2.1.8 Dataset

Dataset popisuje SPARQL endpoint vůči kterému probíhá dotazování na data.

- `dct:title` Název datasetu *(aktuálně se nepoužívá)*
- `void:sparqlEndpoint` URL adresa SPARQL endpointu na kterou se posílají SPARQL dotazy
- `browser:accept` todo

2.1.9 Visual style sheet

Popisuje jakž styl mají mít uzly v dané konfiguraci. Popis je založen na datech pocházejících právě z **preview**. Forma zápisu stylů aktuálně odpovídá stylům jaké používá knihovna Cytoscape.

Visual style sheet má pouze jednu vlastnosti

- `browser:hasVisualStyle` Jedno pravidlo jak se má provést stylování, odkazuje na **Visual style**.

Visual style

Visual style má pak:

- `browser:hasSelector` Selector pro Cytoscape knihovnu, jež vybírá na které uzly nebo hrany daný styl bude aplikován.
- `browser:*` Jednotlivá pravidla jak má být daný uzel nebo hrana ostylována. Používají se přesně ty názvy, které používá knihovna Cytoscape, kupříkladu `browser:border-width` nebo `browser:background-color`.

2.2 Aplikace

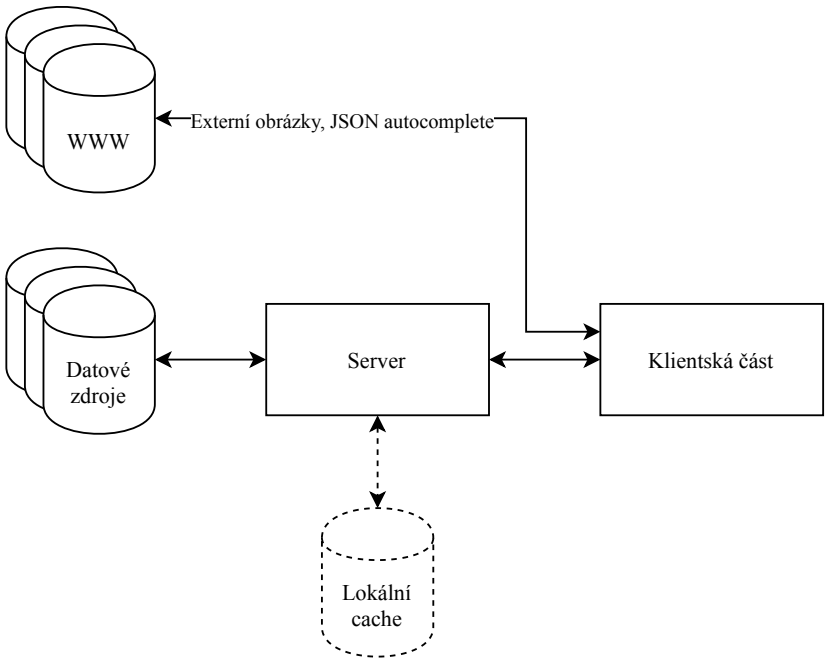
Bylo zamýšleno, že klientská aplikace bude komunikovat se serverovou částí, která bude stahovat data z různých datových zdrojů a výsledky posílat zpět klientské aplikaci. Ta v případě potřeby bude stahovat přímo z internetu obrázky, pokud na ně uzly odkazují. Idea rozdělení klienta a serveru byla v tom, že v budoucnu může být na serveru implementováno chachování, aby aplikace byla plynulejší a neprobíhalo zatěžování datových zdrojů.[2]

Chachování aktuálně implementováno ještě není, server je tedy kompletně bezestavový.

2.2.1 Server

Převážná část serveru byla napsána mým vedoucím a sloužila jako forma specifikace na klientskou část. Server bude podrobněji rozebrán v další kapitole.

[2]Při příliš rychlém stahování z Wikidat lze bohužel snadno dostat IP ban, tedy alespoň pro Wikipedii je cachování nezbytné.



Obrázek 2.1: Komunikace mezi klientem, serverem a datovými zdroji. Cachování na serveru ještě implemontováno není.

2.2.2 Klient

Požadavkem bylo vyrobit webovou aplikaci schopnou komunikovat se serverem a uživateli vizuálně zobrazovat uzly a hrany na obrazovce. Mezi původními požadavky na software bylo (požadavky jsou zkráceny):

1. Uživatel je schopen vybrat IRI konfigurace.
2. Uživatel je schopen vybrat IRI stylesheetu podle kterého budou obarveny a jinak zvýrazněny uzly.
3. Uživatel je schopen přidat uzel na základě jeho IRI a zobrazit si k němu detail.
4. Pokud uživatel klikne na uzel, zobrazí se mu nabídka s detailem a možností další expanze.
5. *sjednoceno do 4*
6. Uživatel může uzel skrýt. V menu je možnost zobrazit skryté uzly.
7. Dvojklikem na uzel se provede expanze podle aktuálního pohledu. Tu je možné provést také pomocí požadavku 4.
8. Uzly je možné filtrovat podle grafových vlastností.
9. Uzly je možné filtrovat podle sémantických vlastností.
10. Uživatelské rozhraní je multijazyčné.
11. Graf je možné uložit do souboru a poté jej ze souboru obnovit.
12. Uživatel může uzel z grafu smazat.
13. Uživatel může přidávat uzly pomocí autocomplete z před připraveného JSON souboru.

Časem byly do aplikace doplněné další požadavky jako

- Volba různých layoutů.
- Uzly mohou být seskupovány do skupin.
- Podpora metakonfigurací.

3. Server

Jak již bylo zmíněno dříve, převážná část serveru byla napsána mým vedoucím Martinem Nečaským a sloužila i jako technická specifikace pro klientsou část. Server je napsaný v JavaScriptu a běží v Node.js[1]. Server je bezestavový a odpovídá na požadavky zmíněné dále.

3.1 Jazyková podpora

Server jsem částečně přepsal, aby podporoval dotazování na data z více světových jazyků. Některé požadavky přijímají parametr `languages` obsahující čárkou `,` oddělené ISO 639-1 jazykové kódy. Server pak vrací objekty jejíž klíčem je jazykový kód a hodnotou daný překlad do jazyku. Pokud překlad neexistuje, hodnotou je `undefined`. V případě, že na všechny jazyky bylo vráceno `undefined`, server se pokusí přidat další jazyk, který existuje. Který jazyk takto bude vybrán není určeno.

Příklad. Pro `languages=cs,en` může server vrátit například

```
{
    cs: undefined,
    en: "Kankakee County"
}
```

ale pokud nezná překlad ani do češtiny, ani do angličity, může vrátit

```
{
    cs: undefined,
    en: undefined,
    sk: "Jazero Beňatina"
}
```

3.2 API

Server vrací data ve formátu JSON, požadavky jsou posílány metodou GET a parametry jsou kódovány do URL adresy.

- `/metaconfiguration`, **parametry:** `iri` a `languages`
  Vrátí informace o metakonfiguraci zadané podle `iri`.
  Vrátí všechna data (viz kapitola 2.1.1) o metakonfiguraci, veškerá data o dceřiných konfiguracích (viz kapitola 2.1.2) a základní data o dceřiných metakonfiguracích (vše kromě seznamu konfigurací a metakonfigurací).

- `/configuration`, **parametry:** `iri` a `languages`
  Vrátí informace o konfiguraci zadané podle `iri`.
  Vrátí stejná data jako `/metaconfiguration` o svých sceřiných konfiguracích.
  Toto volání se používá pouze když uživatel ručně zvolí IRI konfigurace, v opačném případě si aplikace vystačí s voláním `/metaconfiguration`.

[1] `https://nodejs.org/`

- `/stylesheet`, **parametry:** `stylesheet`
  Vrátí kompletní visual style sheet (viz kapitola 2.1.9) na základě jeho IRI jako parametr `stylesheet`.

- `/view-sets`, **parametry:** `config` a `resource`
  Vrátí seznam možných view setů (viz kapitola 2.1.3) které odpovídají uzlu s IRI `resource` při dané konfiguraci `config`.

- `/preview`, **parametry:** `view` a `resource`
  Vrátí data z dotazu preview (viz kapitola 2.1.6) na uzel s IRI `resource` při daném pohledu `view`.

- `/detail`, **parametry:** `view` a `resource`
  Vrátí data z dotazu detail (viz kapitola 2.1.7) na uzel s IRI `resource` při daném pohledu `view`.

- `/expand`, **parametry:** `view` a `resource`
  Vrátí expandované uzly (viz kapitola 2.1.5) pro uzel s IRI `resource` při daném pohledu `view`. Tyto expandované uzly již obsahují data o detailu a tedy není třeba žádného dalšího volání.

4. Implementace

5. Uživatelská dokumentace

6. Možnosti rozšíření

Anděl (2007, str.29)

Závěr

Seznam použité literatury

Anděl, J. (2007). *Základy matematické statistiky.* Druhé opravené vydání. Matfyzpress, Praha. ISBN 80-7378-001-1.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha