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 DATABÁZE UDÁLOSTÍ

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ HANZLÍK**
AUTHOR

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

BRNO 2018

Zadání bakalářské práce/20645/2017/xhanzl11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Hanzlík Tomáš**
Obor: Informační technologie
Téma: **Webové uživatelské rozhraní databáze událostí**
Web Interface for a Database of Events
Kategorie: Web

Pokyny:

1. Prostudujte existující technologie pro tvorbu grafických webových rozhraní pro datové zdroje s rozhraním REST.
2. Seznamte se s existující infrastrukturou systému pro sběr událostí ze sociálních sítí a jeho aplikačním rozhraním.
3. Navrhněte architekturu aplikace poskytující grafické uživatelské rozhraní k danému systému. Zaměřte se na možnost zpracování velkého množství dat.
4. Implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte testování výsledné aplikace na reprezentativní množině dat.
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

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 práce obsahuje návrh a implementaci webové aplikace, sloužící jako rozhraní mezi uživatelem a daty ze sociálních sítí. Ve webovém rozhraní je kladen důraz na zobrazení velkého množství dat a přehledného procházení mezi nimi pomocí filtrů. Je sestavena z JavaScriptové klientské části napsané ve Vue.js a Nuxt.js a serverové části v Django REST frameworku, která slouží jako zdroj dat.

Abstract

This work includes the design and implementation of a web application that serves as an interface between users and data aggregated from social networks. In the web interface, emphasis is placed on displaying large amounts of data and browsing through filters. It consists of the JavaScript client part written in Vue.js and Nuxt.js and the server part of the Django REST framework that serves as a data source.

Klíčová slova

JavaScript, VueJS, Vuex, Django, REST API, Docker, PostgreSQL, SPARQL, RDF4J, uživatelské rozhraní, velká data

Keywords

JavaScript, VueJS, Vuex, Django, REST API, Docker, PostgreSQL, SPARQL, RDF4J, user interface, big data

Citace

HANZLÍK, Tomáš. *Webové uživatelské rozhraní databáze událostí*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Webové uživatelské rozhraní databáze událostí

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.

.....
Tomáš Hanzlík
17. května 2018

Poděkování

Rád bych poděkoval panu doktoru Radkovi Burgetovi za ochotu při konzultacích a vedení bakalářské práce. Také bych rád poděkoval komunitě, která se podílela na vývoji použitých technologií v této práci.

Obsah

1 Úvod **3**

2 Analýza **4**
2.1 Resource Description Framework (RDF) . . . 4
2.2 TimelineAnalyzer . . . 4
2.3 Popis vstupních dat . . . 5
2.4 Přístup k datům . . . 6
2.5 Analýza požadavků . . . 6

3 Související technologie **8**
3.1 Serverová část (backend) . . . 8
3.1.1 REST rozhraní . . . 8
3.1.2 Django REST framework . . . 8
3.1.3 Autentizace pomocí JSON Web Tokenu (JWT) . . . 9
3.1.4 Perzistentní data s PostgreSQL . . . 10
3.2 Klientská část (frontend) . . . 10
3.2.1 HyperText Markup Language . . . 10
3.2.2 Cascading Style Sheets . . . 10
3.2.3 Javascript . . . 12
3.2.4 Vue.js . . . 12
3.2.5 NuxtJs . . . 14
3.3 SPARQL . . . 15
3.4 Docker . . . 15

4 Návrh aplikace **16**
4.1 Architektura systému . . . 16
4.2 Grafické rozhraní . . . 18
4.3 Databáze . . . 19

5 Implementace **21**
5.1 Server . . . 21
5.1.1 Adresářová struktura . . . 21
5.1.2 Databázové modely . . . 22
5.1.3 Autentizace a autorizace . . . 22
5.1.4 SPARQL proxy . . . 23
5.1.5 Spuštění . . . 23
5.2 Klient . . . 23
5.2.1 Adresářová struktura . . . 23

5.2.2 Centrální správa stavu 24
5.2.3 Implementace důležitých částí 25
5.2.4 Spuštění 27
5.3 Spuštění všech částí aplikace jako celku 27

6 Zhodnocení **28**
6.1 Testování serverové části 28
6.2 Testování uživatelského rozhraní 28
6.3 Možná rozšíření 29

7 Závěr **30**

Literatura **31**

Přílohy **33**

A Obsah přiloženého CD **34**

B Konečný vzhled aplikace **35**
C. Albert Thompson Home Papers Classes Tools FAQ Online Latex Formatter

Kapitola 1

Úvod

V dnešní době je vývoj webových aplikací poměrně komplexní téma. Už nestačí pouze staticky zobrazit požadovaná data, ale poskytnout uživateli i interaktivní rozhraní. Webové aplikace jsou multiplatformní, nevyžadují žádné lokální uložiště a k jejich spuštění stačí mít pouze webový prohlížeč. Díky těmto vlastnostem a díky novým technologiím dokážeme psát webové aplikace, které mohou ve velkém počtu případů kompletně nahradit aplikace desktopové.

Tato bakalářská práce se zabývá návrhem a implementací JavaScriptové webové aplikace s intuitivním grafickým rozhraním pro zobrazení a procházení velkého množství dat souvisejících se sociálními sítěmi. Data jsou výstupem nástroje TimelineAnalyzer a aplikace má sloužit jako rozhraní mezi těmito daty a uživateli, kteří jsou s nimi a s nástrojem seznámeni.

Obsah práce je rozdělen do několika logických kapitol včetně téhle, po níž následující druhá kapitola 2 obsahuje popis nástroje TimelineAnalyzer a strukturu vstupních dat. Dále jsou v ní také sepsány a analyzovány požadavky na výslednou aplikaci.

V další kapitole 3 je možné dočíst se o technologiích souvisejíc s vývojem této aplikace. V kapitole čtvrté 4 je navrhnuta celková architektura aplikace na základě analýzy požadavků. Architektura je zde rozdělena do dvou hlavních částí, které jsou klientská a serverová část. Navíc je zde popsán i návrh uživatelského rozhraní a schéma databáze.

Pátá kapitola 5 pojednává o implementaci návrhu a o komunikaci mezi jednotlivými částmi.

V předposlední kapitole 6 je testování aplikace, zhodnocení jeho výstupu a navrhnutí možných budoucích rozšíření.

Poslední kapitole 7 obsahuje závěr.

Kapitola 2

Analýza

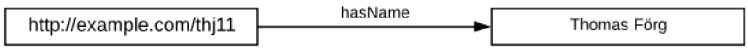
Pro úspěšný návrh a implementaci je třeba ujasnit si, co je vstupem aplikace a jakou funkcionalitu má splňovat výsledný produkt.

V této kapitole je popsáno v jakém formátu jsou vstupní data uložena, jejich struktura, způsob k jejich přistupování a nástroj, kterým jsou generována. Dále tato kapitola obsahuje analýzu požadavků na výslednou aplikaci a určení cílové skupiny uživatelů.

2.1 Resource Description Framework (RDF)

Veškerá vstupní data jsou popsána pomocí frameworku RDF [19] sloužícímu pro popis, výměnu a znovupoužití zdrojů. Zdroje se popisují pomocí tvrzení, která jsou ve tvaru *zdroj (subjekt) - vlastnost (predikát) - hodnota vlastnosti (objekt)*, známé jako trojice. Každý ze zdrojů je definován unikátní adresou URI [1] a může být jak subjekt, tak i objekt. Pro popis významu zdrojů a jejich relací nám slouží ontologický slovník [26], do které se odkazuje pomocí URI zdrojů.

Tento jednoduchý model nám umožňuje vytvořit složité datové struktury, tvořící graf s orientovanými hranami, v nichž hrany znázorňují jednotlivé vlastnosti.



Obrázek 2.1: Příklad RDF trojice.

2.2 TimelineAnalyzer

TimelineAnalyzer je nástroj, který se stará o stahování obsahu z profilů na sociálních sítí přes jejich aplikačního rozhraní. Mimo jiné dokáže ukládat informace o prohlížení obsahu na sociálních sítí, které jsou dostupné v historii v lokálních souborech webového prohlížeče uživatele.

Uložená data budou dále nazývána *záznamy* (příspěvky, historie prohlížení), které jsou sdružovány do *skupin záznamů* (profily uživatelů na sociálních sítích, historie uživatele prohlížeče).

[1] Uniform Resource Identifier – jednotný identifikátor zdroje

Při zpracování nových záznamů TimelineAnalyzer [2] zjišťuje a ukládá, zda se mezi daty nevyskytuje více záznamů, které obsahují stejnou informaci. Příkladem takové informace může být odkaz na stejný obrázek ve dvou záznamech.

Všechny uložené záznamy jsou reprezentovány s využitím datového modelu RDF.

Více informací o fungování a implementaci nástroje TimelineAnalyzer lze najít v [11].

2.3 Popis vstupních dat

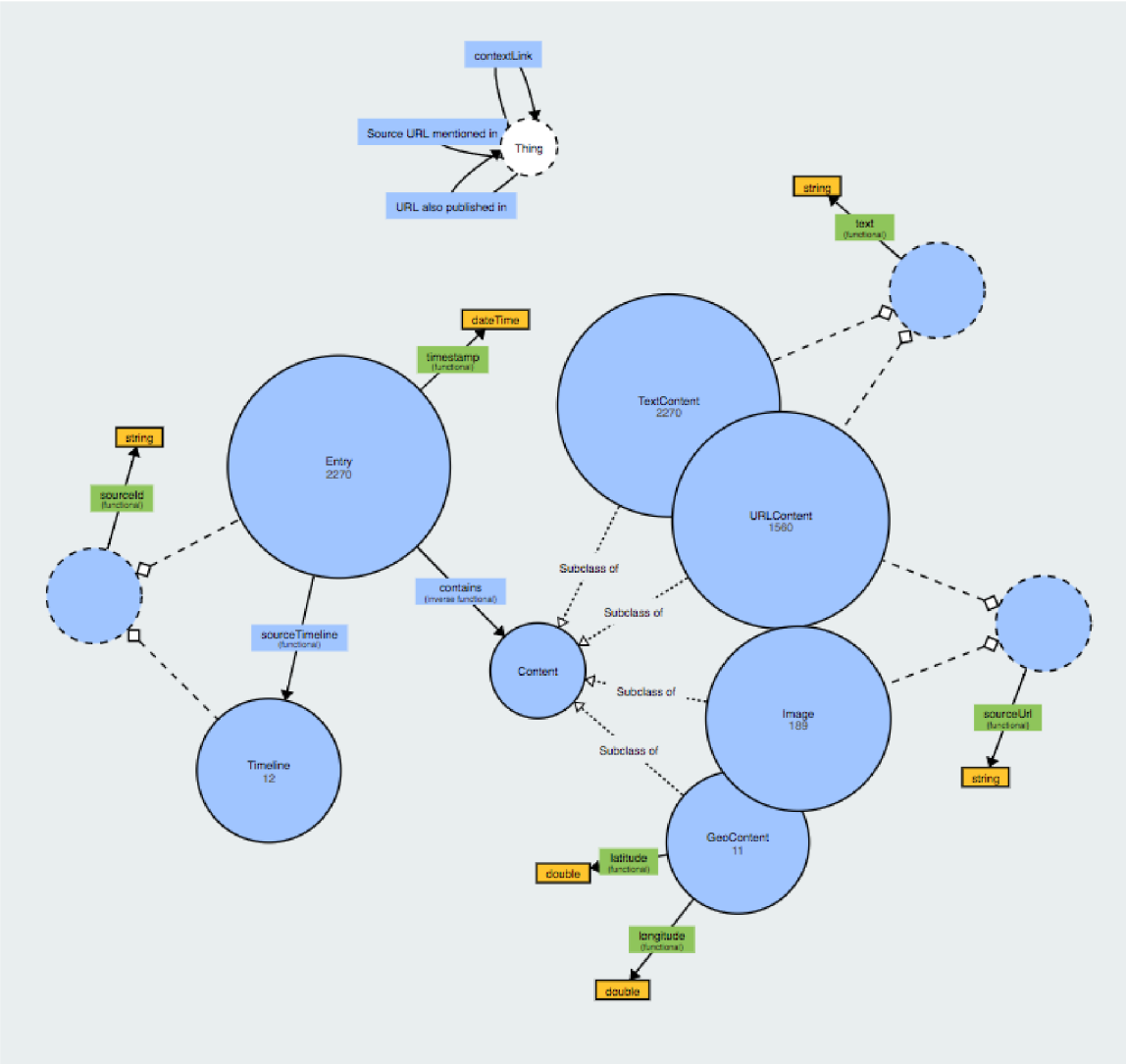
Na obrázku 2.2 je grafické zobrazení struktury vstupních dat.

- **Entry** - Záznam obsahující informace o času zveřejnění a jejich identifikátoru.
- **Timeline** - Obsahuje identifikátor skupiny záznamů. Skupina záznamů znázorňuje uživatelské profily a skupiny na sociálních sítích nebo profil uživatele z historie prohlížení sociálních sítí v prohlížeči.
- **Content** - Obsah záznamu, který se dále dělí na *TextContent*, *URLContent*, *Image* nebo *GeoContent*.
- **TextContent** - Obsahuje text.
- **URLContent** - Obsahuje odkaz.
- **Image** - Obsahuje odkaz na obrázek.
- **GeoContent** - Obsahuje GPS [3] souřadnice.

Podle výpisu typů obsahu a grafického znázornění má každý záznam několik možných typů dat: *TextContent*, *URLContent*, *Image* nebo *GeoContent*. Tyto záznamy se sdružují do skupin, které znázorňují příspěvky na jednom profilu/skupině sociální sítě nebo historii prohlížení obsahu sociálních sítí pro jednoho uživatele.

V datech jsou dále obsaženy vazby na záznamy, které obsahují stejnou informaci jako záznam jiný. V případě poskytnutých vstupních dat jsou brány jako stejné informace URL[4] odkazy.

[2] TimelineAnalyzer - https://github.com/nesfit/timeline-analyzer
[3] Global Positioning System - globální polohový systém
[4] Uniform Resource Locator - jednotná adresa zdroje



Obrázek 2.2: Grafická reprezentace RDF dat. [18]

2.4 Přístup k datům

Data, se kterýma má výsledná aplikace pracovat jsou přístupná na externím serveru, který musí být při spouštění poskytnut. Předpoklad serveru je, že používá databázi RDF4J [5] pro analýzu a dotazování nad RDF daty. Tato databáze umožňuje dotazování na data přes REST 3.1.1 rozhraní. Dotazy se pro přístup k RDF datům provádějí v dotazovacím jazyku SPARQL 3.3.

2.5 Analýza požadavků

Cílem projektu je vytvořit interaktivní webovou aplikaci, která se dokáže dotazovat na data dostupná na externím serveru a poté je přívětivě vizualizovat. Slouží jako most mezi uživatelem a daty poskytnutými nástrojem TimeAnalyzer.

Celá aplikace je značně specifická a je proto určena pro skupinu uživatelů, kteří jsou seznámeni se strukturou dat vytvořených nástrojem TimelineAnalyzer.

Klíčové požadavky na aplikaci byly definovány po konzultaci s vedoucím této práce a obsahují:

- Efektivní zobrazení velkého množství dat.
- Zobrazení pouze vybraných skupin a typů záznamů.
- Zobrazení detailů záznamu a jeho související záznamy se stejnou informací.
- Synchronizace záznamů podle času a současné procházení všech skupin záznamů.

Prvním z požadavků je možnost efektivního zobrazení velkého množství dat. Proto aby toho bylo možné dosáhnout je třeba navrhnout aplikaci tak, aby nebyly záznamy stahovány najednou, ale postupně na základě akcí uživatele.

Dalším požadavkem je možnost vybrání skupin záznamů, které si chce uživatel zobrazit. Z těchto záznamů je třeba pro lepší uživatelský zážitek také umožnit jejich filtrování podle času zveřejnění a typu obsahu. Jelikož není vhodné, aby si uživatel vybíral zobrazené záznamy znovu při každém přístupu do webové aplikace, je nutné autentizovat uživatele a ukládat jeho aktuální konfiguraci zobrazených záznamů. Tento krok nám taktéž aplikaci zabezpečí proti uživatelům, kteří by do aplikace neměli mít přístup. Pro realizaci je nutné navrhnout vlastní serverovou část.

Záznamy mají různé typy dat a je třeba všechen jejich obsah po kliknutí na ně zobrazit a to včetně všech souvisejících záznamů, které obsahují stejnou informaci.

Posledním požadavkem je synchronizace záznamů podle času. V datech máme několik skupin záznamů, napříč kterými je požadována jejich časová synchronizace a současné procházení jejich záznamů.

Kapitola 3

Související technologie

V této kapitole jsou obsaženy dnešní trendy a technologie při vývoji moderních webových aplikací. Vývoj moderních webových aplikací je komplexní záležitostí a skládá se z mnoha spolupracujících technologií, které jsou zde z části popsány a některé z nich i porovnány s alternativami.

Zmíněné technologie budou dále použity v návrhu a implementaci aplikace.

3.1 Serverová část (backend)

Tato sekce pojednává o technologiích serverové části aplikace (back-end), která je často využívána při vývoji moderních webových aplikací jako zdroj dat pro klientskou část (front-end).

3.1.1 REST rozhraní

REST neboli *Representational state transfer* je architektura rozhraní pro komunikaci přes HTTP [1] protokol. Používá se pro komunikaci mezi dvěma nezávislými stanicemi, kdy jedna z nich (klient) potřebuje získat přístup ke zdrojům cílové stanice (server). Server slouží jako jednotný zdroj dat a může z něj čerpat více zařízení najednou. Data se typicky přenášejí pomocí serializačních formátů jako jsou JSON [2] , XML [3] nebo pokročilejší Protocol Buffers[4].

REST je nestavový a při každém požadavku na server je třeba specifikovat všechny údaje pro získání potřebných dat. Skládá se ze čtyř CRUD operací: Create *(POST HTTP request)*, Read - *(GET HTTP request)*, Update - *(PUT/PATCH HTTP request)*, Delete - *(DELETE HTTP request)*.

Jelikož běží nad protokolem HTTP, zdroje jsou identifikovány pomocí URL adres.

3.1.2 Django REST framework

Django REST [1] je open-source framework zaměřující se na jednoduchou tvorbu REST aplikačních rozhraní ve skriptovacím jazyce Python. Je pouze nadstavbou nad Django frameworkem, který rozšiřuje tak, aby bylo dosáhnuto jednoduššího vývoje REST rozhraní.

[1] Hypertext Transfer Protocol - internetový protokol pro přenos multimediálních dokumentů
[2] JavaScript Object Notation - JavaScriptový objektový zápis
[3] Extensible Markup Language - značkovací jazyk
[4] Protocol Buffers - mechanismus pro serializaci strukturovaných dat do binární podoby

Umožňuje nám popsat databázové tabulky relační databáze pomocí Python modelů s vlastním ORM [5]. ORM je programovací technika pro převod dat mezi relační databází a objektově orientovaným programovacím jazykem. Díky tomu je možné s daty z databáze pracovat jako s obyčejným Python objektem. Výhodou Django REST je, že na základě popsaných modelů databázových tabulek dokáže automaticky vygenerovat REST zdroje (endpointy) se všemi potřebnými CRUD operacemi.

Mezi další užitečné funkce a vlastnosti frameworku patří vygenerování webového uživatelského rozhraní pro procházení a zobrazení dat dostupných zdrojů (endpointů), sada nástrojů pro autentizaci a autorizaci uživatelů a omezení jejich přístupu k určitým zdrojům, možnost detailní konfigurace, rozsáhlá dokumentace a aktivní komunita.

Flask jako alternativa

Flask je microframework napsaný taktéž ve skriptovacím jazyce Python. Dá se využít pro tvorbu REST rozhraní, ale oproti Django REST se nesnaží sdružovat mnoho funkcionalit do jednotného balíčku. V základu tedy nepodporuje například ORM a generování dokumentace zdrojů. Na jednu stranu se dá říci, že je Flask limitován, ale na stranu druhou je díky tomu jednoduché vytvořit první aplikaci za pár minut. I když je Flask daleko menší než Django REST, má rozsáhlou komunitu, která se stará o přidávání rozšíření doplňujících chybějící funkcionalitu.

Ostatní alternativy

Pro psaní REST rozhraní je možné využít mnoho dalších frameworků z jiných jazyků. Za zmínku stojí:

- Lavarel - Framework napsaný v PHP [6], který poskytuje podobně rozsáhlou funcionalitu jako Django REST.
- Express - Minimalistický framework postavený na Node.js[7]. Obsahuje pouze základní funkcionalitu jako Flask microframework.

3.1.3 Autentizace pomocí JSON Web Tokenu (JWT)

Serverová část aplikace je často veřejně dostupná na internetu, kde k ní má každý přístup. Pokud nechceme aby k datům mohl přistupovat kdokoliv a aby se k nim mohli dostat pouze autorizovaní uživatelé, je třeba zvolit správná opatření. Jedním z nich je využití JWT [3].

Žadatel o data musí u každého požadavku na server přiložit JWT token, podle kterého si server ověří zda mu může povolit přístup. Token se získává zasláním validních přihlašovacích údajů na vyhrazený REST zdroj serveru a skládá se z údajů o uživateli, datumu vytvoření tokenu, expirace tokenu a dalších informací, které si můžeme při implementaci specifikovat.

Je zakódovaný Base64 kódováním a podepsán tajným klíčem, který je známý pouze na straně serveru. Server si poté dokáže u každého dotazu ověřit tajným klíčem podpis příjmutého JWT tokenu a není třeba ukládat rozeslané tokeny do perzistentního uložiště. Aby byl přenos bezpečný, je třeba využívat šifrovanou komunikaci pomocí HTTPS[8], jelikož JWT Web Token data nešifruje, pouze zaručuje správnou autentizaci uživatele.

[5] Obect relation mapper
[6] PHP: Hypertext Preprocessor - skriptovací programovací jazyk
[7] Node.js - interpret JavaScriptu pro spuštění kódu na straně serveru
[8] HTTP Secure - rozšíření HTTP pro šifrování přenosu

```
{
 typ: "JWT",
 alg: "HS256"
}.
{
 user_id: 2, % id uzivatele
 username: "admin",
 exp: 1522535558, % expirace tokenu
 email: "admin@admin.com",
 orig_iat: 1522531958 % vytvoreni tokenu
}.
```

Výpis 3.1: Struktura dekódovaného JWT tokenu.

3.1.4 Perzistentní data s PostgreSQL

PostgreSQL je open-source objektově-relační databázový systém. Slouží k bezpečnému ukládání perzistentních dat na databázovém serveru.

Komunikace s databází a manipulace s daty je realizována pomocí dotazovacího jazyka SQL[9].

3.2 Klientská část (frontend)

V této sekci jsou rozebrány některé z technologií pro tvorbu dynamických webových uživatelských rozhraní.

3.2.1 HyperText Markup Language

HyperText Markup Language (HTML) [2] je značkovací jazyk a je základním prvkem pro tvorbu webových aplikací. Při přístupu na webovou stránku je HTML kód načten a zpracován prohlížečem a podle jeho struktury jsou zobrazena data. Skládá se ze zanořených elementů známých jako *tagy*.

```
<html>
    <body>
        <h1>Nadpis</h1>
        <p>Text...</p>
    </body>
</html>
```

Výpis 3.2: Příklad HTML kódu.

3.2.2 Cascading Style Sheets

Cascading Style Sheets (CSS) je jazyk pro popsání způsobu zobrazení HTML elementů. Hlavní myšlenka CSS je oddělit obsah HTML dokumentu od jeho struktury zobrazení.

[9] Structured Query Language

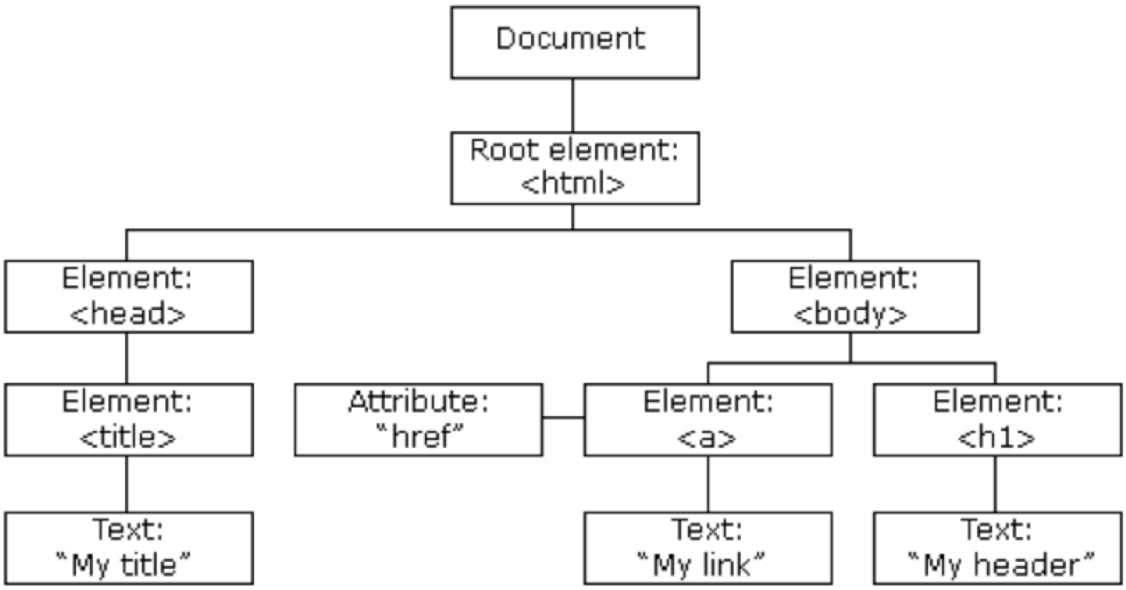
```
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}
```

Výpis 3.3: Příklad CSS stylů.

Document Object Model

Document Object Model (DOM) je výsledkem zpracování HTML dokumentu a CSS stylů prohlížečem. Je to rozložení HTML elementů s přidělenými CSS styly do stromové struktury, ke které lze přistupovat aplikačním rozhraním (API) pomocí JavaScriptu a dynamicky měnit/přidávat/mazat elementy dokumentu, jejich atributy nebo styly.



Obrázek 3.1: Stromová struktura elementů. [4]

ECMAScript

ECMAScript (ES) [10] je standardizace skriptovacího jazyka nejčastěji využívaného pro psaní webových aplikací na klientské straně (frontend), ale v dnešní době je rozšířen i na straně serveru. Je nejčastěji implementován v jazyce JavaScript. ECMAScript se stále vyvíjí a vycházejí jeho novější verze, které přidávají novou funkcionalitu. V případě nových verzí, které nejsou podporovány prohlížečem je umožněno kód zpětně přeložit na starší verzi, což se nazývá transpilace. Aktuálně nejnovější verzí je ECMAScript 2017.

3.2.3 Javascript

JavaScript (JS) je objektově orientovaný a multiplatformní skriptovací jazyk. Využívá se nejčastěji pro tvorbu interaktivních webových aplikací a vkládá se přímo do HTML kódu stránky. Obvyklé spuštění JavaScript kódu probíhá až na straně klienta po stáhnutí stránky z internetu.

Je jej možné využít i na straně serveru pomocí Node.js, což je interpret obsahující knihovny, díky kterým je možné JavaScript využít jako všeobecný programovací jazyk.

3.2.4 Vue.js

VueJS je open-source JavaSriptový framework [23] pro vytváření vysoce interaktivních uživatelských rozhraní. Zaměřuje se na vytváření tzv. SPA (Single Page Application) a tvoří pouze *View* vrstvu v kontextu softwarové architektury *Model-View-Controller*[10]. SPA je webovou aplikací, u které se při návštěvě stránky stáhne všechen HTML, CSS a JavaScript kód bez obsahu, který chceme zobrazit. Stáhne se tedy pouze základní kostra stránky, do které se na základě interakce s uživatelem dynamicky načte zobrazitelný obsah. Pro dynamické načtení se využívá AJAX[11] volání vůči serveru.

Hlavní myšlenkou Vue.js je dekompozice aplikace do malých a znovupoužitelných komponent a pracování s virtuálním DOMem 3.2.4 pro zvýšení rychlosti aplikace při práci s dynamickým obsahem.

Komponenta

Komponenta je část aplikace, která v sobě sdružuje HTML, CSS a JS kód. Ve výsledné aplikaci tvoří komponenty hierarchickou strukturu v níž pro komunikaci mezi nimi slouží předávání proměnných (props) a zasílání událostí (events).

Proměnné přijímá komponenta od své rodičovské (nadřazené) komponenty.

Proto aby mohla podřazená komponenta komunikovat s rodičovskou se využívají události. Při vytvoření události je k ní možné připojit data a rodičovská komponenta si danou událost odchytí a na jejím základě provede požadované akce.

Virtuální DOM

Aby byl zajištěn vysoký výkon aplikace, využívá se v Vue.js techniky tzv. virtuálního DOMu. Virtuální DOM je JavaScriptový objekt, který reprezentuje reálný DOM. Jelikož jsou operace nad reálným DOMem značně pomalé, je virtuální DOM využíván pro prvotní realizaci operací a jakmile jsou hotovy, jsou tyto změny pomítnuty do DOMu reálného. Tato technika dokáže eliminovat některé operace a výpočty, protože může měnit reálný DOM po větších částech, které promítne do reálného DOMu najednou a ne po jednotlivých krocích.

React jako alternativa k Vue.js

React [14] je knihovna využívající stejné myšlenky jako Vue.js a to rozložení stránky do komponent. Stejně jako Vue.js je používán pro tvorbu SPA.

Jednou ze speciálních věcí ve světě Reactu je JSX. JSX je speciální syntaxe, která do sebe sdružuje JS, HTML a CSS kód.

[10] MVC - architektura rozdělující datový model aplikace, uživatelské rozhraní a její řídicí logiku.
[11] Asynchronous JavaScript And XML - asynchronní dotazování serveru

Výhody:

- Existuje delší dobu a tím pádem má rozvinutější ekosystém.

Nevýhody:

- Učící křivka je oproti Vue.js velmi vysoká.
- JSX není přívětivý pro nezkušeného uživatele.

Jednou z dalších alternativ je Angular [15]. Na rozdíl od knihoven Vue.js a React je Angular robustní framework. Místo psaní kódu v JavaScriptu se využívá jeho nadstavba TypeScript.

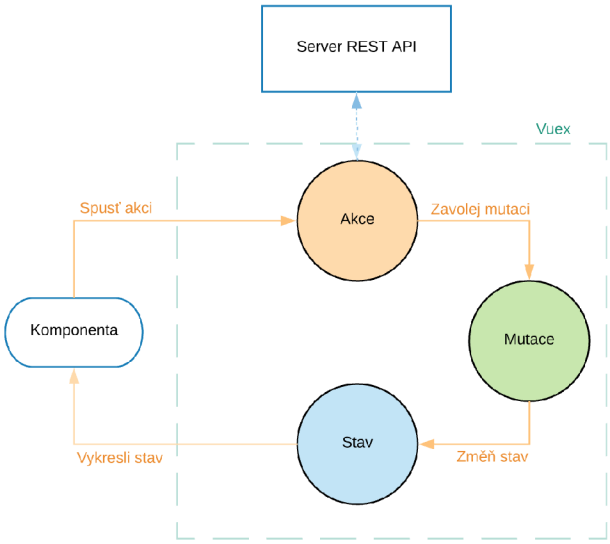
Mezi výhody Angularu může patřit již zmíněný TypeScript, jelikož pomáhá předcházet některým chybám díky statickému typování proměnných. Za další výhodu se dá považovat ověřenost frameworku časem. Na druhou stranu je jedna z jeho nevýhodou robustnost, která nutí vývojáře dodržovat určitá pravidla a tím je omezovat.

Vuex

Jeden z problémů při psaní aplikace ve Vue.js je udržování konzistentního stavu aplikace napříč komponentami. Je totiž nutné si mezi nimi stav postupně propagovat pomocí *props* a *events*. Tenhle přístup je ve větších aplikacích s mnoha komponentami nepřehledný a zdlouhavý.

Vuex [24] je knihovna pro centralizovanou správu stavu aplikace, ke kterému je možné přistupovat z každé komponenty. Skládá se ze tří základních částí *Akce, Mutace a Stav.* 3.2

- *Stav* slouží jako kontejner, kde se uchovávají všechna data.
- *Mutace* jsou funkce, které mají přístup ke stavu a využívají se pro jeho změnu.
- *Akce* jsou funkce, které se využívají k volání mutací. Například stáhnutí nového obsahu a jeho uložení do celkového stavu pomocí mutace.



Obrázek 3.2: Schéma Vuex logiky.

Alternativa Vuex - Redux

Redux [9] je knihovna, která se dá využít místo Vuex. Má stejné myšlenky a využívá se většinou v kombinaci s Reactem.

Skládá se ze stavu, akcí a reducerů. Stav a akce fungují stejně jako v knihovně Vuex. Zbývající reducer je podobný mutaci ve Vuex, ale na rozdíl od ní stav nemění, ale nahrazuje jej stavem novým.

Axios

HTTP klient pro přístup ke zdrojům na vzdáleném serveru. [25]

Vue-Router

Slouží k URL adresaci dynamického obsahu na straně klienta v případě SPA aplikací. Podle toho jakou adresu uživatel zadal do webového prohlížeče se načte a zobrazí jen určitý pohled aplikace.

Vue Virtual Scroll List

Externí komponenta, která se stará o vykreslování velkého množství prvků ve skrolovacím seznamu bez velké ztráty výkonu aplikace. Data seznamu jsou uložena v paměti a komponenta vykresluje pouze určitou množinu prvků najednou. [8] Při skrolování vykreslené prvky mění podle toho, jaká část seznamu je uživateli zobrazena.

VueDraggable

VueDraggable je komponenta umožňující přesouvání elementů, pomocí kliknutí a táhnutí elementu, na polohu jiného elementu.

Vuetify

Vuetify [17] je framework obsahující sadu komponent pro tvorbu uživatelských rozhraní ve stylu Material Design [12].

3.2.5 NuxtJs

Nuxt.js [22] je framework pro povolení *server-side renderingu* pro Vue.js aplikace. Narozdíl od SPA a obyčejného využití Vue.js, kde se vše vykresluje na straně klienta, umožňuje Nuxt.js renderování (vykreslování) přenést na stranu serveru. Výhodou tohoto přístupu je rychlejší prvotní načtení a interakce s webovou stránkou, možnost inicializace Vuex uložiště na straně serveru a lepší SEO [13] podpory.

Next.js jako alternativa

Next.js [16] funguje na stejném principu jako Nuxt.js, ale je vytvořen pro poskytnutí server-side renderingu k React aplikacím.

[12] Material Design - pravidla pro design aplikace vytvořena Googlem
[13] Search Engine Optimization - optimalizace pro vyhledávače

3.3 SPARQL

SPARQL [21] je dotazovací jazyk nad daty popsanými datovým modelem RDF. Syntaxe je podobná jazyku SQL a obsahuje stejně jako SQL klíčová slova *SELECT, WHERE a FROM*. Rozdíl je v tom, že data která jsou uložena v RDF datasetu (kolekce RDF) jako trojice *subject-predikát-objekt*, tak i v SPARQL se dotaz skládá z množiny trojic. Každá z hodnot ve trojici může být proměnná a výsledek je vybírán jako shodné trojice vzhledem k proměnným.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?person foaf:name ?name }
```

Výpis 3.4: Příklad SPARQL dotazu.

3.4 Docker

V dnešní době se stávají aplikace čím dál komplexnější a obsahují mnoho závislostí na externí knihovny, programy a další. Tohle se stává problémem pokud chceme aplikaci spustit na různých strojích, protože musíme nejdříve nainstalovat všechny potřebné závislosti.

Pro vyřešení tohoto problému lze použít Docker [13], který kód aplikace se všemi závislostmi zabalí a spustí v tzv. kontejneru.

Kontejner

Kontejner je balíček, který obsahuje veškeré knihovny, konfigurační soubory a závislosti aplikace. Podobá se plné virtualizaci systému, jako je například VMware [14], ale nezatěžuje tolik zdroje. Každý kontejner sdílí prostředky operačního systému (Linux kernel) se všemi ostatními kontejnery na stejném stroji.

Pro specifikaci závislostí, které má kontejner se využívá soubor Dockerfile.

Docker Compose

V rámci komplexních aplikací, ve kterých je třeba spustit více různých a navzájem komunikujících kontejnerů je možné využít rozšíření Dockeru zvané Docker Compose. Tohle rozšíření dokáže zautomatizovat spuštění více kontejnerů s použitím jediného příkazu.

[14] VMware - slouží k virtualizaci operačních systémů

Kapitola 4

Návrh aplikace

S ohledem na požadavky pro výslednou aplikaci je v této kapitole vytvořen její návrh.

V první sekci je rozebrána celková architektura 4.1 aplikace včetně vybraných technologií. Za ní následuje sekce zabývající se návrhem grafického rozhraní 4.2 a v poslední sekci je obsažen návrh schématu relační databáze 4.3 pro ukládání perzistentních dat, které jsou potřebné pro splnění požadované funkcionality.

4.1 Architektura systému

Schéma architektury komunikujících částí aplikace lze najít na obrázku 4.1.

Hlavním cílem této práce je vytvoření dynamické a interaktivní webové aplikace zobrazující data agregovaná nástrojem TimelineAnalyzer a popsaná datovým modelem RDF. Data jsou přístupná na externím RDF4J databázovém serveru, který bude přístupný při spuštění aplikace. Dotazování na něj probíhá přes REST rozhraní s pomocí SPARQL dotazů.

V kapitole 2.5 je zmíněna potřeba rozlišení uživatelů a ukládání jejich konfigurace. Aby toho bylo možné dosáhnout, rozhodl jsem se rozdělit aplikaci na 2 samostatné části. První z nich je klientská část (front-end), která bude zobrazovat data a druhá část je serverová (backend), která bude zprostředkovávat data klientovi přes REST rozhraní.

Pro vytvoření serverové části využijeme Django REST framework umožňující tvorbu REST rozhraní, jenž bude sloužit jako jediný zdroj dat pro klienta.

Abychom mohli autentizovat a povolit přístup ke zdrojům serveru pouze pro povolené uživatele využijeme ověřování uživatelů pomocí JWT Web Tokenu, jehož implementace je dostupná v rozšíření Django REST frameworku [20]. Pro autentizaci uživatele bude sloužit klasická kombinace uživatelského jména a hesla, k jejíž ukládání bude sloužit databáze.

Pro ukládání uživatelů a jejich konfigurace použijeme databázi PostgreSQL, u které budeme očekávat, že je poskytnuta při spuštění aplikace na produkci. Pro usnadnění vývoje na lokálním stroji implementujeme navíc SQLite databázi [1], do které se budou ukládat data pokud není PostgreSQL databáze k dispozici. Všechny operace s databází na straně serveru v Django REST frameworku budou implementovány pomocí zabudovaného ORM.

Důležité bezpečnostní opatření, které by měl server splňovat je chráněné zpřístupnění dat z externího RDF4J serveru ke klientovi. K tomuto účelu implementujeme na serveru

[1] SQLite - databázový systém ve formě jednoduchého programu

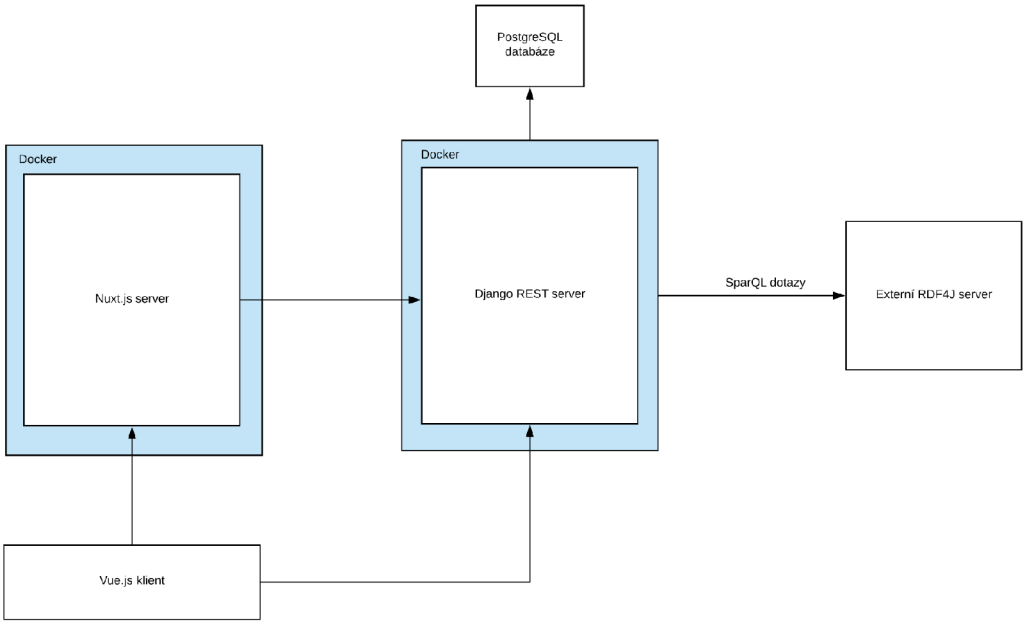
jednoduchou proxy [2] zabezpečenou JSON Web Tokenem, která bude přeposílat SPARQL dotazy na RDF4J server a poté vrátí výsledek dotazu zpět klientovi. Tohle nám umožní bezpečně posílat SPARQL dotazy přímo z klienta.

Vytvoření klientské části implementujeme s Vue.js frameworkem a pro ulehčení práce s udržováním konzistentního stavu napříč klientskou částí využijeme knihovnu pro správu centrálního stavu Vuex. Klientská část bude naplňovat centrální stav aplikace daty dostupnými přes REST rozhraní navrhnuté serverové části s knihovnou Axios3.2.4. V případě dotazování na RDF data se budou posílat SPARQL dotazy.

Poslední důležitá část chybějící v návrhu klientské části je způsob, jakým si ji může uživatel načíst. Jedním z nejjednodušších řešení by bylo zveřejnění HTML, CSS a JavaScript přes navržený server v Django REST frameworku nebo napsání jiného serveru pouze pro tento účel.

V našem případě jsem zvolil jít jinou cestou a to s použitím Nuxt.js frameworku. Nuxt.js umožňuje spustit vlastní server s podporou server-side renderingu. Znamená to, že když uživatel přistoupí na stránku, můžeme některý obsah načíst již na straně serveru, vykreslit jej a uživateli poslat HTML, CSS a JS soubory s vykresleným obsahem. Pokud bychom použili pouze Vue.js, dostal by uživatel jen základní kostru kódu, které by stahovala a vykreslovala všechen obsah až v prohlížeči. Stahování obsahu a vykreslování na straně klienta je ovšem obecně pomalejší.

Poslední část, která chybí navrhnout je způsob spouštění aplikace. Aby bylo spouštění a s ním svázaný vývoj jednoduchý, využijeme možnosti Dockeru. Klientskou a zvlášť i serverovou část tedy zabalíme do Docker kontejnerů a jejich současné spuštění implementujeme s pomocí Docker Compose.



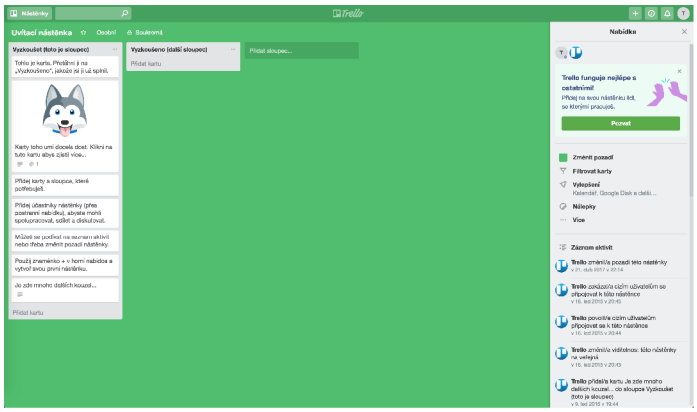
Obrázek 4.1: Schéma achitektury celé aplikace.

[2] Proxy - prostředník mezi dvěma stanicemi

4.2 Grafické rozhraní

Při návrhu grafického rozhraní aplikace jsem vycházel z požadavků a připomínek vedoucího na výslednou funkcionalitu. Celé grafické rozhraní má sloužit jako prostředník mezi uživatelem a zdrojem dat. Jedním z požadavků, na který jsem bral obzvláště ohled byla možnost zobrazení velkého množství záznamů ze sociálních sítích.

Jako inspirace mi při návrhu vlastního designu posloužil nástroj pro správu projektů Trello 4.2. Každý projekt zde má nástěnku, na které jsou horizontálně zobrazené sloupce znázorňující typy úkolů a jejich seznam.



Obrázek 4.2: Uživatelské rozhraní Trella. [12]

Prvky designu Trella jsou vhodné i pro tuto aplikaci, kde může být jedna skupina záznamů v datech zobrazena jako sloupec obsahující všechny její záznamy.

Grafické rozhraní aplikace je rozděleno na 4 hlavní části:

- **Přihlašovací část**

  Obsahuje formulář k přihlášení uživatele zadáním jména a hesla.

- **Hlavní část („Dashboards")**

  Nejdůležitější část, která je rozložena do 4 sekcí. Hrubý náhled rozložení hlavní stránky lze vidět na obrázku 4.3.

  První a hlavní sekce je hned ve středu plochy a zobrazuje horizontálně zarovnané sloupce, které znázorňují jednotlivé skupiny záznamů. Každá skupina záznamů se dále skládá z několika náhledů záznamů. Aby bylo jasné, jaký obsah je v záznamu dostupný je v jeho náhledu vzorek textu záznamu, datum zveřejnění a pomocí ikon vyznačené typy dat, které obsahuje. Navíc je v náhledu zobrazená také ikona, která je přítomna pouze pokud má záznam související záznamy se stejnou informací a po kliknutí na ni se všechny související náhledy záznamů zobrazené na hlavní straně označí tmavou barvou.

  Další dvě sekce jsou výsuvné panely, kde je jeden na levé straně a obsahuje názvy všech zobrazených skupin záznamů v hlavní části. Obsahuje rychlé vyhledávání a tlačítko u každé skupiny na její odstranění z hlavní sekce. Taktéž je v něm tlačítko na otevření administrace zobrazených skupin záznamů.

  Druhý výsuvný panel obsahuje filtry na zobrazení záznamů zveřejněných v určitém čase, filtrování záznamů podle hledaného textu a filtry pro zobrazení záznamů, které

mají potřebný typ obsahu. Kromě filtrů obsahuje přepínače pro zapnutí synchronizace záznamů podle času napříč skupinami a zapnutí synchronizovaného procházení všech skupin najednou.

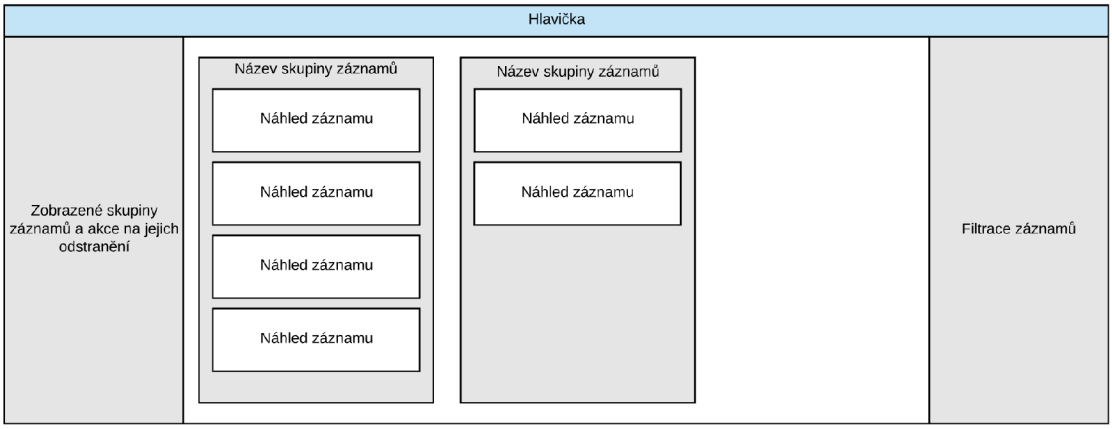
Poslední část je hlavička stránky, která obsahuje tlačítka pro zobrazení a schování výsuvných panelů, tlačítko na odhlášení uživatele a jaké filtry jsou aktivní.

- **Detaily záznamu**

  Obsahuje veškeré informace o daném záznamu a je otevřena po kliknutí na náhled záznamu na hlavní stránce. Pokud v něm jsou odkazy na obrázky, zobrazí je v galerii. Podobně pokud jsou v záznamu zeměpisné souřadnice místa, zobrazí náhled mapy, kde se místo nachází. Ostatní typy dat zobrazuje tak jak jsou uloženy. Všechna data, která záznam obsahuje jsou v levé straně stránky a na straně pravé jsou náhledy souvisejících záznamů obsahující stejnou informaci.

- **Administrace pro správu zobrazených skupin záznamů**

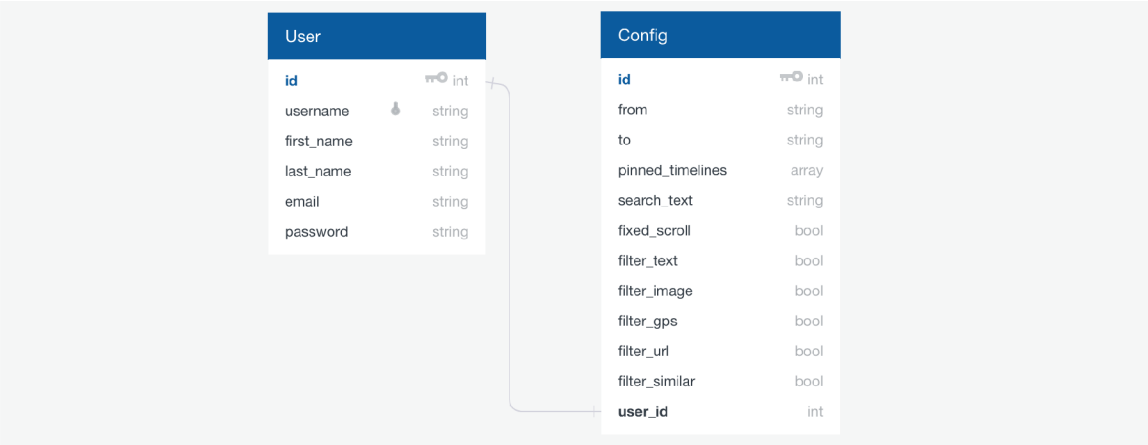
  Tato část slouží k přidávání/odebrání nových skupin záznamů na hlavní stranu a změnu jejich pořadí. Je složený ze dvou seznamů, z nichž jeden zobrazuje skupiny záznamů, které nejsou zobrazeny na hlavní straně a umožňuje jejich přidání. Druhý musí zobrazovat skupiny záznamů obsažené na hlavní stránce a umožnit jejich odebrání nebo změnu pořadí.



Obrázek 4.3: Model grafického rozhraní.

4.3 Databáze

V této aplikaci slouží databáze pouze k uložení informací o uživateli a informace o aktuální konfiguraci aplikace. Jednou z funkcí Django REST frameworku je možnost využití předem definovaného modelu pro uživatele, ze kterého si automaticky vytváří tabulku v databázi. Tohohle jsem se rozhodl využít i v této aplikaci a je proto nutné vytvořit další tabulku, která obsahuje informace o aktuální konfiguraci aplikace pro daného uživatele. Tuto tabulku je třeba spojit s uživatelem pomocí vztahu 1:1, jejichž schéma lze vidět na obrázku 4.4



Obrázek 4.4: Schéma databázových tabulek.

Kapitola 5

Implementace

V předchozí kapitole 4 byla navrhnuta celková architektura výsledné aplikace včetně grafického rozhraní a důležitých technologií potřebných k její implementaci. V této kapitole se zabýváme implementací tohoto návrhu. Implementace probíhala iterativně a v průběhu se značně lišila, jak použitými technologiemi, tak i celkovou strukturou aplikace. Bude zde tedy popsána pouze finální implementace patřičných částí. Jako první je serverová část aplikace 5.1, za kterou následuje popis implementace části klientské 5.2.

5.1 Server

V této sekci se zabýváme tvorbou serverové části s REST rozhraním pomocí Django REST frameworku. Jako první si popíšeme strukturu aplikace, dále tvorbu databázových modelů s nimiž se bude pracovat při ukládání uživatelských dat, implementací autentizace uživatele a SPARQL proxy pro přístup k RDF datům na externím serveru.

5.1.1 Adresářová struktura

Hlavní části adresářové struktury:

- *./requirements.txt*

  Soubor obsahuje všechny Python knihovny potřebné pro spuštění serveru.

- *./app/settings.py*

  Konfigurace Django REST aplikace, která například obsahuje nastavení databáze nebo metodu autentizace uživatelů.

- *./api/models.py*

  Popis ORM modelu uživatelské konfigurace pro relační databáze.

- *./api/serializers.py*

  Serializéry pro převod dat z databáze do formátu JSON.

- *./api/views.py*

  Pohledy REST rozhraní popisující akce, které se mají vykonat při volání různých HTTP metod.

- *./api/urls.py*

  Seznam adres REST zdrojů a k nim přiřazené pohledy.

- *./db.sqlite3*

  Soubor s databází uživatelů pro lokální vývoj v případě, že není poskytnuta PostgreSQL databáze.

- *./Dockerfile*

  Soubor obsahující popis Docker kontejneru se všemi závislostmi a akcemi pro spuštění serverové části.

5.1.2 Databázové modely

Databázové modely zde slouží pro ukládání uživatelů a aktuálního stavu aplikace. V našem případě se ukládají jaké skupiny záznamů jsou zobrazeny na hlavní straně a konfigurace filtrů obsahu. Databázové modely v Django ORM jsou abstrakcí schématu relační databáze a po jejich definici je dokáže Django vytvořit přímo do naší databáze. Jako jeden ze základních abstraktních modelů, které jsou obsaženy přímo v Djangu je *django.contrib.auth.models.User*. S tímto modelem jsem počítal již v návrhu aplikace a na obrázku 4.3 je ekvivalentem tabulky *User*. Abychom k němu mohli ukládat i data o filtrech musíme na ni napojit novou tabulku *Config* pomocí vztahu 1:1.

```
class Config(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    config_data = models.TextField(default='{}')
    dt_from = models.DateTimeField(null=True, blank=True)
    dt_to = models.DateTimeField(null=True, blank=True)
    pinned_timelines = models.TextField()
    search_text = models.TextField(default='', blank=True)
    fixed_sroll = models.BooleanField(default=False)
    filter_text = models.BooleanField(default=False)
    filter_image = models.BooleanField(default=False)
    filter_gps = models.BooleanField(default=False)
    filter_url = models.BooleanField(default=False)
    filter_similar = models.BooleanField(default=False)
```

Výpis 5.1: Výsledný abstraktní model.

5.1.3 Autentizace a autorizace

Pro autentizaci a autorizaci uživatele je zde podle návrhu 4.1 implementován JWT Web Token přes Django REST rozšíření. Proto aby rozšíření mohlo zjistit, jaký uživatel má přístup lze využít model *django.contrib.auth.models.User*, ze kterého si dokáže automaticky ověřit, zda uživatel existuje.

V Django REST globální konfiguraci je povolen přístup ke všem zdrojům REST rozhraní serveru pouze pro uživatele s platným tokenem.

5.1.4 SPARQL proxy

SPARQL proxy je implementována jako jednoduchý zdroj, který příjmuté tělo dotazů přepošle na RDF4J server a po vrácení výsledku je pošle zpět tazateli. Pro posílání requestů posloužil Python balíček *requests* [7]. Při finální implementaci byla proxy vylepšena přidáním *cache* [1] paměti pro ukládání výsledků z RDF4J serveru na určitou dobu. Pokud v době, kdy jsou výsledky uloženy v cache paměti pošle klient stejný SPARQL dotaz, nebude jej proxy přeposílat dále, ale vrátí data z cache paměti. Tohle umožní zvýšit rychlost načítání dat na klientské části. Jako cache paměť je použita malá *key-value* databáze v paměti zvaná Redis. [6]

5.1.5 Spuštění

Pro jednoduché spuštění aplikace je využito Dockeru. K potřebám této části je napsán Dockerfile obsahující příkazy na nainstalování všech závislostí implementovaného serveru. Tenhle postup značně ulehčuje vývoj a nasazení aplikace.

5.2 Klient

Tato sekce pojednává o implementaci klientské části. Využívají se zde technologie pro vývoj webových aplikací: HTML, CSS a JS. V případě JS byl použit Vue.js framework v kombinaci s Nuxt.js serverem pro renderování obsahu na straně serveru a zpřístupnění statických souborů.

Jsou zde také popsány použité techniky pro správu centrálního stavu, implementace komunikace se serverem a zjednodušené spouštění aplikace pomocí Dockeru.

5.2.1 Adresářová struktura

Pro začátek vývoje projektu bylo nutné využít pomocný program nuxt-cli, který vygeneruje základní kostru celé klientské aplikace, včetně potřebné konfigurace tak aby bylo možné využít Nuxt.js a server-side rendering. V této struktuře jsou rovněž automaticky vygenerovány všechny potřebné části pro vývoj Vue.js aplikace.

Hlavní části adresářové struktury:

- *./package.json*

  Konfigurační soubor, ve kterém jsou obsaženy všechny balíčky a jejich závislosti pro spuštění. Obsahuje také pracovní verzi kódu a příkazy pro spuštění aplikace v lokálním prostředí nebo na produkci.

- *./nuxt.config.js*

  Konfigurace Nuxt.js, která je přímo svázaná i s Vue.js částí. Některé z věcí, které se nastavují je import CSS styl a HTML metadata [2] pro zlepšení SEO.

- *./store/index.js*

  Je v něm veškerá funkcionalita pro centrální správu stavu pomocí knihovny Vuex. Obsahuje akce, mutace a stav aplikace.

[1] Cache - rychlá mezipaměť uchovávají data pro jejich pozdější znovuvyužití
[2] HTML metadata - titulek stránky, favicon

- *./pages/*.vue*

  Obsahuje hlavní komponenty. Každá z hlavním komponent znázorňuje stránku, na kterou uživatel přistoupil.

- *./components/*.vue*

  Adresář se všemi komponentami, které aplikace obsahuje.

- *./Dockerfile*

  Soubor obsahující popis Docker kontejneru se všemi závislostmi pro spuštění klientské části.

5.2.2 Centrální správa stavu

Pro centrální správu aplikace je zde využit Vuex. Je již zabudovaný v automaticky vygenerované kostře aplikace s nuxt-cli a proto je možné rovnou psát vnitřní logiku.

Stav

V této aplikaci se využívá pro centrální ukládání všech skupin záznamů a filtrů obsahu.

```
allTimelines: {},
jwtToken: null,
userConfig: {
    pinnedTimelines: [],
    filters: {
        ...
    }
},
```

Výpis 5.2: Část centrálního stavu aplikace.

Akce

Nejdůležitější částí pro správu stavu jsou akce.

Příklady akcí pro práci s daty na serveru:

- *fetchTimelines*

  SPARQL dotaz na server pro stáhnutí všech dostupných skupin záznamů. Bere pouze metadata jako je název, identifikátor (URI) a celkový počet záznamů ve skupině.

- *fetchTimelineEntries*

  SPARQL dotaz na server pro stáhnutí omezeného počtu záznamů z jedné skupiny. Záznamy obsahují všechna dostupná data.

- *loadMoreTimelineEntries*

  SPARQL dotaz na server pro stáhnutí omezeného počtu záznamů z jedné skupiny. Stahuje pouze ty, které ještě nejsou uloženy.

- *fetchActiveEntry*

  Stahuje pomocí SPARQL dotazu pouze jeden daný záznam a k němu všechny další záznamy se stejnou informací.

- *pinTimeline/unpinTimeline*

  Zobrazí, případně odstraní skupinu záznamů na hlavní stránce ze seznamu zobrazených skupin ve stavu aplikace a následně aktualizuje zobrazené skupiny na straně serveru.

- *getCurrentUserConfig*

  Stáhne ze serveru aktuální konfiguraci aplikace daného uživatele.

- *obtainToken*

  Používá se při přihlášení uživatele pro poslání dotazu na server o poskytnutí JWT tokenu. Token poté uloží do stavu aplikace a každá další akce, které volá vzdálený server tento token přidá do hlavičky dotazu aby mohl být ověřen přístup. Volá se pouze na straně klienta.

5.2.3 Implementace důležitých částí

V této části si popíšeme jak probíhala implementace nejpodstatnějších částí klientské strany aplikace.

Aby byla při implementaci ulehčena práce s psaním CSS stylů a implementováním výsledného rozložení rozhraní, tak je podle návrhu použit framework Vuetify 3.2.4.

Autentizace

Implementovaná serverová část má zabezpečený přístup k REST zdrojům pomocí JSON Web Tokenu. Abychom mohli server z klientské části aplikace používat, je třeba ke zdrojům jeho REST rozhraní nejdříve získat přístup.
Při úvodním navštívení aplikace je uživatel přesměrován na stránku s přihlašovacím formulářem. Po zadání přihlašovacích údajů jsou poslány na server, který klientovi vrátí JWT token. Jakmile klient příjme token, nastaví jej do *cookies*[3] pro danou doménu stránky a přesměruje uživatele na hlavní stránku s obsahem aplikace. Token je posílán v hlavičce každého dotazu na zabezpečené zdroje serveru pro povolení přístupu.
Při každém vstupu do aplikace je kontrolováno, zda je v cookies nastaven JWT token. Pokud není k dispozici nebo pokud se blíží jeho expirace, tak je uživatel přesměrován na stránku s přihlašovacím formulářem.

Stahování a zobrazení záznamů

Stahování dat ze serveru probíhá na dvou místech. Pokud je uživatel přihlášen a přistoupí na hlavní stránku, tak jsou na straně Nuxt.js serveru stažena metadata o všech dostupných skupinách, které se vyskytují v RDF datech, aktuální uložená konfigurace filtrů uživatele a seznam všech skupin, které mají být zobrazeny na hlavní straně. Po stažení těchto dat a vyrenderování stránky jsou teprve poslány statické soubory uživateli.

[3]Cookies - data uložená v prohlížeči pro konkrétní doménu. Při každém přístupu na tuto doménu jsou cookies zaslány na server

Načítání záznamů zobrazených skupin probíhá až na straně prohlížeče, kdy se pro každou skupinu stáhne pouze jejich omezený počet. Tyto záznamy jsou zobrazeny jako seznam v kartě na hlavní straně 4.3 v komponentě *Vue Virtual Scroll List* 3.2.4. Pokud se v této komponentě dostaneme na konec stažených záznamů, automaticky se vyvolá akce na stáhnutí dalších. Výhodou této komponenty je vykreslování pouze omezeného počtu záznamů najednou a jejich překreslování při posunu v seznamu. Bez ní by se mohlo stávat, že bude prohlížeč vykreslovat najednou několik desítek tisíc záznamů a výrazně se tím sníží výkon aplikace. Výsledný vzhled B.1.

Filtrace a synchronizace záznamů

V předchozí sekci je pokryto stažení a zobrazení záznamů na hlavní stránce. V této sekci si popíšeme, jak se záznamy synchronizují podle času a filtrují podle typu obsahu. Filtrace podle typů obsahu probíhá v reálném čase z dat co jsou uloženy na straně uživatele.

Knihovna Vuex má speciální typ funkcí nazvaných *getters*, které slouží k derivaci dat ze stavu aplikace. Pokaždé, když změníme stav aplikace, tak se tyto funkce spustí a Vue.js si uloží jejich výsledek do paměti. Jakmile nějaký *getter* zavoláme v kódu aplikace, vrátí nám před vypočtený výsledek co je již uložen v paměti. Téhle funkcionality je využito při filtrování a synchronizaci dat. Důležité je, že nemění data, ze kterých derivuje výsledek.

V aplikaci je tedy vytvořen *getter*, který vrací nový seznam objektů se skupinami záznamů splňující kritéria zapnutých filtrů v aplikaci. Filtrování je implementováno jako obyčejný cyklus se sadou podmínek, který prochází postupně záznamy a vytváří z nich nový seznam skupin záznamů.

V případě zapnuté časové synchronizace se nejdříve vytvoří nový seznam objektů 5.3 obsahující všechna metadata zobrazených skupin bez záznamů. Poté funguje algoritmus pro synchronizaci následovně:

- 1. Vyber čas nejnovějšího záznamu ze všech skupin.
- 2. Procházej všechny skupiny a z každé z nich přidej záznam do nového objektu pokud je čas jeho zveřejnění v rozmezí pár minut od nejnovějšího záznamu. Je třeba zaznačit si, že tento záznam již nemáme procházet při další iteraci. Pokud není jeho zveřejnění v rozmezí pár minut od nenovějšího, přidej do nového objektu prázdný záznam.
- 3. Opakuj krok 1, ale bez záznamů, které jsou uloženy v novém objektu. Opakuj dokud nejsou všechny záznamy v novém objektu.
- 4. Vrať výsledný objekt.

```
[
    {
        id: '12', // id skupiny zaznamu
        title: 'ab', // nazev skupiny zaznamu
        itemsCnt: 200, // celkovy pocet zaznamu v celem datasetu
        loading: false, // zda skupina zrovna nacita nove zaznamy
        entries: [], // seznam zaznamu obsazenych ve skupine
    }, ...
]
```

Výpis 5.3: Nový objekt bez záznamů sloužící jako kontejner pro synchronizovaná data.

Administrace skupin záznamů

Stránka s administrací skupin záznamů je rozdělená do dvou záložek a to na zobrazené a nezobrazené skupiny na hlavní straně. Je implementovaná jako Vuetify *dialog* [4]. Pro zobrazení dialogu čeká na globální událost, která se spouští po kliknutí na tlačítko na hlavní straně v levém vysouvacím panelu. 4.3

V první záložce administrace jsou všechny nezobrazené skupiny záznamů na hlavní straně s možností kliku na tlačítko pro jejich zobrazení. Ve druhé jsou zobrazené záznamy z hlavní stránky s tlačítkem na jejích odstranění. Jednou z možnosti ve druhé záložce je měnění pořadí skupin jejich přetahováním. Tahle funkce je implementována pomocí VueDraggable 3.2.4. Výsledný vzhled B.3.

Zobrazení detailů záznamu

Tato část je implementovaná stejně jako v 5.2.3 dialogem a naslouchá na událost pro jeho zobrazení. Událost se spustí klikem na záznam na hlavní straně.

Po přijmutí události se stáhnou všechny detaily záznamu a k němu všechny záznamy, které obsahují stejnou informaci. Výsledný vzhled B.2.

5.2.4 Spuštění

Stejně jako u serverové části je využito Dockeru pro automatické nainstalování všech potřebných závislostí a spuštění Nuxt.js serveru.

5.3 Spuštění všech částí aplikace jako celku

Jak serverová, tak i klientská část využívá Docker. V této chvíli je možné použít Docker-Compose, který dokáže spustit více kontejnerů najednou a usnadnit si tím vývoj a nasazení. V této aplikaci používáme dva typy Docker-Compose konfiguračního souborů. Jeden z nich spustí pouze serverovou a klientskou část a očekává poskytnutí údajů o adrese PostgreSQL databáze a serveru s RDF daty. Obsahuje také proměnnou značící produkční prostředí, podle které Django vypne debugovací komentáře. Druhému stačí pouze adresa serveru s daty a PostgreSQL si spouští s pomocí dalšího Docker kontejneru, který je již definovaný a není třeba psát Dockerfile. První z nich slouží pro nasazení aplikace na produkci, kde chceme mít perzistentní PostgreSQL uložiště na externím místě, které je stabilní. Druhý slouží pro lokální vývoj, kdy jsou data v PostgreSQL databázi smazána při opakovaném sestavení kontejneru.

Vzhledem k tomu, že je implementováno *"cachování"* dat do Redisu, tak je v obou Docker Compose souborech Redis automaticky spuštěn a není s ním třeba nic nastavovat.

[4] Dialog - vyskakovací okna na webové stránce, které překrývá její obsah.

Kapitola 6

Zhodnocení

Tato kapitola pojednává o způsobu testování dílčích částí aplikace. V první sekci je popsáno testování serverové části, která slouží jako zdroj dat a ve druhé testování grafického uživatelského rozhraní. Dále jsou zde navrhnuty i rozšíření, s nimiž by bylo možné aplikaci vylepšit.

6.1 Testování serverové části

Serverová část byla testována iterativně v průb2hu jejího vývoje. Skládá se z REST rozhraní a z tohoto důvodu jsem zvolil testování této části pomocí programu Postman [1]. Tento program slouží k posílání HTTP dotazů na zvolený server a jejich ukládání pro případné znovupoužití.

Vytvořil jsem si tedy sadu HTTP dotazů pro získání JWT tokenu, stáhnutí a aktualizaci uživatelské konfigurace a SPARQL dotazů pro načtení dat přes proxy na straně serveru. Uložené dotazy jsem poté opakovaně posílal při vetších změnách serverové části.

Při testování jsem nenarážel na žádné problémy a serverová část se jeví jako stabilní a funkční.

6.2 Testování uživatelského rozhraní

Testování uživatelského rozhraní je důležité, jelikož je třeba aby všechny prvky aplikace spolu bez problémů spolupracovaly. Testování probíhalo iterativně při každé změně kódu klientské i serverové části. Jako testovací prostředek z mé strany posloužily webové prohlížeče Firefix 59, Safari 11.1 a Chrome 66 a jako hlavní body na které jsem kladl důraz patří:

- Výkon aplikace při velkém množství záznamů.
- Správné zobrazení napříč prohlížeči.
- Funcionalita filtrů, synchronizace záznamů a jejich správné zobrazení.
- Funkční přihlašování, stahování/nastavení/aktualizování uložené konfigurace uživatele.

[1] Postman - HTTP klient

Výkon celkové aplikace byl při testování dostačující, ale při zobrazení více skupin záznamů na hlavní straně si lze všimnout občasné trhavé animace při jejich procházení. Tento problém nastává častěji pokud je zapnuta synchronizace záznamů podle času napříč skupinami nebo pokud je použit prohlížeč Safari. I přes tento výkonnostní pokles je ale aplikace použitelná. Problém je zaviněn velkým množstvím výpočtů, které na straně klienta vykonává pro splnění filtrace a synchronizace záznamů a také tím, kolik záznamu může mít potenciálně uložených v paměti.

Při testování jsem nenarazil na žádné odchylky v zobrazení aplikace v různých prohlížečích.

Další bod na který byl kladen důraz je filtrování a synchronizování záznamů, kde se nacházely občasné problémy s jejich špatným zobrazením. Ve finální verzi byly tyto nedostatky odstraněny.

Posledním bodem je testování funkčního přihlašování, stahování/nastavení/aktualizování uložené konfigurace uživatele. Zde probíhalo vše bez problémů a aplikace tyto akce zvládala.

Testování provedl také vedoucí práce, ale byla mu poskytnuta pouze verze aplikace, ve které nebyla zpřístupněna synchronizace záznamů podle času napříč skupinami. Kromě připomínky o potřebě přidání této chybějící funkcionality mě nebyly sděleny další podstatné nedostatky.

6.3 Možná rozšíření

Na aplikaci je možné nadále pracovat a jedny z rozšíření, které můžou být dle mého názoru zajímavé jsou:

- Povolení vytváření nových uživatelů z uživatelského rozhraní

  V této práci je třeba vytvářet nové uživatele aplikace přímo na straně serveru hostujícím danou aplikaci a to zadáním příslušného příkazu. I když je aplikace cílena na uzavřenou skupinu lidí seznámených s projektem TimelineAnazyler a nečeká se, že budou aplikaci využívat stovky uživatelů, tak i přesto je registrace na serveru značně nepraktická. Proto je vhodné implementovat do uživatelského rozhraní možnost přihlášeným uživatelům vytvořit uživatele nové a případně smazat již existující.

- Přesun práce s daty na server

  V této práci jsou všechny výpočty na zpracování dat a všechna data na straně klienta. Tohle má za následek při některých situacích sníženou výkonnost aplikace, čemuž by bylo dobré zabránit. Jako řešení navrhuji přesunout všechny výpočty na práci se záznamy, jako je jejich stahování, filtrování a synchronizace na stranu serveru. Server by tato data mohl na základě uživatelských akcí posílat na klienta s nízkou režií pomocí WebSocketů [2], které dovolují obousměrnou komunikaci. Tímhle by si mohl klient efektivně žádat o data a server by mu je poslal. Tohle řešení má výhodu v tom, že na straně serveru je možné disponovat lepšími HW [3] prostředky a tím pádem zpracovávat větší množství záznamů najednou.

[2] WebSocket - komunikační protokol poskytující obousměrnou komunikaci
[3] HW - hardware

Kapitola 7

Závěr

V této práci byla vytvořena moderní webová aplikace skládající se z klientské (front-end) a serverové (back-end) části. Aplikace slouží jako prohlížeč RDF dat vygenerovaných nástrojem TimelineAnalyzer a je určena pro skupinu lidí, která je s tímto nástrojem seznámena.

Před vývojem aplikace jsem analyzoval požadavky na aplikaci, vstupní data včetně nástroje TimelineAnalyzer a rozebral jednotlivé technologie použitelné k jejímu uskutečnění. Poté jsem na základě sepsaných technologií a požadavků navrhl celkovou architekturu aplikace a způsob komunikace mezi jejími částmi. Při návrhu uživatelského rozhraní byl brán zřetel na jeho přehlednost a použitelnost.

Jakmile byla celá aplikace navrhnuta, tak jsem ji implementoval a otestoval. Ve výsledku si dovoluji říci, že práce po implementaci splňuje zadané požadavky.

Literatura

[1] Django REST framework: a powerful and flexible toolkit for building Web APIs. [Online; navštíveno 13.05.2018].
URL http://www.django-rest-framework.org/

[2] HTML. [Online; navštíveno 12.05.2018].
URL https://en.wikipedia.org/wiki/HTML

[3] Introduction to JSON Web Tokens. [Online; navštíveno 12.05.2018].
URL https://jwt.io/introduction/

[4] JavaScript HTML DOM. [Online; navštíveno 11.05.2018].
URL https://www.w3schools.com/js/js_htmldom.asp

[5] RDF4J. [Online; navštíveno 10.05.2018].
URL http://rdf4j.org

[6] Redis. [Online; navštíveno 11.05.2018].
URL https://redis.io/

[7] Requests: HTTP for Humans. [Online; navštíveno 11.05.2018].
URL http://docs.python-requests.org/en/master/

[8] Vue Virtual Scroll List: A vue component that support big data list with high scroll performance. [Online; navštíveno 11.05.2018].
URL https://github.com/tangbc/vue-virtual-scroll-list

[9] Abramov, D.: Redux. [Online; navštíveno 11.05.2018].
URL https://redux.js.org/

[10] Aranda, M.: What's the difference between JavaScript and ECMAScript? Říjen 2017, [Online; navštíveno 14.05.2018].
URL https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5

[11] Burget, R.: Sociální sítě: Sběr a analýza dat v souvislosti s bezpečnostními incidenty. Technická zpráva, 2017.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11573

[12] Corp., A.: Trello. [Online; navštíveno 11.05.2018].
URL https://trello.com/

[13] Gensler, P.: Creating Sandbox Environments for R with Docker. Listopad 2017, [Online; navštíveno 10.05.2018].
URL https://towardsdatascience.com/creating-sandbox-environments-for-r-with-docker-def54e3491a3

[14] Inc., F.: React: A JavaScript library for building user interfaces. [Online; navštíveno 05.05.2018].
URL https://reactjs.org/

[15] Inc., G.: Angular: develop accross all platforms. [Online; navštíveno 06.05.2018].
URL https://angular.io/

[16] Kanezawa, N.; Rauch, G.; Kovanen, T.: Next.js. [Online; navštíveno 15.05.2018].
URL https://zeit.co/blog/next

[17] Leider, J.: Material Design Component Framework. [Online; navštíveno 11.05.2018].
URL https://vuetifyjs.com/en

[18] Lohmann, S.; Link, V.; Marbach, E.; aj.: WebVOWL: Web-based Visualization of Ontologies. [Online; navštíveno 10.05.2018].
URL http://vowl.visualdataweb.org/webvowl.html

[19] Miller, E.: An Introduction to the Resource Description Framework. Květen 1998, [Online; navštíveno 10.05.2018].
URL http://www.dlib.org/dlib/may98/miller/05miller.html

[20] Padilla, J.: JSON Web Token Authentication support for Django REST Framework. [Online; navštíveno 11.05.2018].
URL https://github.com/GetBlimp/django-rest-framework-jwt

[21] Prud'hommeaux, E.; Seaborne, A.: SPARQL: Query Language for RDF. Leden 2008, [Online; navštíveno 15.05.2018].
URL https://www.w3.org/TR/rdf-sparql-query/

[22] Sozo, D.: 10 reasons to use Nuxt.js for your next web application. Březen 2018, [Online; navštíveno 15.05.2018].
URL https://medium.com/vue-mastery/10-reasons-to-use-nuxt-js-for-your-next-web-application-522397c9366b

[23] You, E.; aj.: Vue.js: The Progressive JavaScript Framework. [Online; navštíveno 11.05.2018].
URL https://vuejs.org/v2/guide/

[24] You, E.; aj.: What is Vuex? [Online; navštíveno 11.05.2018].
URL https://vuex.vuejs.org/en/intro.html

[25] Zabriskie, M.; contributors: Axios: Promise based HTTP client for the browser and node.js. [Online; navštíveno 11.05.2018].
URL https://github.com/axios/axios

[26] Štencek, J.: *Principy sémantického webu*. [Online; navštíveno 10.05.2018].
URL http://vse.stencek.com/semanticky-web/ch03s05.html

Přílohy

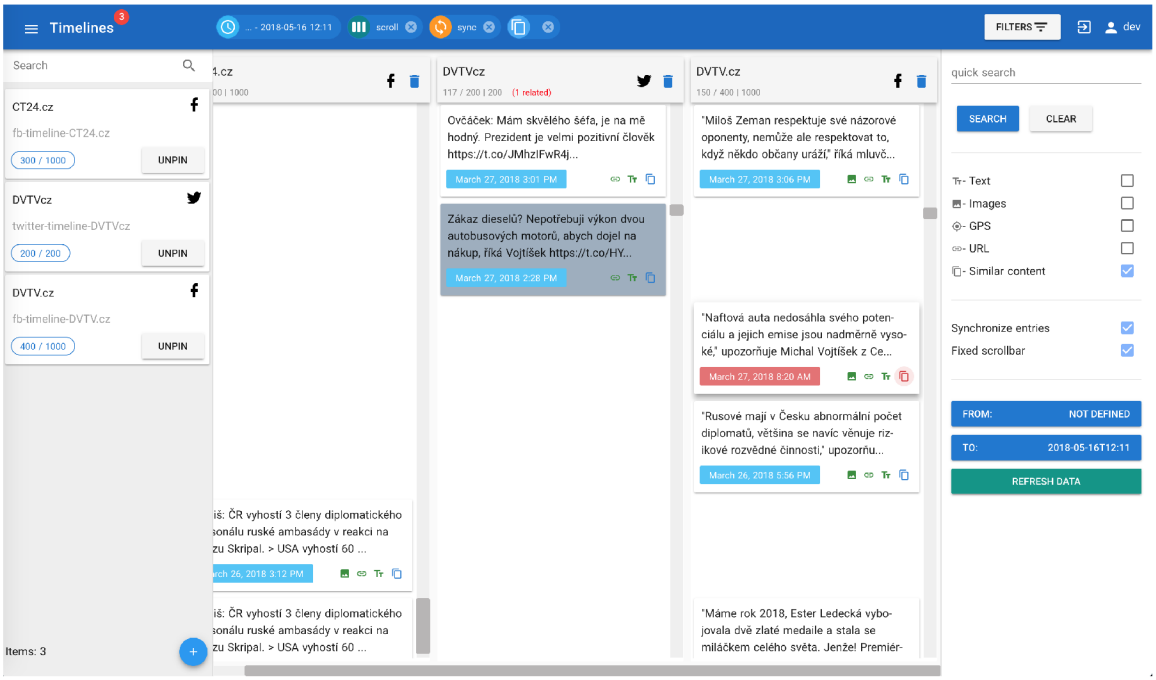
Příloha A

Obsah přiloženého CD

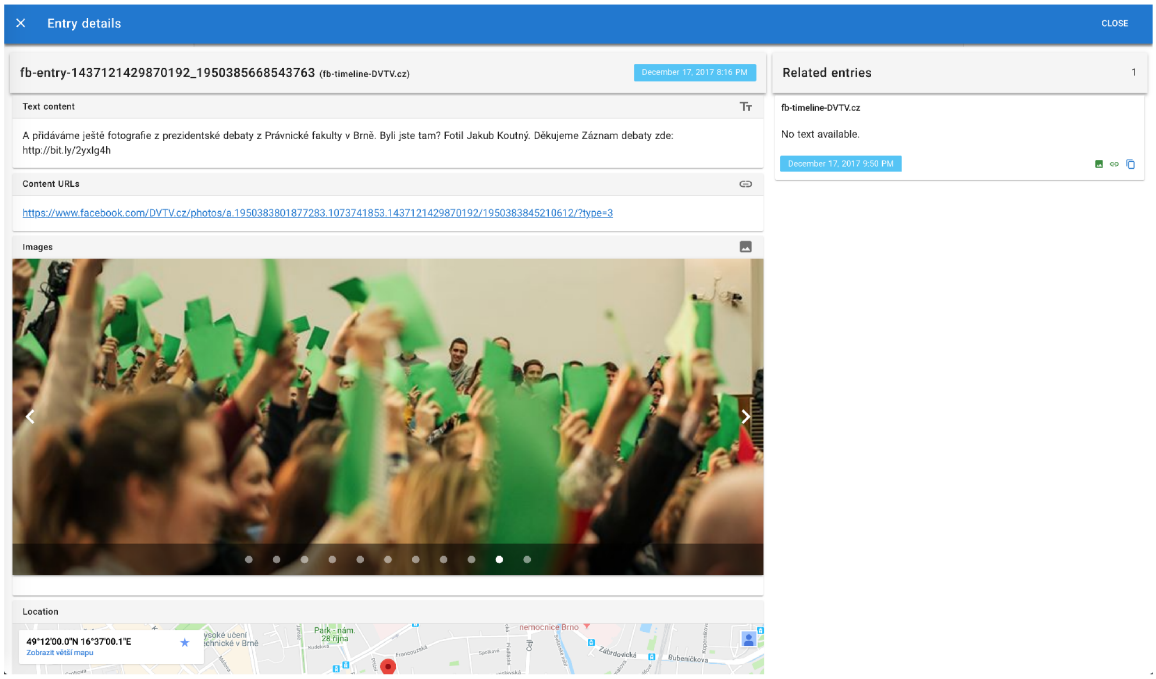
- **./backend/** - soubory pro serverovou část aplikace
- **./frontend/** - soubory pro klientskou část aplikace
- **./docs/** - .tex soubory a technická zpráva
- **./docker-compose.dev.yaml** - konfigurace pro spuštění aplikace na lokálním stroji
- **./docker-compose.prod.yaml** - konfigurace pro spuštění aplikace na produkci
- **./README.md** - pokyny pro spuštění

Příloha B

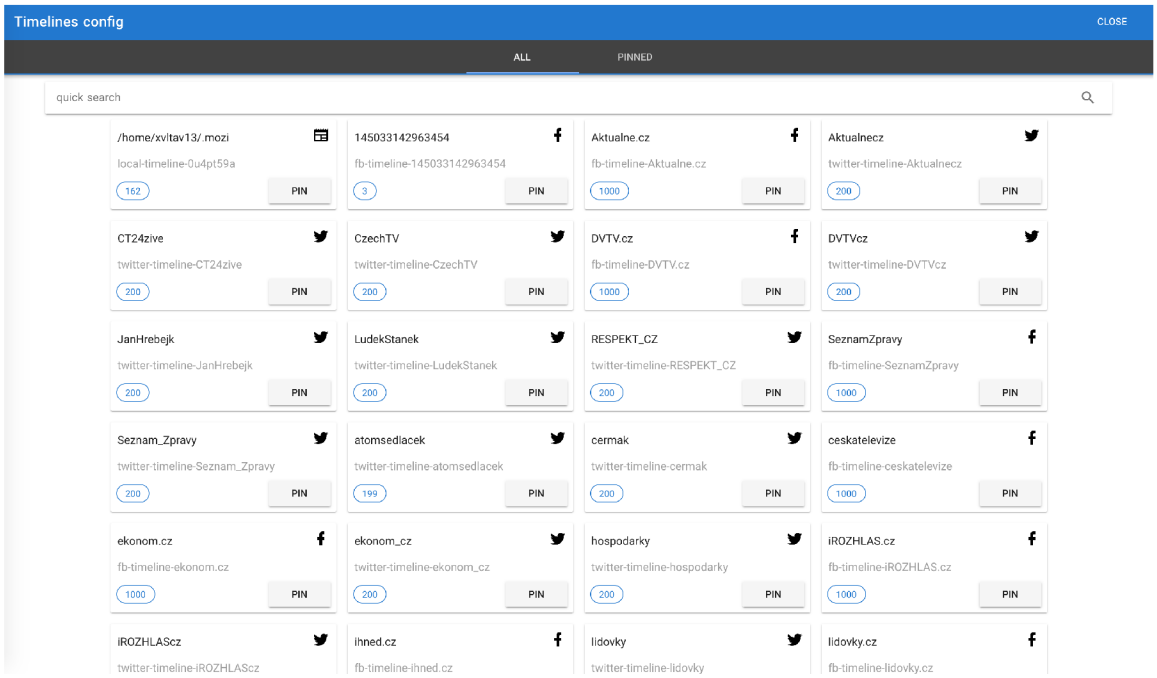
Konečný vzhled aplikace



Obrázek B.1: Hlavní strana se synchronizovanými záznamy a značením souvisejících záznamů.



Obrázek B.2: Detail záznamu včetně náhledu záznamu se stejnou informací.



Obrázek B.3: Administrace skupin záznamů.