



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

**WEBOVÉ ROZHRANÍ NÁSTROJE PRO ANALÝZU
DOKUMENTŮ**

WEB INTERFACE OF A DOCUMENT ANALYSIS TOOL

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

ADAM ŠEVČÍK

VEDOUcí PRÁCE

SUPERVISOR

RADEK BURGET, Ing.,Ph.D.

BRNO 2022

Zadání bakalářské práce



Student: **Ševčík Adam**
Program: Informační technologie
Název: **Webové rozhraní nástroje pro analýzu dokumentů**
Web Interface of a Document Analysis Tool
Kategorie: Web

Zadání:

1. Prostudujte současné technologie pro tvorbu klientských webových aplikací v jazyce JavaScript a existující řešení pro anotaci webových dokumentů.
2. Seznamte se s experimentálním nástrojem FitLayout, jeho aplikačním rozhraním a způsobem reprezentace dokumentů.
3. Na základě konzultací s vedoucím navrhnete klientskou aplikaci pro zobrazení zpracovaných dokumentů a výsledků analytických dotazů.
4. Implementujte navržené řešení pomocí vhodně zvolených technologií.
5. Proveďte testování vytvořené aplikace na množině testovacích dokumentů.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Swicegood, T.: Programming Node.js, O'Reilly, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015
- Dokumentace projektu FitLayout: <https://github.com/FitLayout>

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**

Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.

Datum zadání: 1. listopadu 2021

Datum odevzdání: 11. května 2022

Datum schválení: 11. října 2021

Abstrakt

Táto práca sa zaoberá vývojom webového rozhrania pre nástroj na analýzu webových dokumentov. Toto rozhranie má za cieľ nahradiť už existujúcu, zastaralú desktopovú aplikáciu FitLayout. Z FitLayout preberá back-end a pridáva vlastný front-end. Ide o aplikáciu postavenú pomocou JavaScriptu, HTML a vzhľad stránky je dosiahnutý pomocou CSS a frameworku Bootstrap. Popis webových dokumentov je vo formáte RDF serializovaného do JSON-LD. V tomto formáte prebieha aj výmena dát medzi serverovou a klientskou časťou. Výsledok tejto práce umožňuje užívateľovi analyzovať webové dokumenty vo svojom prehliadači rovnako v desktopovej aplikácii.

Abstract

This thesis deals with the development of a web interface of a web document analysis tool. Goal of this interface is to replace old desktop application FitLayout. Own front-end is added to FitLayout back-end. This application is made by Javascript, HTML and visual part is done by CSS and framework Bootstrap. Web documents are in format RDF serialized in JSON-LD. Data are exchanged in the same format between client and server. Result of this thesis enables user to analyse documents in browser same way as in desktop application.

Klíčové slová

webová aplikácia, Javascript, analýza dokumentov, FitLayout, HTML, CSS, Bootstrap

Keywords

document analysis, Javascript, web application, FitLayout, HTML, CSS, Bootstrap

Citácia

ŠEVČÍK, Adam. *Webové rozhraní nástroje pro analýzu dokumentů*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Radek Burget, Ing.,Ph.D.

Webové rozhraní nástroje pro analýzu dokumentů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána docenta Radka Burgeta. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....

Adam Ševčík
10. mája 2022

Podakovanie

Moje podakovanie patrí vedúcemu práce, profesorovi Radekovi Burgetovi za jeho rady, doporučenia a za pomoc pri tvorbe tejto práce.

Obsah

1	Úvod	3
2	Súčasn� technol�gie na tvorbu webov�ch aplik�ci�	5
2.1	Frontend	5
2.1.1	V�vojov� vzory	5
2.1.2	HTML	7
2.1.3	CSS	8
2.1.4	JavaScript	9
2.1.5	AngularJS	10
2.1.6	React	10
2.1.7	Vue.js	10
2.1.8	Bootstrap	11
2.2	Backend	12
2.2.1	Javascript	12
2.2.2	Python	12
2.2.3	PHP	13
2.2.4	Java	13
2.3	�al�ie technol�gie	13
2.3.1	RDF - Resource Description Framework	13
2.3.2	DOM - Document Object Model	14
3	FitLayout	16
3.1	Z�kladn� vlastnosti	16
3.2	Z�kladn� koncepty	17
4	N�vrh rie�enia	19
4.1	Po�iadavky na pou�itie aplik�cie	19
4.2	Architekt�ra	20
4.3	Existuj�ce rie�enia	20
4.3.1	MarkUp - www.app.markup.io	20
4.4	N�vrh GUI	21
5	Implement�cia rie�enia	23
5.1	Pou�it� v�vojov� n�stroje	23
5.2	Kroky implement�cie	24
5.3	Bli��ie op�sanie jednotliv�ch krokov	25
6	Testovanie	35

7 Záver	38
Literatúra	39
A Obsah pamäťového média	41
B Manuál k praktickej časti	42
B.1 Stiahnutie potrebných balíkov	42
B.2 Konfigurácia	42
B.3 Spustenie	42
B.4 Použitie	42

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce je návrh a implementácia klientskej časti webového rozhrania ktoré má za úlohu nahradiť zastaralú desktopovú aplikáciu FitLayout. FitLayout je framework slúžiaci na vytváranie aplikácií a algoritmov slúžiacich na analýzu renderovaných webových dokumentov. Tento framework je od roku 2015 vyvíjaný na FIT VUT v Brne pod vedením pána docenta Radka Burgeta.

Prvým krokom zadania bolo naštudovanie súčasných technológií pre tvorbu webových aplikácií a naštudovanie existujúcich riešení analýzy webových dokumentov. Pre lepšie porozumenie procesu tvorby webových aplikácií som naštudoval ako klientskú tak aj serverovú časť aj keď cieľom práce je vytvorenie iba klientskej časti. Tento krok je popísaný v kapitole 2. Najskôr sa čitateľ oboznámi s tým čo je to frontend alebo aj klientská časť aplikácie. Ďalej sa oboznámi s tým aké vývojové vzory sa používajú pri tvorbe frontendu a nakoniec získa prehľad o mnou naštudovaných technológiách ktoré sa používajú pri vytváraní tejto časti aplikácie. V druhej časti kapitoly sa čitateľ dozvie čo je backend a aké technológie sa používajú pri jeho tvorbe. Pri pokuse o nájdenie existujúcich riešení som zistil, že ide o veľmi špecifickú aplikáciu a nič podobné zatiaľ neexistuje. Z toho dôvodu som túto časť zadania zobral ako možnosť inšpirácie pri tvorbe užívateľského rozhrania z aplikácií ktoré aspoň okrajovo pripomínajú funkcionality FitLayout. Tento proces je popísaný v kapitole 4 v sekcii 4.3.

Ďalším krokom bolo zoznámenie sa s frameworkom FitLayout, jeho aplikačným rozhraním a spôsobom akým reprezentuje dokumenty. Tento krok je popísaný v kapitole 3. V tejto kapitole je najskôr priblížený samotný framework aj s ukázkou desktopovej aplikácie a neskôr sú popísané jeho základné vlastnosti a koncepty jednotlivých častí frameworku.

Po naštudovaní potrebných znalostí bolo ďalším krokom navrhnuť samotné rozhranie. Tento návrh je popísaný v kapitole 4. Prvou časťou návrhu je presný popis požiadavok na používanie aplikácie. V tejto časti sa čitateľ dozvie ako sa bude aplikácia používať a ako sa bude správať. V ďalšej časti nazvanej architektúra je tak ako z názvu vyplýva opísaný návrh architektúry. Po opise architektúry nasleduje tak ako bolo povedané vyššie popis aspoň konceptovo podobných aplikácií ako je FitLayout a inšpirácia z ich užívateľského prostredia. Záverečnú časť tejto kapitoly tvorí návrh GUI kde sa čitateľ najskôr oboznámi s tým čo to GUI je a neskôr zistí ako som ja navrhoval to moje.

Po návrhu riešenia nasledovala samotná implementácia. Tento proces je popísaný v kapitole 5 ktorá sa skladá z dvoch väčších častí. Tou prvou je popis a priblíženie použitých vývojových nástrojov a druhou, hlavnou sú kroky implementácie. Najskôr sú kroky popísané v skratke, aby čitateľ pochopil dôvod každého kroku a potom sú všetky kroky popísané podrobne aj s technológiami ktoré sa pri nich použili a ich ukázkami.

Ďalšia kapitola, teda kapitola 6 sa zaoberá testovaním. V tejto kapitole sa čitateľ oboznámi s tým ako bola aplikácia testovaná a aké sú dosiahnuté výsledky.

Práca končí kapitolou 7 v ktorej sú zhrnuté dosiahnuté výsledky a takisto spomenuté možné rozšírenia rozhrania do budúcnosti.

Kapitola 2

Súčasné technológie na tvorbu webových aplikácií

Webová aplikácia je softvér, ktorý beží na webovom serveri namiesto lokálneho operačného systému zariadenia. Skladá sa z klientskej a serverovej časti (alebo aj frontendu a backendu) ktoré medzi sebou komunikujú. V tejto kapitole priblížim návrhové vzory, technológie a formáty ktoré sa v súčasnosti používajú na tvorbu webových aplikácií.

2.1 Frontend

Frontend (alebo klientská časť aplikácie) je všetko, čo užívateľ (klient) vidí na svojom zariadení. Ide napríklad o text, obrázky, tlačidlá a tiež aj všetko ostatné, čo aplikácia vykonáva v rámci užívateľovho prehliadača.

2.1.1 Vývojové vzory

Pri vývoji užívateľského rozhrania existuje niekoľko vzorov pomocou ktorých môže byť logika kódu rozdelená. V nasledujúcej časti popíšem tri najpoužívanejšie vzory.

Model-View-Controller

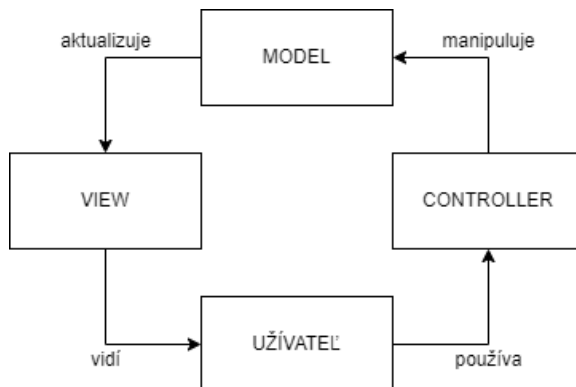
Model-View-Controller alebo častejšie označovaný ako MVC je jeden z najčastejšie používaných architektonických vzorov používaných na tvorbu webových aplikácií. Základnou myšlienkou je oddelenie logiky od výstupu. Tak ako názov napovedá je rozdelený na tri časti a to **Model**, **View** a **Controller**.

Model obsahuje logiku aplikácie. Môže to byť napríklad validácia, komunikácia s databázou alebo jednoduché výpočty. Jeho úlohou je prijatie vonkajších parametrov a poslanie dát ďalej do View. O tom odkiaľ tieto parametre prichádzajú model nevie.

View sa stará o zobrazenie dát používateľovi. Vo webových aplikáciach je táto časť najčastejšie HTML dokument prípadne dokument v inom značkovacom jazyku. View môže obsahovať aj minimálne množstvo logiky potrebnej na zobrazenie obsahu. Podobne ako **Model** ani **View** nevie odkiaľ dáta ktoré zobrazuje prichádzajú a stará sa len o to aby boli zobrazené užívateľovi.

Controller je akýsi spojovník s ktorým komunikuje ako užívateľ tak aj model a view. Stará sa o to aby model a view nemuseli priamo medzi sebou komunikovať a zároveň aby boli obe tieto časti informované o aktuálnom stave tej druhej.

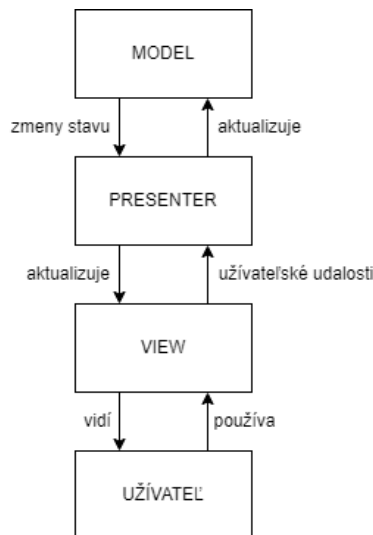
Takto rozdelená aplikácia zlepšuje čitateľnosť kódu, uľahčuje testovanie a rozširovanie a umožňuje pracovať viacerým programátorom na jednej aplikácii nezávisle na sebe. [14]



Obr. 2.1: Model-View-Controller

Model-View-Presenter

Model-View-Presenter (MVP) je derivácia vzoru MVC ktorá sa snaží o úplné oddelenie Model a View. Namiesto toho aby Controller získaval vstup od používateľa a View zobrazoval výstup, vzor MVP používa View ako jediný nástroj komunikácie s užívateľom. **Presenter** sa nachádza za View a stará sa o všetkú funkcionalitu. Keď View získa nejaké dáta pošle ich do Presenteru ktorý následne aktualizuje Model. Nakoniec Presenter získa nové dáta z aktualizovaného Modelu a pošle ich späť do View na zobrazenie užívateľovi.[14]

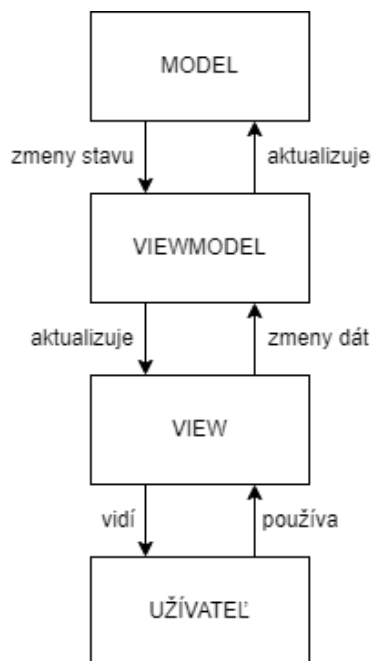


Obr. 2.2: Model-View-Presenter

Model-View-ViewModel

Model-View-ViewModel (MVVM) je podobne ako MVP derivácia vzoru MVC ktorá sa snaží o oddelenie Model a View častí. V tomto vzore je namiesto časti Presenter časť ViewModel ktorá obsahuje všetok kód ktorý by sa inak nachádzal v časti View. To v praxi znamená,

že časť View môže byť nahradená statickou HTML šablónou stránky ktorá bude použitá na tvorbu užívateľského rozhrania. [14]



Obr. 2.3: Model-View-Presenter

2.1.2 HTML

HTML - Hypertext Markup Language (alebo Hypertextový značkovací jazyk) je štandardný jazyk pre tvorbu webových dokumentov. Značkovacie jazyky sú jazyky, v ktorých prevláda všeobecný text. Do tohto textu sú vložené štrukturované úseky pomocou značiek, ako napríklad nasledujúca značka odkazu:

Do textu je vložený `odkaz`.

Každá webová stránka je primárne tvorená dokumentom v jazyku HTML, ktorý môže byť prípadne rozšírený o ďalšie dáta a rozšírenia. Najnovšia existujúca verzia HTML je v súčasnosti HTML5[3]

Aplikácia, ktorá tento dokument prekladá sa nazýva Webový prehliadač. Ideálne, on-line obsah webovej stránky by mal vyzeráť rovnako, nebdajúc na to, na ktorom operačnom systéme je prehliadač spustený, prípadne, ktorý webový prehliadač sa používa. Tento cieľ platformovej nezávislosti je v praxi dosiahnutý len čiastočne, t.j. niektoré špecifické funkcie alebo časti nefungujú pri všetkých prehliadačoch rovnako. Klasický HTML dokument vyžaduje minimálne tieto štyri elementy:

- `<html> ... </html>`
- `<head> ... </head>`
- `<title> ... </title>`
- `<body>...</body>`

Vyššie uvedené elementy definujú esenciálnu časť HTML dokumentu: samotný dokument, hlavičku, nápis/titulok a telo. Každý z týchto elementov je definovaný dvojicou tagov (značiek) - začiatočným a koncovým tagom. Tagy sú vždy umiestnené v lomených zátvorkách: <...>. Koncový tag začína lomítkom (/). Ako bude ukázané neskôr, niektoré HTML elementy majú iba jeden tag. Väčšina tagov by sa mala objavovať v pároch, aj keď toto pravidlo je pri HTML dodržiavané len zľahka. Ak chceme podporovať skriptovacie jazyky, ako napríklad JavaScript (popísaný nižšie), ďalší element musí byť pridaný k základným štyrom:

```
<script>...</script>
```

Tieto elementy sú v HTML dokumente zorganizované nasledovne:

```
<html>
  <head>
    <title> ... </title>
    ...
    <!-- Voliteľné elementy skriptu podľa potreby. -->
    <script> ... </script>
  </head>
  <body>
    ...
  </body>
</html>
```

Tag <html> uzatvára všetky ostatné tagy a definuje hranice HTML dokumentu. Tag <script> sa často objavuje vo vnútri tagu <head>, ale takisto sa môže objaviť aj inde v dokumente. Odsadzovanie tagov nie je povinné, ale robí dokument lepšie čitateľným. [1]

2.1.3 CSS

CSS - cascading style sheets (alebo kaskádové štýly) definujú zvlhad dokumentu nezávisle na jeho obsahu. Vzhľad je definovaný samostatne, čo umožňuje ľahkú modifikáciu. Môže byť definovaný pre celú množinu stránok, čo prináša konzistentný vzhľad týchto stránok. Jednoducho, môže byť vytvorený nový, alebo alternatívny vzhľad. CSS obsahuje jednotný spôsob definície štýlu pre všetky elementy, čo zaručuje jednoduchosť a prehľadnosť. Prvá verzia (CSS 1) vznikla v roku 1996 a momentálne sa CSS nachádza vo verzii 4.15. Style sheets, teda štýlovy predpis, je súhrn pravidiel určujúcich štýl dokumentu. Každý dokument je popísaný stylesheetom z troch zdrojov:

- štýl pripravený tvorcom stránok
- užívateľský štýl - vlastná definícia užívateľom
- štýl prehliadača

Každý štýl obsahuje pravidlá týkajúce sa rôznych elementov, typicky viac pravidiel pre každý element. Relevantné pravidlá sa zoradia do **kaskády** a kompletne definujú výsledný vzhľad elementu.

Štýlový predpis je postupnosť pravidiel typu:

```
selektor { deklarácia vlastností }
selektor { deklarácia vlastností }
selektor { deklarácia vlastností }
...
```

Kde **selektor** určuje, pre ktoré elementy dané pravidlo platí a **vlastnoti** definujú vizuálne (prípadne iné) vlastnosti dotyčných elementov. Existuje niekoľko typov selektorov:

- ***** - pravidlo platí pre všetky elementy
- **meno elementu** - pravidlo platí pre všetky elementy daného mena
- **trieda elementu** - pravidlo platí pre všetky elementy s danou hodnotou atribútu class (. na začiatku)
- **identifikátor elementu** - pravidlo platí pre všetky elementy s danou hodnotou atribútu id (# na začiatku)

Tvar deklarácie vlastností je:

```
vlastnosť: hodnota; vlastnosť: hodnota;
```

[2] Príklad predpisu:

```
* { margin: 0; padding: 0; }
body, html { height:100%; overflow: hidden; }
.list { float:left; width: 20%; height: 100vh; border: 1px solid black; }
#to_right{ right: 0; }
```

2.1.4 JavaScript

JavaScript je interpretovaný ako objektovo orientovaný programovací jazyk, ktorý bol vyvinutý na používanie popri iných webových nástrojoch. JavaScript nefunguje ako samotný jazyk. Je dizajnovaný tak, aby pracoval spolu s HTML na vytváranie reaktívnych webových stránok. Je rozdielny od Javy, ktorá je kompikovaný objektovo orientovaný jazyk. Keďže je súčasťou frontendu, kód nieje posielaný užívateľovi, ale je rovno vykonávaný riadok po riadku v JavaScript prekladači. Tento prekladač sa nachádza v užívateľovom webovom prehliadači. Takýto návrh minimalizuje bezpečnostné hrozby, ktoré vznikajú, keď užívateľov počítač interaguje s počítačom, ktorý stránku odoslal. Zároveň, uľahčuje to zabaliť celý problém s vlastným užívateľským rozhraním a riešením, do jedného vlastného dokumentu. Nemožnosť interagovať dynamicky s informáciami uloženými na serveri limituje to, čo JavaScript dokáže.

JavaScript je jedným z triedy skriptovacích jazykov, ktorých účelom je prístup a modifikovať už existujúce informačné prostredie. V tomto prípade, toto prostredie je HTML dokument. Hneď ako sa HTML dokumenty vyvinuli z jednosmerných systémov na zobrazovanie štatických informácií, bolo potreba niečo ako JavaScript. Jedným z prvých použití pre tento jazyk bola potreba kontrolovať hodnoty vložené užívateľom do HTML formulára. JavaScript môže porovnať vložené hodnoty voči predpokladanej množine hodnôt a generovať vhodnú správu, prípadne vykonať inú akciu, práve na základe tohto porovnania. Neskôr sa vyvinul na kompletný jazyk schopný manipulovať text a rôzne matematické operácie, vhodný na široký okruh problémov. Ako bolo spomenuté vyššie, limitovaný je tým, že nemá prístup k externým informáciám, či už ide o informácie na lokálnom alebo vzdialenom počítači. [1]

JavaScript frameworky

2.1.5 AngularJS

Angular je open source JavaScript-ový projekt, ktorý sa používa na vytváranie JavaScriptových aplikácií. Jeho cieľom je rozšíriť ich na základe modelu MVC (vysvetleného nižšie) a zjednodušiť testovanie a vývoj. Funguje tak, že prenesie všetok obsah zo serveru do prehliadača a načíta všetky stránky zároveň. Keď sú stránky načítané, kliknutie na odkaz nenačíta celý obsah stránky odznova, ale jednoducho prepíše kontrétnu sekciu stránky. [8]

Spolu s bohatou štandardnou knižnicou bolo pre Angular napísaných veľa používateľských rozšírení. Existuje dokonca aj špeciálny framework, ktorý značne zjednodušuje programovanie mobilných aplikácií, ako multiplatformové riešenie pre Android, iOS či Windows phone. Dôležitou súčasťou Angularu je testovanie. Pomocou Angularu boli vybudované stránky ako Google, G-mail či Paypal.

2.1.6 React

React je open source knižnica/framework postavená na JavaScripte. Vznikla v roku 2013 v sídle Facebooku a dnes patrí k jednej z najpoužívanejších knižníc/frameworkov. Tak ako Angular, funguje na základe modelu MVC (model view controler), ktoré sa venuje hlavne view zložke. Vďaka Reactu je vytváranie dynamických aplikácií veľmi jednoduché a intuitívne. Funguje rovnako ako Angular, a to na princípe, že je prepísaná iba požadovaná časť stránky, nie celá stránka. Týmto sa urýchľuje vykresľovanie. Základným blokom stavby React aplikácie sú znova použiteľné komponenty. Jedna aplikácia sa skladá z viacerých komponentov, ktoré je veľmi jednoduché upraviť a znova použiť, čo značne urýchľuje vývoj aplikácií. Facebook vydal rozšírenie do prehliadača Chrome, ktoré slúži na debuggovanie React aplikácií, čo uľahčuje a urýchľuje vývoj.[7]

2.1.7 Vue.js

Vue je framework a zároveň ekosystém, ktorý pokrýva skoro všetky elementy front end developingu. Tak ako aj ostatné frameworky, stavia na HTML, CSS a JavaScripte. Je deklaratívny a založený na komponentoch, čo pomáha k efektívnosti, či už pri veľkých alebo malých projektoch. [12]

V nasledujúcom príklade môžeme vidieť jednoduché počítadlo klikov, vytvorené pomocou Vue.js, ktoré vždy pri kliknutí na tlačidlo pripočíta jeden. Skladá sa z dvoch častí, a to JavaScriptovej časti a Template časti, ktorá obsahuje HTML kód spojený s JavaScript prvkami. Tento template je vždy kompilovaný na vysoko optimalizovaný JavaScript kód.

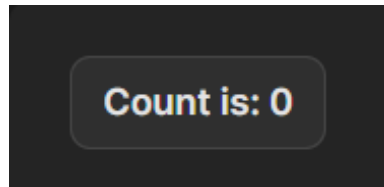
Časť JavaScript

```
import { createApp } from 'vue'
```

```
createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
```

Template

```
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```



Obr. 2.4: Výsledok kódu

Na tomto príklade môžeme vidieť dva základné prvky, a zároveň aj výhody Vue.js.

- **Deklaratívne renderovanie:** Vylepšenie schopností čistého HTML, ktoré dovoľuje deklaratívne popisovať HTML prvky na základe JavaScriptu.
- **Reaktivita:** Vue automaticky sleduje stav JavaScriptu a efektívne updatuje DOM vždy, keď nastane zmena.

2.1.8 Bootstrap

Bootstrap je framework jazyka CSS. Inak povedané, je to sada nástrojov, ktorá využíva CSS, HTML a JavaScript na jednoduchšie a lepšie formátovanie webových stránok. Vznikol v auguste v roku 2011 v spoločnosti Twitter, z dôvodu potreby zjednotiť toolset frontend developerov v celej firme. Dnes je to celosvetový open source produkt, ktorý využíva veľké množstvo developerov. Vyvinul sa z čisto CSS frameworku, až na dnešnú podobu, kedy dokáže pracovať s JavaScript pluginmi. [16]

Bootstrap sa momentálne nachádza vo verzii 5, ktorá vyšla v roku 2021. V dnešnej dobe využíva tento framework mnoho známych spoločností, ako napríklad Spotify, Twitter, Docker a Intel, a to najmä z týchto dôvodov:

- Konzistentný framework, ktorý podporuje väčšina moderných prehliadačov (Chrome, Firefox, Edge, Safari, and Opera).
- Malá veľkosť.
- Vysoká customizácia.
- Responzívne štruktúry a štýly.
- Mnoho JavaScript pluginov.
- Dobrá dokumentácia.
- Veľa profesionálnych šablón zdarma.
- Dobrý grid systém s 12 stĺpcami.[16]

Hlavné časti Bootstrapu

- **Kontajnery:** Jedna z hlavných častí Bootstrapu. Pridávajú CSS vlastnosti, ako sú margin, padding a alignment HTML elementom.
- **Farby:** Bootstrap zjednodušil používanie farieb, kedy prednastavil 6 základných farieb, a to: muted, primary, success, info, warning a danger. Tieto farby idú použiť ako na pozadie, tak aj na text.
- **Responzívne stĺpce:** Bootstrap má jeden až dvanásť responzívnych stĺpcov, ktoré sa v prípade, že sa nezmestia na obrazovku, samé usporiadajú do viacerých riadkov pod seba.
- **Tabuľky:** Jednoduché tvorenie ľubovoľnej tabuľky, ktorá je automaticky ohraničená a má striedavé farby riadkov pre lepšiu čitateľnosť.
- **Hlásenia:** Preddefinované farebne rozdelené hlásenia, ktoré sa ľahko vložia do kódu.
- **Tlačidlá:** Preddefinované moderne vyzerajúce tlačidlá rôznych veľkostí a farieb. [6]

2.2 Backend

Backend (alebo serverová časť) je to, čo bežný užívateľ nevidí. Je to administratívna časť stránky, do ktorej sa dostane len autorizovaný užívateľ.

2.2.1 Javascript

Tak ako frontend, aj backend môže byť vytvorený pomocou JavaScriptu, a to za pomoci napríklad Node.js.

Node.js

Node.js je runtime environment (spúšťacie prostredie) pre asynchrónne webové aplikácie napísané v jazyku JavaScript. Node.js beží na V8 JavaScript engine, ktorý je základom Google Chrome, mimo webového prehliadača. Je jednovláknový, čo znamená, že vždy beží len jeden proces bez možnosti spustenia viacerých vlákien súčasne. Ak Node.js vykonáva nejaké čítacie alebo zapisovacie operácie, ako napríklad čítanie z databázy, alebo prepisovanie databázy, namiesto toho aby blokoval vlákno a mrhal CPU vláknami, Node.js pokračuje s operáciou až pokým nepríde odpoveď. Takýmto spôsobom kompenzuje nemožnosť vytvorenia viac thredov, čo by mohlo viesť k veľkému množstvu ťažko zistiteľných chýb. Jedna z najväčších predností tohto prostredia je **npm** alebo Node Package Manager. Je to správca voľne dostupných balíčkov/knižníc, ktoré podstatne rozširujú schopnosti samotného Node.js.[13]

2.2.2 Python

Python je interpretovaný, objektovo-orientovaný jazyk s dynamickou sémantikou. Jeho výhoda je jednoduchá, ľahko naučiteľná syntax, čo znižuje cenu na obstarávanie programu. Python podporuje veľké množstvo modulov a balíkov, ktorými sa podstatne zvyšuje jeho použiteľnosť. [15]

Tak ako JavaScript, aj Python má mnoho frameworkov. Medzi najznámejšie patrí Django, Falcon alebo Flask.

Django

Django je jeden z najpopulárnejších frameworkov jazyka python. Podporuje rýchly vývoj a čistý pragmatiký dizajn aplikácií.

2.2.3 PHP

PHP je skriptovací, objektovo orientovaný programovací jazyk, ktorý slúži najmä na tvorenie serverovej časti dynamických webových aplikácií. Mnohí ľudia ho považujú za hybridný jazyk, keďže preberá syntax z jazykov ako Perl, C alebo Java a vytvára silný a intuitívny kód. [9]

PHP je takzvaný „embedded“ jazyk, čo v praxi znamená, že je vložený do iného jazyka, v tomto konkrétnom prípade do HTML. Tak ako bolo povedané v sekcii 2.1.2 HTML kód, skladá sa vždy najmenej zo štyroch základných tagov. Pre pridanie PHP scriptu stačí začať tagom `<?php` a ukončiť `?>` ako v nasledujúcej ukážke, ktorá vypíše do tela HTML dokumentu vetu: Ahoj, ako sa máš?

```
<html>
  <head>
    <title> ... </title>
  </head>
  <body>
    <?php
      echo "Ahoj, ako sa máš?";
    ?>
  </body>
</html>
```

2.2.4 Java

Java je objektovo orientovaný jazyk vyvíjaný spoločnosťou Oracle. Je postavený podľa C++ tak, aby bol čo najkratší, najjednoduchší a prenositeľný medzi platformami a operačnými systémami. To je dosiahnuté tak, že zdrojový kód nie je kompilovaný do strojového kódu, ale do takzvaného „byte-code“. Tento „byte-code“ neskôr vykonáva Java Virtual Machine, čo je interpreter a kompilátor jazyka Java zároveň. Java je často spájaná s webovým prehliadačom zvaným HotJava.[10]

Existuje niekoľko základných balíkov jazyka Java:

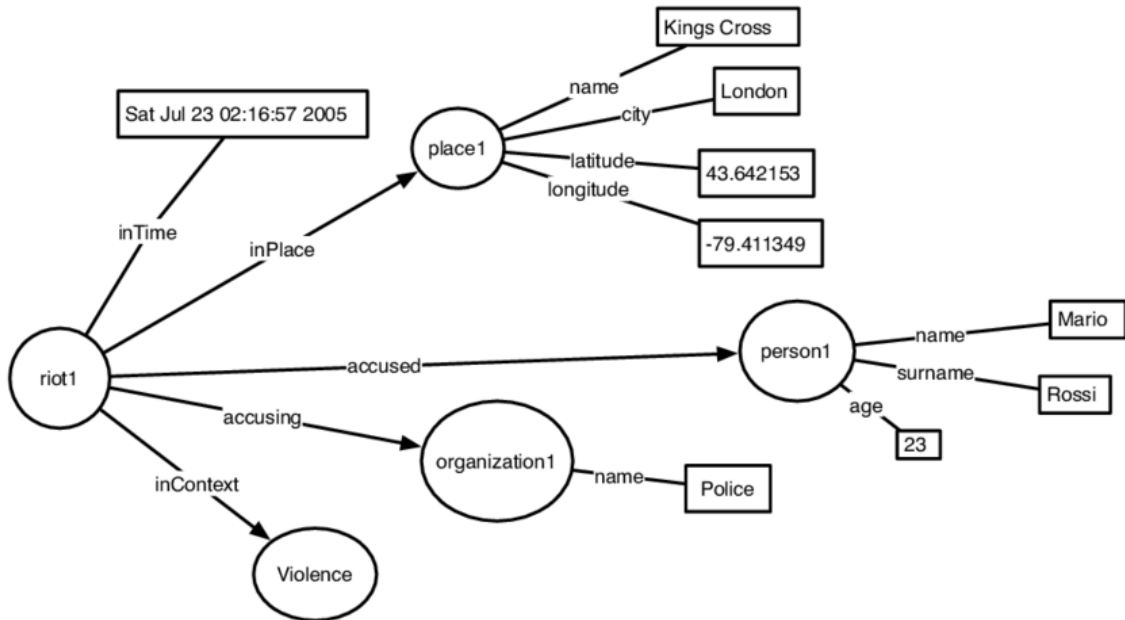
- **Java ME (Micro Edition)** - mobilné telefóny a malé zariadenia
- **Java SE (Standard Edition)** - domáce počítače
- **Java EE (Enterprise Edition)** - podnikové počítače
- **Java Card** - inteligentné čipové karty (napríklad SIM)

2.3 Ďalšie technológie

2.3.1 RDF - Resource Description Framework

RDF je štandard pre vymieňanie dát a popisovanie zdrojov na webe. Je štandardizovaný konzorciom W3C. Spôsob, akým RDF spája dáta dokopy, je pomocou trojíc (triples). Tieto

trojice sú potom identifikované pomocou URI, čím sa z nich vlastne stávajú štvorice (quads). V bežnom jazyku sú tieto trojice v podstate vetné členy (podmet, prísudok, predmet ...), spojené určitými vzťahmi. Spájanie týchto trojíc vytvára orientovaný graf, kde hrany predstavujú spojenie medzi trojicami a uzly predstavujú samotné trojice. Tento graf sa často používa ako RDF vizualizácia informácií pre jeho jednoduchosť.



Obr. 2.5: RDF graf reprezentujúci informácie o proteste.

2.3.2 DOM - Document Object Model

Pri písaní HTML dokumentu, skoro vždy balíme nejaký HTML obsah do iného HTML obsahu. Takýmto štýlom vzniká hierarchia, ktorá sa dá reprezentovať ako strom. V HTML dokumente je táto hierarchia často vizualizovaná pomocou indentovania (odsadenia) jednotlivých častí.

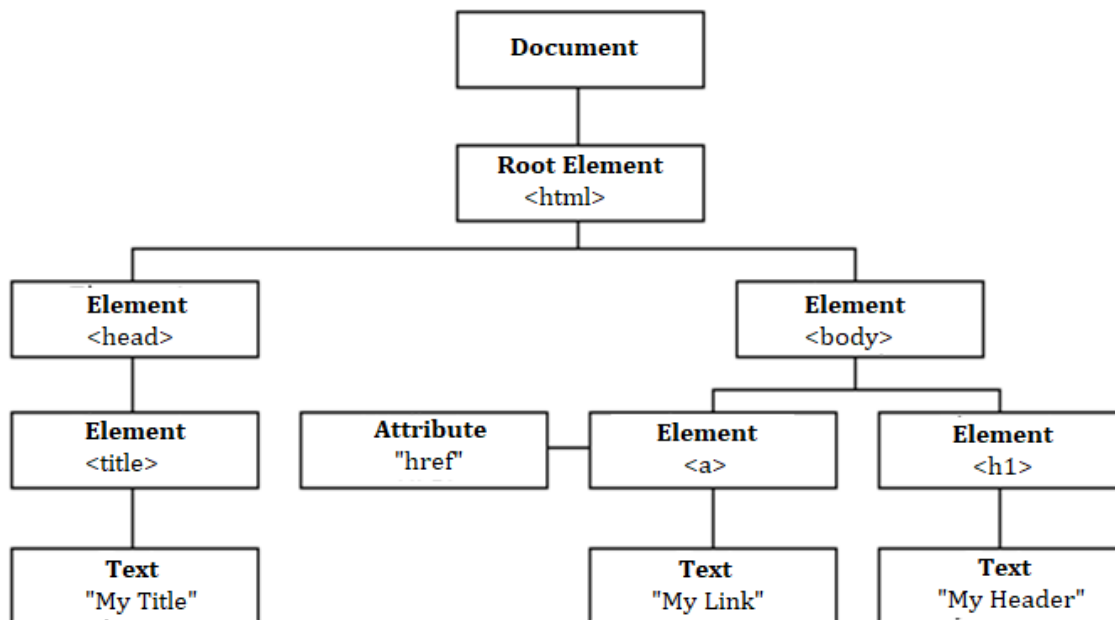
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>HTML</title>
  </head>
  <body>
    <!-- Telo stránky -->
  </body>
</html>
```

Keď prehliadač načíta takýto HTML dokument, interpretuje a parsuje túto hierarchiu, aby vytvoril strom objektových uzlov, ktorý simuluje ako sú jednotlivé časti usporiadané. Podstatou tohto procesu je poskytnúť prostredie pre úpravu (pridávanie, mazanie, eventy ...) HTML dokumentu pomocou JavaScriptu. [11]

Najzákladnejšie typy uzlov, ktoré sa nachádzajú v DOM sú:

- **DOCUMENT_NODE** Základný objekt, všetky ostatné sa nachádzajú v ňom (window.document).
- **ELEMENT_NODE**: Jednotlivé elementy (tagy) v dokumente (<body>, <a>, <style>, <h1>).
- **ATTRIBUTE_NODE**: Vlastnosti elementov (class="big_red", id="top").
- **TEXT_NODE**: Všetok text vo vnútri HTML dokumentu.
- **DOCUMENT_FRAGMENT_NODE**: Podobný ako DOCUMENT_NODE s tým rozdielom, že nie je súčasťou aktívneho dokumentu, ale je ro samostatná časť, ktorú môžeme meniť bez toho, aby sa menil originálny dokument.
- **DOCUMENT_TYPE_NODE**: Určuje typ dokumentu (<!DOCTYPE html>).

Výsledná vizualizácia DOM stromu za použitia týchto typov potom vyzerá takto:

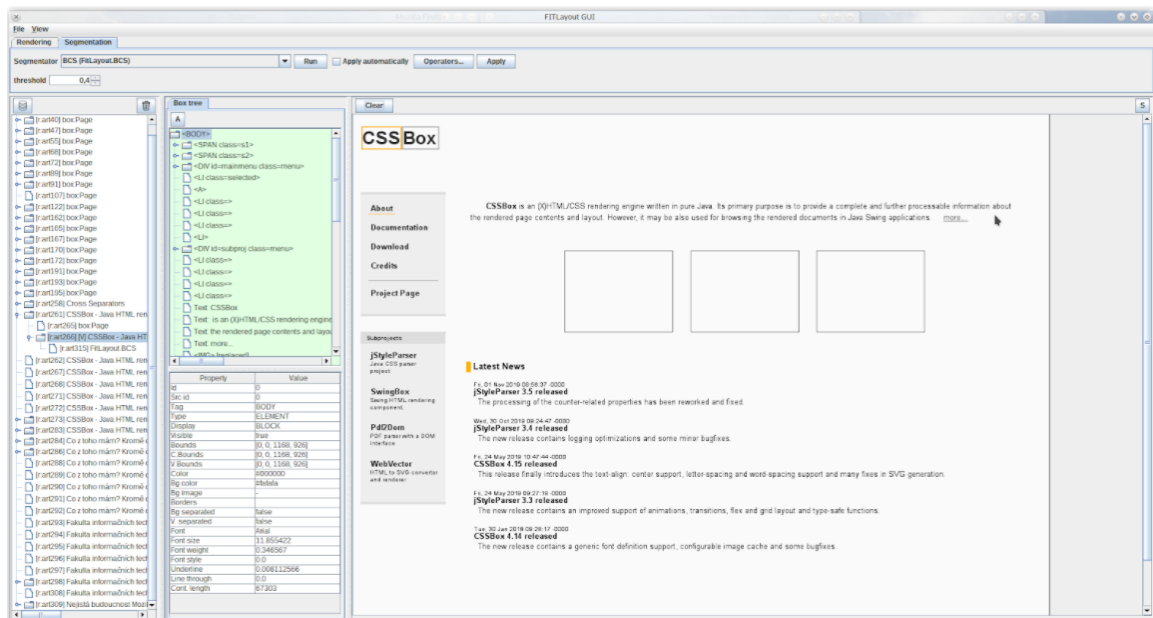


Obr. 2.6: Príklad HTML DOM stromu.

Kapitola 3

FitLayout

V tejto kapitole sa čitateľ oboznámi s nástrojom FitLayout. Tento nástroj je základom tejto bakalárskej práce. K už existujúcej desktopovej aplikácii znázornenej na obrázku 3.1 bolo treba vytvoriť webové rozhranie. Následujúci text vysvetľuje základnú funkcionality a základný koncept nástroja FitLayout.



Obr. 3.1: Desktopová aplikácia FitLayout

3.1 Základné vlastnosti

FitLayout je rozšíriteľný framework na vytváranie aplikácií a algoritmov, ktoré analyzujú renderované webové dokumenty. Zjednodušuje a automatizuje úlohy spojené s renderovaním webových dokumentov. Jeho základnú funkciu sú:

- Java API reprezentujúca vyrenderované stránky a ich obsah, segmentáciu stránok a s nimi spojených artefaktov.

- Lahká integrácia s aplikáciami napísanými v inom jazyku ako Java, pomocou príkazového riadka (CLI) alebo webovej API (REST).
- Proces na spracovanie dokumentov od renderovania, segmentáciu, post-processing, anotáciu a iné analytické kroky, až po uloženie výsledku a exportovanie do rôznych formátov.
- Implementácia rôznych backendov pre renderovanie stránok a segmentačné metódy.
- Ontologická implementácia RDF reprezentácie dokumentov a ich uskladnenie.[4]

3.2 Základné koncepty

Artefakty

Artefakty reprezentujú produkt každého kroku počas procesu spracovania, či už ide o render, segmentáciu, anotáciu alebo iné. Artefakt je vždy vytvorený **Artefaktovou službou** 3.2. Môže byť vytvorený buď od nuly, napríklad vyrenderovaním stránky na základe jej URL, alebo odvodený z rodiča, ako napríklad segmentácia už vyrenderovanej stránky. Na základe takéhoto postupu vytvárania artefaktov vzniká strom artefaktov s koreňom (typicky stránkou) a od neho odvodených dedičných artefaktov, ktoré sú získané ako výsledok analýzy rodiča.[5]

FitLayout definuje dátový model na reprezentáciu jednotlivých artefaktov. Skladá sa z dátových štruktúr, ktoré sú definované ako Java interfaces a platformovo-nezávislého ontologického modelu. [5]

Existuje niekoľko typov artefaktov, a to:

- **Page alebo box tree** - reprezentuje vyrenderovanú stránku. Typický vzniká vyrenderovaním webového dokumentu. Reprezentuje stránku ako strom boxov, kde každý box vzniká pomocou renderu backendu z DOM modelu.
- **Area tree** - reprezentuje výsledok segmentácie stránky. Vzniká zo segmentačných metód použitých na vyrenderovanú stránku, alebo ako výsledok post-processingu iného area tree. Uchováva vizuálne oblasti, ktoré zodpovedajú vizuálnym blokom detekovaným na stránke. Strom zodpovedá ich hierarchii. Pre segmentačné metódy, ktoré používajú plochý model, ako napríklad BCS, má strom iba dva levely, a to koreň a listové uzly.
- **LogicalAreaTree** - reprezentuje výsledok logickej interpretácie artefaktu typu area tree. Obsahuje logické oblasti, ktoré reprezentujú sadu vizuálnych oblastí. Ich vzťah reprezentuje akýkoľvek logický vzťah ktorý medzi nimi je.
- **Text Chunk Set** - množina textových chunkov, ktoré boli extrahované z vizuálnych plôch. Každý takýto chunk reprezentuje obdĺžnikovú oblasť obsahujúcu kontinuálny text.[5]

Identifikácia artefaktov

V rámci repozitára má každý artefakt svoj unikátny identifikátor IRI (Internationalized Resource Identifier). Vo väčšine prípadov má IRI formát URL s predponou **http://fitlayout.github.io/resource/**, ako napríklad **http://fitlayout.github.io/resource/art261**. V

špeciálnych prípadoch môže byť použitý aj iný formát. Tento formát je používaný len ako interný identifikátor a na webe sa nič pod týmto URL nenachádza. Každý artefakt má typ identifikovaný pomocou IRI a odkazuje na konkrétny typ popísaný vyššie. Okrem toho, môže mať každý artefakt rodičovský artefakt, na ktorý je tak isto odkazované pomocou IRI.[5]

Repozitár artefaktov

Jednotlivé artefakty sú uložené v úložisku na ďalšie spracovanie. Toto úložisko je implementované pomocou **in-memory storage** čo znamená, že artefakty sú len dočasne ukladané medzi jednotlivými krokmi spracovania.

Pre zložitejšie prípady existuje úložisko založené na RDF, zvané RDFArtifactRepository, ktoré uchováva RDF reprezentáciu každého artefaktu. Je založené na RDF4J a podporuje nasledujúce úložiská:

- In-memory RDF úložisko
- Natívne RDF úložisko v lokálnych súboroch
- Remote úložisko na RDF4J serveri pomocou HTTP[5]

Artefaktové služby

Artefaktové služby sú moduly, ktoré produkujú artefakty daného typu. Produkujú ich z iného zdrojového artefaktu. Každá služba má vlastný unikátny identifikátor, ktorý ju umožňuje invokovať. Tento identifikátor je uchovávaný s vytvoreným artefaktom. Ďalej majú služby definovanú sadu vstupných parametrov rôznych typov, ktoré sú popísané v dokumentácii konkrétnej služby. Ich hodnoty musia byť definované pred ich invokáciou. [5]

Najdôležitejšie služby sú:

- **Renderovanie backedn implementácie:** FitLayout.CSSBox, FitLayoutPuppeteer
- **Segmentačné metódy:** FitLayout.BCS, FitLayout.VIPS, FitLayout.BasicAreas
- **Box tree post-processing:** FitLayout.VisualBasicTree
- **Area tree post-processing:** FitLayout.ApplyOperators[5]

Serializácia artefaktov

Každý artefakt má serializovanú formu, ktorá sa používa na export, import alebo uschovanie artefaktov. FitLayout definuje ontologický model, ktorý umožňuje vytváranie RDF popisov artefaktov, ktoré môžu byť neskôr serializované do akéhokoľvek RDF formátu, ako napríklad RDF/XML alebo Turtle. Artefakty môžu byť takisto serializované do XML, HTML, alebo dokonca PNG obrázku.[5]

Kapitola 4

Návrh riešenia

Po naštudovaní technológií na tvorbu webových aplikácií a samotného nástroja FitLayout, som pokračoval vyhľadávaním a študovaním existujúceho riešenia. V kapitole návrh riešenia opíšem použitie aplikácie, architektúru, existujúce riešenia a nakoniec návrh GUI.

4.1 Požiadavky na použitie aplikácie

Pri spustení aplikácie je z konfiguračného súboru načítaný identifikátor predvoleného repozitára. Po príchode užívateľa na stránku, uvidí načítané artefakty z vybraného repozitára a aj samotný identifikátor. Pri tomto identifikátore sa nachádza tlačidlo **Change repository**, ktoré po zmene hodnoty identifikátora a kliknutí zmení repozitár, s ktorým chce užívateľ pracovať. O prípadnom neúspechu zmeny je informovaný a repozitár zostáva pôvodný. Ak zmena prebehne úspešne, načíta sa v ľavom stĺpci nový zoznam artefaktov.

Nad zoznamom artefaktov sa nachádza tlačidlo **Create artifact**, po ktorom stlačení sa zobrazí užívateľovi modálne okno s formulárom pre pridanie nového artefaktu do aktuálne používaného repozitára. Toto okno obsahuje kolónku na vloženie URL nového artefaktu a ďalej šírku (width), výšku (height) a službu (service), pomocou ktorej má byť artefakt vyrenderovaný. Služby má užívateľ na výber dve, a to **Puppeteer** a **CSSBox**. Po vyplnení všetkých políček a stlačení tlačidla **Create** je užívateľ informovaný o prípadnej chybe, alebo v opačnom prípade uvidí pridanie nového artefaktu do zoznamu na ľavej strane. V zozname artefaktov, pri ich identifikátore, sa nachádza tlačidlo **SEG**. To slúži na ďalšiu segmentáciu vybraného artefaktu. Nachádza sa tu tiež tlačidlo červeného krížika, pri ktorom je po stlačení zvolený artefakt vymazaný zo zoznamu a takisto z repozitára.

Po kliknutí na identifikátor jedného z artefaktov je artefakt načítaný, čo je vizuálne prezentované točiacim sa koliečkom. Po ukončení načítavania je stránka vykreslená do stredného panelu aplikácie a všetky jej boxy sú vykreslené v strome na pravej hornej strane stránky. V tejto vykreslenej stránke sa môže užívateľ ľubovoľne pohybovať a klikať na jednotlivé boxy. Po kliknutí na jeden box je tento box v stránke vizuálne označený a takisto je označený v strome boxov. Pod stromom boxov sa nachádza tabuľka so všetkými podstatnými informáciami o momentálne vybratom boxe.

Na pravej strane vrchnej navigácie sa nachádza posledné tlačidlo **Search**. Po kliknutí na toto tlačidlo je podobne ako pri **Add artifact**, zobrazené modálne okno s jednou hodnotou **query**. Do tejto hodnoty užívateľ vpisuje SPARQL QUERY popísanú v časti 5.3. Pomocou tejto query môže užívateľ nájsť požadované boxy podľa určitej vlastnosti. Po vpísaní hodnoty, užívateľ klikne na tlačidlo **Search**. Pri chybe alebo žiadnom výsledku hľadania je

užívateľ informovaný, a naopak, pri úspechu sú v zozname artefaktov zvýraznené artefakty, v ktorých sa takýto box nachádza. Ak je momentálne nejaký artefakt vykreslený a obsahuje hľadané boxy, tak sú tieto boxy zvýraznené, ako na vykreslenej stránke, tak aj v strome boxov. Tieto boxy a artefakty sú na stránkach zvýraznené až pokiaľ užívateľ neklikne na tlačidlo **Remove search**, ktoré sa zobrazí pri tlačidle **Search** po úspešnom vyhľadaní.

4.2 Architektúra

Cieľom tejto bakalárskej práce je implementovať webové rozhranie pre nástroj **FitLayout**. Pre toto rozhranie je potreba vytvoriť frontend, ktorý bude komunikovať s backendom tohto nástroja. Tak ako bolo spomenuté v kapitole FitLayout 3, tento nástroj má vlastné REST API, vďaka ktorému je možné v mojej frontend aplikácii jednoducho využívať backend nástroja. Keďže je moja aplikácia písaná v jazyku JavaScript, bolo od začiatku jasné, že na komunikáciu s REST API budem využívať JavaScript API, zvanú **Fetch API**. Fetch API poskytuje JavaScriptu rozhranie na pristupovanie a manipuláciu k HTTP požiadavkám a odpovediam. Na komunikáciu sa využíva niekoľko základných typov HTTP requestov. Tieto typy sú **GET**, **POST** a **DELETE**. Samozrejme v HTTP komunikácii sa používajú aj ďalšie iné typy ale pre potreby tejto bakalárskej práce postačia tieto tri.

4.3 Existujúce riešenia

Pri hľadaní už existujúceho riešenia som prišiel k záveru, že FitLayout je veľmi špecifická a unikátna aplikácia, a nič podobné, čím by som sa mohol inšpirovať, zatiaľ neexistuje. Z tohto dôvodu som sa rozhliadal v širšom spektre a snažil sa nájsť webové aplikácie, ktoré slúžia na analýzu akýchkoľvek dokumentov, a tak sa inšpirovať aspoň ich GUI. Medzi aplikácie, ktoré aspoň čiastočne zodpovedali mojim kritériám, patria:

- **Kami (www.kamiapp.com)**: Aplikácia pre učiteľov a žiakov na spoločný feedback k dokumentom.
- **Annotate (www.annotate.com)**: Aplikácia na kolaboráciu pri vytváraní spoločných dokumentov.
- **hypothes.is (web.hypothes.is)**: Rozšírenie webového prehliadača na kolaboratívnu anotáciu webových dokumentov.

V nasledujúcej časti bližšie predstavím najviac sa podobajúcu aplikáciu k FitLayout.

4.3.1 Markup - www.app.markup.io

Markup je aplikácia, ktorá užívateľovi dovoľuje pridávať feedback, ako komentáre alebo anotácie k jednotlivým častiam webových stránok, PDF súborov a obrázkov. GUI tejto aplikácie je veľmi jednoduché a príjemné na používanie. Najskôr pri vstupe na stránku si užívateľ zvolí, či chce pracovať s webovou stránkou, alebo prípadne PDF či obrázkami. Po výbere je aplikácia rozdelená do dvoch, respektíve troch častí, kde v ľavom paneli užívateľ vidí všetku interakciu s anotovaným obsahom. Hlavnou časťou aplikácie je samotné vykreslenie dokumentu. Na jednotlivé časti dokumentu sa dá klikať, prípadne ich potiahnutím viac označovať a pridávať k nim feedback. Pri PDF dokumente sa potom nachádza aj tretia časť stránky, a to zoznam strán na pravo od vykresleného dokumentu.



Obr. 4.1: GUI aplikácie Markup

4.4 Návrh GUI

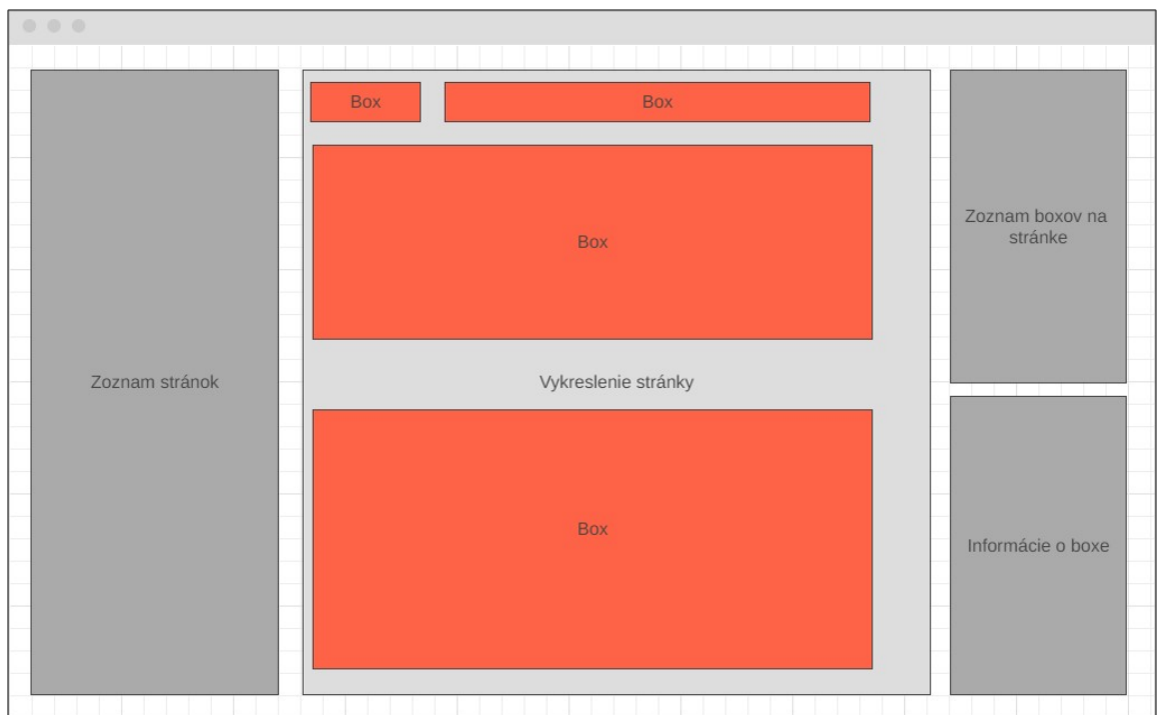
GUI, alebo inak užívateľské rozhranie, je dôležitou súčasťou každej aplikácie. Je to jediná časť, s ktorou sa bežný užívateľ stretáva, a s ktorou pracuje. Preto je potrebné, aby bolo čo najprehľadnejšie a najpohodlnejšie. Z tohto dôvodu som sa snažil vytvoriť čo najjednoduchšie a zároveň čo najfunkčnejšie rozhranie z ktorého užívateľ aj bez väčších inštrukcií pochopí ako sa aplikácia ovláda. Návrh **GUI** bol inšpirovaný nástrojom FitLayout 3 a podobnými aplikáciami spomenutými v časti 4.3. Samotný návrh užívateľského prostredia sa menil postupne, pridávaním nových funkcií rozhrania.

Pôvodný návrh obsahoval tri základné časti:

- **Ľavý panel**, na ktorom sa nachádza zoznam všetkých stránok (artefaktov) v úložisku.
- **Stredný panel**, na ktorý sa bude vykresľovať vybraný artefakt.
- **Pravý panel**, ktorý je rozdelený na dve časti. Vrchná časť slúži ako zoznam všetkých boxov vo zvolenom artefakte a spodná časť, na ktorej sa zobrazia informácie o konkrétnom vybranom boxe.

Postupne pri pridávaní ďalších funkcií vyšlo najavo, že do **GUI** bude treba pridať ďalšiu časť ktorá bude obsahovať viaceré ovládacie prvky ako napríklad pridávanie artefaktov alebo zmenu repozitára. Túto časť som sa nakoniec rozhodol pridať na vrch užívateľského rozhrania ako navigačnú lištu.

Konečný návrh **GUI** sa nachádza na obrázku 4.2.



Obr. 4.2: Základný návrh GUI

Kapitola 5

Implementácia riešenia

Táto kapitola sa zaoberá samotnou implementáciou aplikácie. Čitateľ sa najskôr dozvie ako vývojové nástroje som použil na rýchlejšiu a prehľadnejšiu implementáciu. V druhej časti popíšem najskôr kroky a dôvod týchto krokov implementácie v skratke a potom popíšem každý krok aj s technológiou pri ňom využitou dopodrobna.

5.1 Použité vývojové nástroje

Pre zjednodušenie a zlepšenie procesu vývoja webovej aplikácie sa často používajú rôzne nástroje napríklad na vytvorenie developerského serveru alebo rôzne kompilačné nástroje. Inak tomu nebolo ani v mojom prípade. V tejto časti popíšem konkrétne nástroje ktoré som sa rozhodol využiť pre jednoduchší a efektívnejší vývoj aplikácie.

Browserify

Browserify je open-source nástroj na kompiláciu a zhukovanie modulov štýlu node.js pre prehliadač. Rieši problém príliš veľkého množstva JS skriptov v HTML dokumente, problém nemožnosti použiť Node.js moduly v prehliadači a problém nemožnosti odkazovať na vlastné moduly vo vlastnom kóde. Podobne ako Node.js Browserify, združuje moduly do zväzku, v mojom prípade je to súbor **bundle.js**, ale na rozdiel od node.js zväzku, ktorý je určený na spúšťanie na serveri, je tento zväzok určený pre spustenie v prehliadači.

Princíp browserify

Browserify je tak isto modulom node.js, ktorý je potreba najskôr nainštalovať. Po nainštalovaní pomocou npm môžeme začať pracovať s nasledujúcim príkazom:

```
browserify script1.js script2.js -o bundle.js
```

Tento príkaz zoberie všetky deklarované scripty (v tomto prípade script1.js a script2.js) a vytvorí z nich jeden script, takzvaný bundle, deklarovaný za parametrom **-o**, v tomto prípade bundle.js.

Development server budo

Toto rozšírenie dodáva k Browserify vlastný vývojový server na vývoj aplikácie (development server), ktorý si zakladá na pravidelnom obnovovaní stránky, čo znamená, že má

integrovanej LiveReload. LiveReload znamená, že aplikácia je automaticky pri zmene a uložení súboru znova prekompilovaná a zabalená do bundle.js, a tak nie je potrebné vždy používať príkaz uvedený vyššie. Hlavnými výhodami sú:

- Vlastný vývojový server, ktorý automaticky načíta index.html nachádzajúci sa v aktuálnom priečinku.
- Rýchle zhlukovanie súborov. Ak zhlukovanie trvá dlhšie, budo zobrazí pôvodný bundle.js až pokiaľ nový nie je pripravený.
- Sledovanie aj HTML a CSS súborov.
- Zmena CSS súboru nezapríčiní reload, ale rovno zmenu na obrazovke.
- Jasné a presné chybové hlášky v konzole.

5.2 Kroky implementácie

Na základne analýzy požiadavkov a návrhu riešenia sme implementáciu rozdelili do niekoľkých krokov. Tieto kroky sú nasledujúce:

1. Parsovanie JSON-LD (vysvetlené nižšie) dokumentu podľa dodaného príkladu pomocou parseru.

Tak ako bolo v návrhu povedané, dáta sa z backendu FitLayout budú získavať pomocou REST API. Tieto dáta budú získavané vo formáte JSON-LD a z tohto dôvodu bolo treba vytvoriť prípadne najšť vhodný parser.

2. Vykresľovanie vlastného obsahu v prehliadači pomocou WebComponents.

Podľa návrhu je jednou z hlavných častí samotné vykresľovanie renderovanej stránky a preto bolo treba najšť vhodný spôsob vkladania vlastných elementov do stránky.

3. Vytvorenie boxov z rozparovaného JSON-LD a vykreslenie jednotlivých boxov artefaktu.

Ako bude popísané nižšie, výstup rozprasovaných dát z REST API nebol príliš vhodný na okamžité vykresľovanie jednotlivých boxov a preto ich najskôr bolo treba spojiť a vytvoriť objekty. Po vytvorení objektov som mohol začať s vykresľovaním konkrétnych boxov.

4. Použitie Fetch na získanie JSON-LD reprezentácie stránky (artefaktu) z backendu.

Podľa návrhu, komunikácia s backendom bude prebiehať pomocou REST API a preto bolo treba najšť spôsob ako komunikovať a vyskúšať komunikáciu medzi mojou aplikáciou a REST API.

5. Zobrazenie zoznamu všetkých artefaktov a boxov a vykreslenie zvoleného artefaktu..

Ako bolo spomenuté v sekcii 3.2 artefakty sú uložené v repozitároch. Preto je vhodné, aby boli zobrazené všetky artefakty z jedného repozitára na stránke a užívateľ tak mohol prechádzať medzi nimi a vyberať si ktorý chce zobraziť.

6. Interaktivita boxov.

Samotné vykreslenie boxov a stromu ich zoradenia mnoho informácií užívateľovi nedá, preto ďalším krokom bola ich interaktivita ktorá zahŕňa ich vyznačovanie a zobrazovanie informácií.

7. Pridávanie, mazanie, segmentácia artefaktov.

Tak ako boxy aj artefakty a repozitáre majú svoju interaktívnu časť popísanú v návrhu. Preto bolo ďalším krokom pridať možnosti pridávania a mazania artefaktov z repozitára a aj možnosť hlbšej segmentácie vybraného artefaktu.

8. Zmena repozitára.

Ak by užívateľ chcel pracovať s viacerými repozitármi bolo potreba pridať túto funkciu ktorá dovoľuje zmenu repozitára podľa jeho identifikátora.

9. Zasielanie SPARQL požiadavky.

Vyhľadanie boxu podľa konkrétneho parametru je veľmi zložitý a dlhý pretože sa užívateľ musí preklikať a skontrolovať konkrétny údaj manuálne. Z tohto dôvodu bolo treba pridať vyhľadávanie. Toto vyhľadávanie je implementované pomocou zasielanie SPARQL požiadavku na REST API.

10. Vizualná úprava

Prvá vec ktorú si užívateľ na každej aplikácii všimne je jej vizuálna stránka. Ani webová aplikácia na analýzu webových dokumentov nie je výnimka preto posledným bodom implementácie je vizuálna úprava.

Podrobný opis implementácie jednotlivých krokov je popísaný v nasledujúcej časti.

5.3 Bližšie opísanie jednotlivých krokov

V tejto časti bližšie popíšem každý krok aj s ukážkami a použitými technológiami pre lepšie porozumenie procesu implementácie.

1. Parsovanie JSON-LD dokumentu podľa dodaného príkladu pomocou parseru.

Informácie o boxoch a artefaktoch sú z backendu získavané vo formáte **JSON-LD**. Preto bolo treba nájsť alebo vytvoriť vhodný parser. Pred začatím práce mi vedúci práce poskytol ukážkové súbory vo formáte **JSON-LD** ktoré som si lokálne otváral a skúšal parsovať.

JSON-LD 1.1

JSON-LD je užitočný formát používaný na serializáciu a linkovanie dát. Je založený na veľmi známom a populárnom formáte JSON a jeho syntax je optimalizovaná tak, aby sa čo najľahšie dalo prejsť v aplikáciách používajúcich JSON na JSON-LD. Je primárne určený na používanie dát s odkazmi vo webovom prostredí, tvorenie webových aplikácií a ukladanie takýchto dát s odkazmi v databázach používajúcich formát JSON, ako napríklad MongoDB. Momentálne sa JSON-LD nachádza vo verzii 1.1. [18] Príklad popisu človeka vo formáte JSON-LD:

```
{
```

```

"@context": "https://json-ld.org/contexts/person.jsonld",
"@id": "http://dbpedia.org/resource/John_Lennon",
"name": "John Lennon",
"born": "1940-10-09",
"spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}

```

Predtým ako som sa pustil do tvorby vlastného parseru som hľadal či neexistuje nejaký vhodný parser čo by mi veľmi uľahčilo prácu. Po nájdení viacerých som sa rozhodol pre jeden konkrétny parser a to **JSON-LD Streaming Parser** popísaný v ďalšej časti.

JSON-LD Streaming Parser

Pri testovaní tohto parseru samotného som narazil na problém cyklickej závislosti čo v praxi znamená, že dva alebo viac modulov (prípadne súborov) sú na sebe závislé. Tento problém sa mi podarilo vyriešiť pomocou použitia vyššie popísaného **Browserify** ktorý spojil všetok kód do jedného skriptu a tak tento problém pominul.

Po niekoľkých testoch na rôznych JSON-LD dokumentoch som zistil, že mi tento parser vyhovuje a preto som sa ho rozhodol použiť. Výsledok parsovania tak ako aj ďalšie použitie výstupu parsera je popísané nižšie v bode 3.

2. Vykresľovanie vlastného obsahu v prehliadači pomocou WebComponents

Web Components je suite troch technológií, ktorá umožňuje vytvárať a používať vlastné webové elementy. Základnou vlastnosťou je možnosť opätovného používania vlastných HTML elementov bez hrozby kolízií kódu. Toto je dosiahnuté za pomoci týchto technológií:

- **Custom elements:** JavaScript API, ktorá umožňuje definíciu vlastných elementov a ich vlastností pre ľubovoľné použitie.
- **Shadow DOM:** JavaScript API, ktorá umožňuje pripnutie zapúzdreného "shadow" DOM stromu k elementu. Tento strom je vyrenderovaný zvlášť od hlavného DOM stromu. Týmto spôsobom je zaručené, že element má privátne vlastnosti, a tak môže byť použitý bez rizika kolízie s ostatnými časťami dokumentu.
- **HTML template** Tagy `<template>` a `<slot>`, ktoré umožňujú písať predlohy, ktoré nie sú viditeľné na stránke. Tieto predlohy môžu byť použité viackrát ako základ štruktúry vlastných elementov.

Princíp vytvorenia vlastného elementu:

1. Vytvorenie triedy špecifikujúcej funkcionality web componenty s použitím ECMAScript 2015 syntaxe pre tvorbu tried.
2. Registrovanie nového elementu do registru elementov pomocou **CustomElementRegistry.define()**. Do tejto funkcie sa vloží meno elementu, trieda, ktorá špecifikuje funkcionality a meno elementu, od ktorého dedí vlastnosti (ak nejaký element je).
3. Pripnutie shadow DOM stromu k elementu pomocou **Element.attachShadow()**.

4. Ak je to potreba, definovanie HTML template pomocou `<template>` a `<slot>` HTML tagov.
5. Použitie custom elementu v HTML dokumente, tak isto ako klasického HTML elementu.

Konkrétny **custom element** používaný v mojej práci potom vyzerá nejako nasledovne:

```
<box-element id="art65#b92" style="position: absolute;
  top: 199px;
  left: 781px; width: 343px;
  height: 72px; background-color:#none; color:#000000; font-size:70.0px;
  font-weight: bold; font-family:DauphinPlain; font-style: 0.0;
  border-left:0px #;
  border-right:0px #;
  border-top:0px #;
  border-bottom:0px #;
  white-space: nowrap;">
  Lorem Ipsum
</box-element>
```

Tento element sa nachádza vo **for** cykle ktorý prechádza všetky boxy a pre každý vytvorí vlastný element s jeho hodnotami. Ukážka je pre demonštráciu skrátená pretože reálny element obsahuje viac ako dvadsať vlastností.

3. Vytvorenie boxov z rozparovaného JSON-LD a vykreslenie jednotlivých boxov artefaktu.

Výsledok parsovania z JSON-LD popisu stránky sú vždy RDF štvorice 2.3.1 (quads) popisujúce jednu konkrétnu informáciu či už o boxe alebo o celom artefakte.

Nasledujúci príklad v podstate prepisuje vetu Box 98 má farbu #000000a6 do RDF štvorice.

```
▼ Quad {termType: 'Quad', value: '', subject: NamedNode, predicate: NamedNode, object: Literal, ...}
  ► graph: NamedNode {termType: 'NamedNode', value: 'http://fitlayout.github.io/resource/art13'}
  ► object: Literal {termType: 'Literal', value: '#000000a6', language: '', datatype: NamedNode}
  ► predicate: NamedNode {termType: 'NamedNode', value: 'http://fitlayout.github.io/ontology/render.owl#color'}
  ► subject: NamedNode {termType: 'NamedNode', value: 'http://fitlayout.github.io/resource/art13#b98'}
  termType: "Quad"
  value: ""
  ► [[Prototype]]: Object
```

Obr. 5.1: Výsledok parsovania jednej vlastnosti

- **Graph** je IRI identifikátor artefaktu ku ktorému box patrí.
- **Object** je predmet. Čiže farba #000000a6.
- **Predicate** je prísudok čiže má farbu.
- **Subject** je podmet teda box 98.

Všetky takto získané informácie sú prejdené a podľa políčka subject pospájané dokopy do boxov. Každý tento box je potom vytvorený ako custom element zvaný **boxElement** tak ako je naznačené v ukážke vyššie. Každý box obsahuje informáciu o jeho absolútnej pozícii na stránke a vďaka tomuto môžem boxy vykreslovať na správnych súradniciach v rodičovskom elemente HTML dokumentu. Toto vkladanie elementu do rodičovského elementu sa robí pomocou JavaScript metódy **appendChild()**. Elementy sú podľa návrhu GUI 4.2 vložené do stredného panelu aplikácie. Tento Panel má pevnú výšku aj šírku závislú od veľkosti užívateľovej obrazovky.

4. Použitie Fetch na získanie JSON-LD reprezentácie stránky (artefaktu) z backendu.

Tak ako bolo spomenuté v návrhu JSON-LD reprezentácia stránky sa získava z backendu za pomoci JavaScript API **fetch**. Získanie dát prebieha zaslaním HTTP requestu typu **GET** na URL adresu konkrétneho artefaktu a následnou odpoveďou. Táto **URL** má tvar:

```
https://layout.fit.vutbr.cz/api/r/repo_id/artifact/item/r:art_id,
```

kde **repo_id** je identifikátor repozitára, z ktorého chceme artefakt získať a **art_id** je identifikátor artefaktu.

Príklad HTTP requestu pomocou **fetch()**:

```
fetch(url,{
  method: 'GET',
  headers: { 'Accept': 'application/ld+json' }
})
.then(function(body){
  return body.text();
}).then(function(data) {
  console.log(data);
  artParser.write(data);
  artParser.end();
}).catch(function(error){
  console.log(error);
})
```

V requeste sa nachádzajú informácie o tom o aký typ HTTP requestu ide a jedna hlavička, a to **Accept**, ktorá naznačuje, že bude prijatá odpoveď vo formáte JSON-LD. Po získaní odpovede bez chyby je telo odpovede vypísané do konzoly a posunuté do **JSON-LD Streaming Parseru** 5.3 v tomto prípade nazvaného **artParser**. Ak namiesto správnej odpovede príde chyba, tak bude chybová hláška vpísaná do konzoly a skript pokračuje ďalej.

5. Zobrazenie zoznamu všetkých artefaktov a boxov a vykreslenie zvoleného artefaktu.

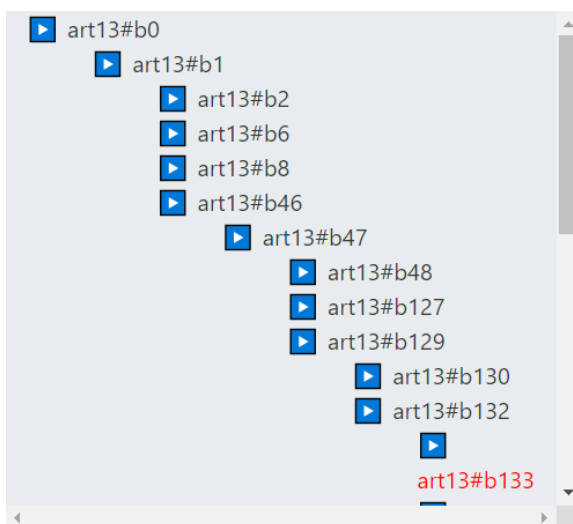
Tak ako jeden artefakt aj zoznam všetkých artefaktov v konkrétnom úložisku je z backendu získaný vo formáte JSON-LD. Po rozparovaní sú taktiež získané RDF štvorice obsahujúce informácie ako výška, šírka, zdrojové URL alebo názov konkrétneho artefaktu. Tieto informácie sú pospájané do jedného objektu. Po načítaní všetkých artefaktov je z pola objektov

obsahujúcich len identifikátor a rodičovský identifikátor strom pomocou ktorého je na ľavej strane obrazovky vytvorený zoznam. Po kliknutí na konkrétny artefakt je vykonaný bod 4. na získanie informácií o artefakte a následne bod 3. na vykreslenie artefaktu na obrazovku.

Princíp vytvárania stromu z pola objektov.

Jedna z väčších prekážok implementácie bola potreba vytvoriť strom (artefaktov a boxov) a následne z neho vytvoriť HTML zoznam (tag ``). Pri pokuse nájsť nejaký vhodný **NPM** balíček ktorý by takéto niečo dokázal som neuspel a preto som musel tento problém vyriešiť vlastnoručne pomocou JavaScriptu. Pri popisovaní tohto procesu budem používať slovo **element** ktoré označuje boxy alebo artefakty v závislosti na tom aký zoznam chceme vytvoriť. Tento proces začína funkciou **treeMaker** (**boxTreeMaker** pre boxy). V tejto funkcii je pole objektov obsahujúcich identifikátor elementu a identifikátor rodičovského elementu transformované na pole objektov kde objekt obsahuje identifikátor elementu a pole identifikátorov jeho detí. Tento krok bol potrebný z toho dôvodu, že som strom a následne aj zoznam potreboval vytvárať od koreňového elementu až po listy a samotné elementy prichádzajúce z backendu obsahovali len informáciu o rodičovskom liste, čo by znamenalo tvorbu stromu od listov po koreň.

Následne sú funkciou **forEach()** prejdené všetky tieto elementy. Každý element sa dostane do funkcie **mainRecursion** kde je rozhodnuté o tom či je rodič alebo nie. Podľa toho element pokračuje buď do funkcie **parent** alebo **child**. Funkcia **parent** vytvorí nový HTML element typu `` a nastaví v ňom všetky podstatné parametre ako napríklad identifikátor a trieda (ktorá zaručuje vizuálnu stavbu stromu). Ak ide o element typu artefakt tak je ďalej do `` pridané tlačidlo na zmazanie a segmentáciu (popísané nižšie v bode 8) a samotné `` je vložené do HTML zoznamu. Následne sú takisto funkciou **forEach()** prejdené všetky deti tohto rodičovského elementu a tak ako pri **mainRecursion** je rozhodnuté či ide o rodičovský element alebo element dieťaťa. Funkcia **child**, do ktorej sa dostanú len listové elementy, následne takisto vytvorí HTML element `` ale tentokrát s inou triedou ktorá počíta s tým, že žiadne iné elementy sa z tohto elementu už nebudú vetviť. Výsledný zoznam môžeme vidieť na obrázku 5.2



Obr. 5.2: Rozvetvený strom boxov

6. Interaktivita boxov

Po zobrazení všetkých podstatných informácií o artefaktoch a boxoch bolo treba pridať interaktivitu medzi jednotlivými časťami. Prvým bodom bolo zvýraznenie boxu na ktorý ukazuje myš na vykreslenej stránke, čo bolo dosiahnuté jednoduchým **:hover** selektorom na všetky boxy a ich ohraničením oranžovou farbou. Tento efekt je možné vidieť na obrázku 5.3

What is Lorem Ipsum?

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Obr. 5.3: Hover nad boxom

Ďalším krokom bolo zvýraznenie boxu po kliknutí na zobrazenej stránke alebo v strome boxov. Zvýraznenie na stránke je dosiahnuté jednoduchým pridávaním a odoberaním CSS triedy ktorá pridáva elementu červené ohraničenie podobne ako pri ukázaní na box myšou zobrazenom na obrázku 5.3 ale tento efekt na boxe zostáva až pokiaľ užívateľ neklikne na iný box. V strome boxov na pravej strane GUI sa táto akcia prejavuje tak, že strom sa rozbalí až po vybraný box a identifikátor zvoleného boxu sa farebne odliši ako je možné vidieť na obrázku 5.2 v predchádzajúcej časti. Ak je box už medzi rozbalenými boxami tak sa iba farebne odliši. Pri opätovnom kliknutí na rozbalený box sa všetci jeho potomkovia naspäť zabalia.

Poslednou časťou bolo vypísanie všetkých podstatných informácií o boxe do pravého dolného rohu GUI pod strom boxov. Toto zobrazenie je dosiahnuté pomocou jednoduchej HTML tabuľky ktorú je možné vidieť na obrázku 5.4. Informácie vpísané do tabuľky sú základné informácie o boxe ako napríklad identifikátor, šírka a výška ale aj špecifickejšie vlastnosti ako podčiarknutie textu, štýl textu a ohraničenie.

Attribute	Value
ID	art12#b107
Width	345px
Height	19px
PosX	233px
PosY	397px
Color	#000000
Font family	Open Sans
Font weight	bold
Font size	14.0
Font style	0.0
Background color	#none
Text	is simply dummy text of the printing and typesetting

Obr. 5.4: Informácie o boxe

7. Pridávanie, mazanie, segmentácia artefaktov.

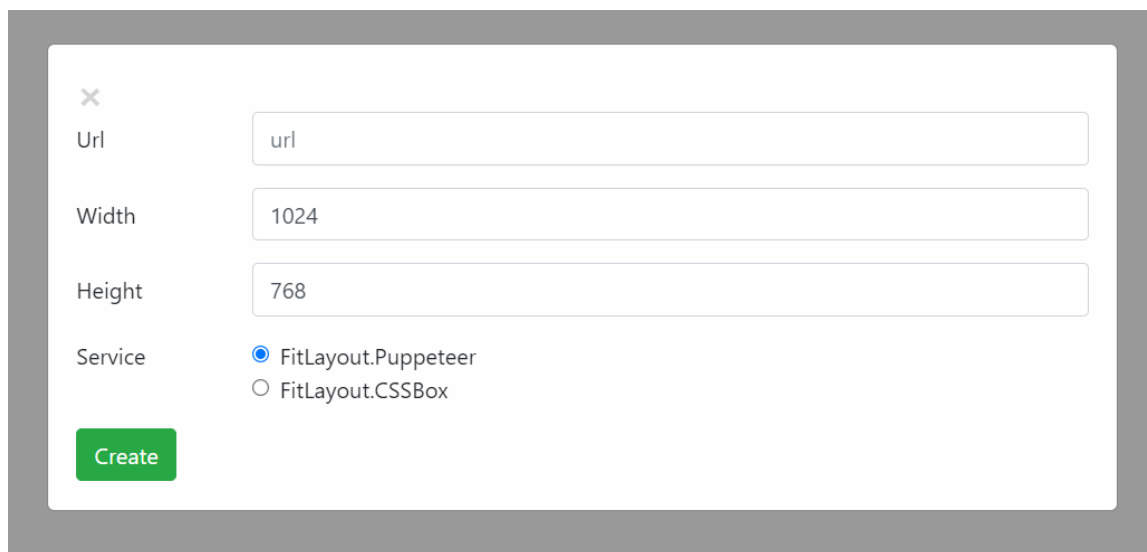
Pridávanie a odoberanie artefaktov do respektíve z repozitára je ďalšou podstatnou časťou FitLayout a tak je samozrejmé, že som túto časť pridal aj do webového rozhrania. Ďalšou dôležitou časťou je segmentácia artefaktu. Všetky tri akcie sa vykonávajú pomocou HTTP requestov s určitými parametrami na FitLayout backend z ktorého je získaný výsledok formou HTTP odpovede. V nasledujúcej časti bližšie popíšem tieto requesty.

Pridávanie

Pre pridanie nového artefaktu sú potrebné nasledujúce informácie:

- **URL:** Url adresa stránky.
- **Width:** Požadovaná šírka s maximálnou hodnotou 1920px a minimálnou 1024px.
- **Height:** Požadovaná výška s maximálnou hodnotou 1080px a minimálnou 768px.
- **Service:** Služba ktorá bude použitá na vyrenderovanie výsledku.

Tieto informácie sú získané od užívateľa pomocou HTML formuláru vo vyskakovacom modulárnom okne ktoré sa užívateľovi zobrazí po tom ako klikne na tlačidlo **Add artifact** v ľavom hornom rohu aplikácie.



Obr. 5.5: Formulár pre pridávanie nových artefaktov.

Po vyplnení týchto informácií a kliknutí na tlačidlo **Create** je užívateľovi oznámené, že artefakt bude čoskoro pridaný. Tieto informácie sú potom posielané HTTP requestom typu **POST** na špecifickú url adresu repozitára, obsahujúcu identifikátor repozitára a končiacu slovom **create**.

Príklad adresy:

```
https://layout.fit.vutbr.cz/api/r/ID/artifact/create
```

Po získaní kladnej odpovede od backendu je zoznam stránok znova načítaný už aj s novým artefaktom.

Odoberanie

Pre odoberanie alebo inak povedané mazanie artefaktov z repozitára je takisto použitý HTTP request ale v tomto prípade ide o request typu **DELETE**. Url adresa je identická ako pri získavaní artefaktu popísaného v sekcii 5.3 s tým rozdielom, že pri získavaní je typ requestu **GET**. Tento úkon odoberania je vykonaný kliknutím na červený krížik nachádzajúci sa pri vybranom artefakte v zozname artefaktov.

Segmentácia

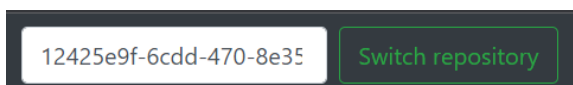
Pre hlbšiu segmentáciu artefaktu je potrebné zaslať **POST** request na rovnakú url ako pri pridávaní nového artefaktu s tým rozdielom, že v tele requestu je uvedený názov služby ktorá segmentáciu vykoná (defaultne FitLayout.VIPS) a IRI identifikátor artefaktu ktorý má byť spracovaný. Táto akcia sa podobne ako odoberanie vykonáva kliknutím na tlačítko **SEG** nachádzajúce sa pri tlačítku slúžiacom na odoberanie.



Obr. 5.6: Tlačidlá na odoberanie a segmentáciu artefaktu

8. Zmena repozitára.

Úložisk artefaktov popísaných v 3.2 je viacero, preto je vhodné aby rozhranie dokázalo pristupovať ku jednému konkrétnemu ktoré si užívateľ zvolí. Tak ako bolo popísané vyššie k repozitárom a ich artefaktom sa pristupuje pomocou URL v ktorej je uvedený aj identifikátor repozitára. Z tohto dôvodu je prepínanie medzi repozitármi pomerne jednoduché keďže stačí od užívateľa zistiť identifikátor repozitára s ktorým chce pracovať. Pri prvom spustení aplikácie je identifikátor repozitára, tak ako aj URL adresa REST API, načítaný z konfiguračného súboru **config.js**. Aktuálne používaný repozitár (konkrétne jeho identifikátor) je vždy zobrazený v strede horného navigačného panela. Pre zmenu identifikátora a teda aj repozitára jednoducho stačí prepísať túto hodnotu a stlačiť tlačidlo **Switch repository** a obsah požadovaného repozitára sa načíta. Ak sa načítanie nepodarí z dôvodu serverovej chyby alebo chybného identifikátora tak je užívateľovi zobrazená hláška a aplikácia sa pokúsi načítať pôvodný repozitár používaný pred pokusom o jeho zmenu.



Obr. 5.7: Formulár pre zmenu repozitára

9. Zasielanie SPARQL požiadavku.

Praktickou funkciou pri analýze dokumentov je možnosť vyhľadať časti podľa určitej vlastnosti. V prípade mojej aplikácie sa takáto funkcia rieši pomocou zasielania SPARQL požiadavku na server ktorý vráti výsledné boxy.

SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) je štandardný query jazyk podobný SQL používaný pre RDF databázy aké využíva aj FitLayout 3.2. SPARQL štandard je dizajnovaný a udržiavaný W3C konzorciom tak aby sa používateľ mohol zamerať na to čo potrebuje a nie na to ako databáza funguje.[17]

Query písaná v tomto jazyku môže obsahovať nasledujúce typy:

- **SELECT:** Podobne ako v SQL vracia výsledok hľadania.
- **CONSTRUCT:** Vytvára nový RDF graf na základe query výsledku.
- **INSERT:** Podobne ako v SQL vkladá trojice do grafu.
- **DELETE:** Rovnako ako v SQL maže trojice z grafu.
- **ASK:** Vracia áno alebo nie na základe toho či query má výsledok.
- **DESCRIBE:** Vracia RDF dáta o zdroji. Užitočné ak užívateľ nepozná štruktúru RDF dát v zdroji.[17]

Pre zaslanie takéhoto požiadavku musí užívateľ kliknúť na tlačidlo **Search** v pravej hornej časti aplikácie. Po jeho kliknutí sa mu zobrazí modulárne okno s formulárom do ktorého musí vpísať query a následne požiadavok odoslať.

Samotná query potom vyzerá nasledovne:

```
PREFIX fl: <http://fitlayout.github.io/ontology/fitlayout.owl#>
PREFIX r: <http://fitlayout.github.io/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX box: <http://fitlayout.github.io/ontology/render.owl#>
PREFIX segm: <http://fitlayout.github.io/ontology/segmentation.owl#>

SELECT ?iri WHERE {
    ?iri QUERY_PARAMETER
}

ORDER BY ?time
```

Kde **QUERY_PARAMETER** predstavuje samotnú vlastnosť ktorý musí užívateľ doplniť. Napríklad **QUERY_PARAMETER** s hodnotou: **box:fontWeight "1.0"xsd:float** nájde všetky boxy ktorých **fontWeight** je 1.0 čo v praxi znamená „bold“.

10. Vizuálna úprava

Posledným bodom implementácie boli štylistické úpravy. Na dosiahnutie použitého vzhľadu bolo použité CSS a framework Bootstrap. Podľa návrhu GUI je celá aplikácia rozdelená na navigačný panel, telo rozdelené na 3 časti a päť stránky. Výsledný vzhľad je možné vidieť na obrázku 5.8 pod popisom jednotlivých častí.

Zaujímavou časťou je **navigačný panel** nachádzajúci sa na vrchnej časti aplikácie. Tento panel je vytvorený pomocou Bootstrap triedy **navbar**. Na ňom sa nachádzajú tri tlačidlá a jedna textová kolónka. Na ľavej strane sa nachádza tlačidlo **Add artifact** ktorého funkcionality je popísaná v bode 7. V strednej časti panela sa nachádza názov aplikácie

FitLayout a vedľa neho textová kolónka a tlačidlo pre zobrazenie a zmenu identifikátora aktuálne používaného repozitára. Popis funkcionality tejto časti navigačného panela je popísaný v bode 8. Nakoniec je na pravej strane umiestnené tlačidlo **Search** ktoré slúži na vyhľadanie boxov ako bolo popísane v bode 9. Ak sú nájdené nejaké boxy tak sa pri tomto tlačidle zobrazí ešte jedno tlačidlo **Remove search** ktoré slúži na zrušenie vyhľadania. V prípade, že nie sú vyhľadané žiadne boxy toto tlačidlo na navigačnom paneli nie je.

Hlavné telo aplikácie bolo vytvorené pomocou jednou z najväčších predností Bootstrapu a to jeho grid systémom. Celé telo sa nachádza v HTML tagu `<div>` s triedov **row** teda riadok. Tak ako bolo popísané v časti 2.1.8 tento riadok môže byť rozdelený až na 12 stĺpcov v ľubovlnom pomere. Pre moje potreby som riadok rozdelil do troch stĺpcov a po experimentovaní s optimálnou šírkou som prišiel na to, že najlepšie bude rozdeliť aplikáciu na pomer 2.5:7:2.5. To znamená, že ľavá strana obsahujúca zoznam artefaktov zaberá približne 20% stránky, stredná časť na ktorej sa zobrazuje renderovaná stránka zaberá približne 60% stránky a pravá časť patriaca boxom zaberá takisto približne 20% stránky.

Poslednou časťou je footer teda päta stránky. Táto časť je vytvorená pomocou Bootstrap triedy **footer** a obsahuje rok vypracovania tejto bakalárskej práce a odkaz na dokumentáciu nástroja **FitLayout**.



Obr. 5.8: Výsledný vzhľad aplikácie

Kapitola 6

Testovanie

Táto kapitola sa zaoberá testovaním finálnej podoby aplikácie. Samotná aplikácia bola testovaná testovacou sadou počas celej doby vývoja. Vďaka tomuto som počas finálnych testov som narazil len na pár malých chýb ktoré sa mi podarilo včas opraviť. V nasledujúcej časti sú popísané výsledky funkčných testov aplikácie po ukončení jej vývoja. Keďže išlo o jednoduchú testovaciu sadu obsahujúcu manuálne testy, ich popis a výsledky sú uvedené v prehľadných tabuľkách rozdelených podľa toho, o ktorú časť aplikácie išlo.

Podmienky na testovanie: Aplikácia bola spustená na vývojovom serveri budo tak ako je popísané v manuále. Testy boli vykonané v prehliadači Google Chrome verzia 100.0.4896.127 a Mozilla Firefox verzia 87.0.

Testované artefakty a repozitáre:

- Repozitár s id **12425e9f-6cdd-4700-8e35-6a4c6504a258**: Art65, Art66, Art67
- Repozitár s id **7d1a35b2-22dd-451f-9c6b-476c821b8c9b**: Art1, Art10, Art396

Testovaná časť/ konkrétny test	Požadované správanie	Výsledne správanie
<u>Konfigurácia</u>		
Nastavenie existujúceho identifikátora a existujúcej REST API url v konfiguračnom súbore.	Aplikácia načíta repozitár z REST API podľa identifikátora a url.	Aplikácia načítala repozitár so správnym identifikátorom z požadovaného REST API.
Nastavenie neexistujúceho identifikátora v konfiguračnom súbore.	Oznam o nesprávnom identifikátore.	Aplikácia pri jej spustení vypísala na obrazovku informáciu o nesprávnom identifikátore.
Nastavenie neexistujúcej REST API url v konfiguračnom súbore.	Oznam o nesprávnej REST API url.	Aplikácia pri jej spustení vypísala na obrazovku informáciu o nesprávnom REST API url.

Obr. 6.1: Testovanie konfigurácie.

Testovaná časť/ konkrétny test	Požadované správanie	Výsledne správanie
<u>Artefakty</u>		
Pridanie nového artefaktu s platnou url.	Nový artefakt bude pridaný do zoznamu artefaktov.	Po pridaní artefaktu bolo na obrazovke oznámené, že artefakt bude pridaný a repozitár aj s novým artefaktom bol do pár sekúnd načítaný.
Pridanie nového artefaktu s chybnou alebo žiadnou url.	Oznam o nesprávnej url.	Na obrazovke sa objavil oznam s informáciou, že zadaná url adresa je nesprávna.
Vymazanie artefaktu.	Artefakt bude odstránený zo zoznamu.	Po kliknutí na tlačidlo vymazania artefaktu bol artefakt vymazaný zo zoznamu a aj z repozitára.
Segmentácia artefaktu.	Nový artefakt bude pridaný ako potomok segmentovaného.	Po kliknutí na tlačidlo segmentácie bol po pár sekundách pridaný nový artefakt ako potomok toho segmentovaného.
Zvolenie artefaktu	Artefakt bude zobrazený v strednom paneli a zvýraznený v zozname artefaktov	Artefakt sa zvýraznil v zozname, jeho potomkovia boli zobrazený pod ním a po načítaní bola v strede aplikácie vykreslená renderovaná stránka.

Obr. 6.2: Testovanie práce s artefaktmi.

Testovaná časť/ konkrétny test	Požadované správanie	Výsledne správanie
<u>Boxy</u>		
Prejdenie myšou ponad box a následné kliknutie vo vykreslenej stránke.	Box bude pri prejení ponad ohraničený farbou a po následnom kliknutí zvýraznený až do kliknutia na iný box.	Pri prechode myšou ponad boxy boli zvýrazňované oranžovou farbou. Po kliknutí na jeden bol zvýraznení červenou farbou a bolo tak až dovtedy kým nebolo kliknuté na iný box.
Správne zadaná query s 0 výsledkami.	Oznam o výsledku.	Na obrazovke sa objavil oznam s informáciou, že neexistuje žiadny box s požadovanou vlastnosťou v tomto repozitári.
Nesprávne zadaná query.	Oznam o nesprávnej query.	Na obrazovke sa objavil oznam s informáciou, že zadaná query je chybná.
Správne zadaná query s výsledkami.	Zvýraznenie boxov a artefaktov obsahujúcich boxy s požadovanou vlastnosťou.	Boxy zodpovedajúce hľadaniu boli zvýraznené zelenou farbou v zozname a aj vo vykreslenej stránke. Takisto artefakty v ktorých sa takéto boxy nachádzali boli zvýraznené v zozname.

Obr. 6.3: Testovanie práce s boxami.

Testovaná časť/ konkrétny test	Požadované správanie	Výsledne správanie
<u>Repozitár</u>		
Zmena repozitára na existujúci identifikátor.	Nový repozitár bude načítaný.	Nový repozitár bol načítaný správne.
Zmena repozitára na neexistujúci identifikátor.	Oznam o nesprávnom identifikátore.	Po zmene bol na obrazovke zobrazený oznam o nesprávnom identifikátore repozitára. Bol načítaný predchádzajúci funkčný repozitár.

Obr. 6.4: Testovanie práce s repozitárom.

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať klientskú časť webovej aplikácie na analýzu webových dokumentov. Návrh a funkcionality vychádzali z existujúcej desktopovej aplikácie FitLayout. Výsledná aplikácia umožní užívateľovi analyzovať webové dokumenty podobne ako v desktopovej aplikácii ale s tým rozdielom, že sa k aplikácii dostane pohodlnejšie pomocou svojho webového prehliadača.

Návrhu a implementácií predchádzalo naštudovanie problematiky tvorby webových aplikácií ktoré je popísané v kapitole 2 a naštudovanie a oboznámenie sa s nástrojom FitLayout ktoré sa nachádza v kapitole 3. V kapitole 2 boli popísané rôzne technológie na tvorbu klientskej a serverovej časti ako aj popis niektorých ďalších, pre túto problematiku podstatných technológií ktoré sa neradia výhradne k frontendu ani backendu. V kapitole 3 sa nachádza popis a priblíženie frameworku FitLayout a jeho nástrojov.

Návrh aplikácie je čitateľovi priblížený v kapitole 4. V tejto časti sa čitateľ oboznámil s požadovanou funkcionalitou aplikácie, architektúrou, existujúcimi riešeniami a nakoniec s návrhom grafického užívateľského prostredia. Pri prvotnom návrhu aplikácie som sa pohrával s myšlienkou využiť niektorý s JavaScript frameworkov ale od tohto nápadu som neskôr upustil. Tento krok považujem za vhodný keďže na takúto komplexnosť aplikácie samotný JavaScript úplne stačí.

Po návrhu nasledovala samotná implementácia ktorá je popísaná v kapitole 5. Počas celého vývoja som používal nástroj GitHub, ktorý pomohol pri organizácii a umožňoval vedúcemu práce sledovať môj postup. Pre zefektívnenie a uľahčenie implementácie bol použitý framework Browserify a k nemu patriaci vývojový server budo. Základ aplikácie bol implementovaný v jazyku HTML jeho funkčnosť v jazyku JavaScript a k nemu patriacim modulom a vzhľad bol docielený pomocou CSS a frameworku Bootstrap. Pri implementácii som narazil na pár väčších prekážok ako napríklad spôsob vykresľovania stromu boxov a artefaktov alebo problémy s cyklickou závislosťou pri moduloch. Všetky tieto problémy sa mi počas vývoja podarilo vyriešiť a tak aplikácia dosahuje požadovanú funkčnosť.

Vykresľovanie stránok som porovnal s reálnymi stránkami a ich vzhľad sa rámcovo podobal. Najväčšia nezhoda s väčšinou stránok bola vo fontoch keďže nie všetky fonty HTML pozná a bolo by ich treba pred použitím získať.

Celková funkčnosť aplikácie bola nakoniec otestovaná v kapitole 6 pomocou sady manuálnych testov. Pri finálnom testovaní som narazil na pár menších problémov ktoré som tak mohol vďaka vhodným testom včas vyriešiť.

Literatúra

- [1] BROOKS, D. R. *Guide to HTML, JavaScript and PHP: For Scientists and Engineers*. 1. vyd. Springer-Verlag London, 2011. ISBN 978-0-85729-448-7.
- [2] BURGET, R. *Kaskádové styly (CSS), Tvorba webových stránek - Přednášky* [online]. 2015 [cit. 2022-03-06]. Dostupné z: https://www.fit.vutbr.cz/study/courses/ITW/private/prednasky/itw_p03.pdf.
- [3] BURGET, R. *Značkovací jazyky: HTML, Tvorba webových stránek - Přednášky* [online]. 2015 [cit. 2022-03-06]. Dostupné z: https://www.fit.vutbr.cz/study/courses/ITW/private/prednasky/itw_p02.pdf.
- [4] BURGET, R. *FitLayout/2 - Web Page Analysis Framework* [online]. 2021 [cit. 2022-03-17]. Dostupné z: <https://github.com/FitLayout/FitLayout/wiki>.
- [5] BURGET, R. *FitLayout/2 Basic Concepts* [online]. 2021 [cit. 2022-03-17]. Dostupné z: <https://github.com/FitLayout/FitLayout/wiki/Basic-Concepts>.
- [6] DATA, R. *What is Bootstrap?* [online]. 2021 [cit. 2022-03-08]. Dostupné z: https://www.w3schools.com/whatis/whatis_bootstrap.asp.
- [7] DESHPANDE, C. *The Best Guide to Know What Is React?* [online]. 2021 [cit. 2022-03-09]. Dostupné z: <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs>.
- [8] DESHPANDE, C. *What is Angular?: Architecture, Features, and Advantages* [online]. 2022 [cit. 2022-03-09]. Dostupné z: <https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular>.
- [9] DWIGHT, R. *PHP: Learn PHP in 24 Hours or Less - A Beginner's Guide To Learning PHP Programming Now*. 1. vyd. CreateSpace Independent Publishing Platform, 2016. ISBN 978-1530904389.
- [10] LEMAY, L. a PERKINS, L. C. *Teach Yourself JAVA in 21 Days*. 1. vyd. Sams.net Publishing, 1996. ISBN 1-57521-030-4.
- [11] LINDLEY, C. *DOM Enlightenment*. 1. vyd. O'Reilly, 2013. ISBN 978-1-449-34284-5.
- [12] MACRAES, C. *Vue.js: Up and Running*. 1. vyd. O'Reilly Media, Inc., 2018. ISBN 9781491997246.
- [13] NODE.JS. *Introduction to Node.js* [online]. [cit. 2022-03-15]. Dostupné z: <https://nodejs.dev/learn/introduction-to-nodejs>.

- [14] ODELL, D. *Pro JavaScript Development: Coding, Capabilities, and Tooling. Expert's voice in Web development*. 1. vyd. Apress, 2014. ISBN 978-1-4302-6268-8.
- [15] PYTHON. *What is Python? Executive Summary* [online]. [cit. 2022-03-11]. Dostupné z: <https://www.python.org/doc/essays/blurb/p>.
- [16] SPURLOCK, J. *Bootstrap*. 1. vyd. O'Reilly Media, Inc., 2013. ISBN 978-1-449-34391-0.
- [17] W3C. *SPARQL Query Language for RDF* [online]. 2008 [cit. 2022-04-05]. Dostupné z: <https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>.
- [18] W3C. *JSON-LD 1.1 A JSON-based Serialization for Linked Data* [online]. 2020 [cit. 2022-03-28]. Dostupné z: <https://www.w3.org/TR/json-ld11/>.

Príloha A

Obsah pamäťového média

- **xsevci64.pdf** - PDF súbor práce
- **xsevci64.zip** - zdrojové súbory teoretickej časti práce
- **src.zip** - zdrojové súbory praktickej časti práce
 - **src** - repozitár zdrojových súborov
 - **ReadMe.md** - manuál k praktickej časti práce

Príloha B

Manuál k praktickej časti

B.1 Stiahnutie potrebných balíkov

Pred začiatkom používania aplikácie je potreba stiahnuť z npm potrebné balíky. Nasledujúci príkaz stiahne všetky potrebné balíky:

```
npm install
```

B.2 Konfigurácia

V súbore **config.js** nájdete konfiguračné hodnoty pre **URL** použitého REST API a **id** použitého repozitára. Ak si prajete tieto hodnoty zmeniť jednoducho pred spustením aplikácie vymeňte prednastavené hodnoty za vaše hodnoty.

```
repo_id: "12425e9f-6cdd-4700-8e35-6a4c6504a258"  
url: "https://layout.fit.vutbr.cz/api"
```

B.3 Spustenie

Pre spustenie aplikácie je potreba napísať príkaz:

```
npm start
```

Tento príkaz spustí na adrese <http://192.168.0.105:9966/> server budo s aplikáciu v predvo-
lenom prehliadači.

B.4 Použitie

Pridávanie artefaktov

Pre pridanie artefaktov je potreba v ľavej hornej časti aplikácie kliknúť na tlačidlo **Add artifact**. Po kliknutí na toto tlačidlo sa zobrazí okno s formulárom pre pridanie artefaktu do ktorého je potreba vložiť potrebné informácie a to URL, šírku, výšku a službu ktorou bude stránka segmentovaná.

Mazanie artefaktov

Pre vymazanie artefaktu z repozitára stačí pri názve artefaktu kliknúť na červené tlačidlo **X**. Po kliknutí na toto tlačidlo je artefakt vymazaný.

Segmentácia artefaktov

Pre hlbšiu segmentáciu artefaktu stačí kliknúť na tlačidlo SEG nachádzajúce sa pri názve artefaktu.

Zmena repozitára

Pre zmenu repozitára je potreba v hornej časti aplikácie zmeniť ID repozitára a kliknúť na tlačidlo **Switch repository**. Po kliknutí bude načítaný nový repozitár.

Vyhľadanie boxov

Pre vyhľadanie boxov je potreba kliknúť na tlačidlo **Search** nachádzajúce sa v pravom hornom rohu aplikácie. Po jeho kliknutí je zobrazené okno s formulárom do ktorého je potreba vpísať požadovaný query príkaz pre vyhľadanie. Pre zrušenia vyhľadania je potrebné kliknúť na tlačidlo **Remove search** ktoré sa zobrazí pri tlačidle **Search** po úspešnom vyhľadaní.