VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

KOMPLEXNÍ VALIDÁTOR PRO WEBOVÉ STRÁNKY

COMPLEX VALIDATOR FOR WEB PAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JOZEF HORVÁT**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ VOLF**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Horvát Jozef**
Program: Informační technologie
Název: **Komplexní validátor pro webové stránky**
Complex Validator for Web Pages
Kategorie: Web

Zadání:

1. Seznamte se s validátory webových stránek (HTML, JS, CSS) a jejich API.
2. Seznamte se s knihovnami pro zpracování vzdálených stránek a pro práci s DOM stromem.
3. Navrhněte webovou aplikaci, která pro zadanou nebo nahranou stránku umožní zvolit validátory (příp. také jejich parametry), kterými bude vstupní stránka zkontrolována skrze API validátorů. Navrhněte způsob, jakým by mohly být u chyb nabízeny možné varianty úpravy pro jejich rychlou aplikaci (po odsouhlasení varianty uživatelem), uživatel by měl mít rovněž možnost ruční editace. Po ukončení validace by měl mít možnost upravené vstupy stáhnout či stáhnout diff pro patch. Navrhněte uživatelské rozhraní vhodně tak, aby chyby byly uživateli zobrazeny přívětivě.
4. Navrženou aplikaci implementujte, volbu použitých technologií zdůvodněte. Po dohodě s vedoucím vyberte chyby, které budou na výstupu nabízeny k přímé opravě.
5. Ověřte funkčnost aplikace na vhodně zvolených webových stránkách.
6. Zhodnoťte dosažené výsledky, diskutujte další možné rozšiřování aplikace.

Literatura:

- Ward, J.: Instant PHP Web Scraping, Packt Publishing, 2013, ISBN: 9781782164760
- Mekhatria, M.: Learning Web Scraping with JavaScript, 2018, ISBN: 9781789611311
- Kosek, J.: PHP a XML, Grada Publishing, 2009, ISBN: 978-80-247-1116-4

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání (u bodu 3 není pro účely zápočtu za zimní semestr vyžadováno finální / detailní vypracování).

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

Vedoucí práce: **Volf Tomáš, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 16. října 2019

Abstrakt

Táto práca sa zaoberá vytvorením komplexného validátora pre webové stránky používajúce HTML, CSS a JavaScript. Aplikácia je implementovaná pomocou frameworku Angular. Okrem validácie aplikácia poskytuje návrhy na opravu chýb.

Abstract

This thesis deals with the creation of a complex validator for web pages using HTML, CSS and JavaScript. The application is implemented using the Angular framework. In addition to validation, the application provides suggestions for error correction.

Kľúčové slová

validátor, web, HTML, CSS, JavaScript, webové stránky, Angular, oprava chýb

Keywords

validator, web, HTML, CSS, JavaScript, web pages, Angular, error correction

Citácia

HORVÁT, Jozef. *Komplexní validátor pro webové stránky.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Volf

Komplexní validátor pro webové stránky

Prehlásenie

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

.....
Jozef Horvát
25. mája 2020

Poďakovanie

Rád by som poďakoval pánovi Ing. Tomášovi Volfovi, za množstvo cenných rád a pripomienok, ktoré mi poskytol pri spracovaní tejto bakalárskej práce.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je vytvorenie webovej aplikácie umožňujúcej komplexnú validáciu webových stránok. Aplikácia prehľadne zobrazí chyby užívateľovi s návrhom opráv vybraných chýb, pomocou ktorých si tak môže rýchlo opraviť svoj kód. Aplikácia umožní užívateľovi rovno opraviť chyby v kóde pomocou editora. Užívateľ si môže vybrať druh validácie a aj jej rôzne parametre.

Aktuálne existuje veľké množstvo validátorov webových technológií. Väčšinou tieto validátory ponúkajú svoje služby samostatne, čo neumožňuje jednoduchú validáciu celej webovej stránky. Po validácií si musí užívateľ zvalidovaný kód upraviť u seba, čo nemusí byť pre užívateľa príjemné. Tieto validátory neponúkajú rýchle opravy chýb.

Moja práca bude riešiť práve tieto problémy. Aplikácia bude bežať v internetových prehliadačoch. Užívateľ si tak bude môcť na jednom mieste nechať skontrolovať svoju webovú stránku, bude si môcť rýchlo opraviť chyby a stiahnuť upravenú verziu.

Kapitola 2 popisuje programovacie jazyky, ktoré sa používajú pri vývoji webových stránok, a ktoré by mala byť moja aplikácia schopná validovať. V kapitole 3 sa zaoberám spôsobmi ako pristúpiť k vzdialeným webovým stránkam a následne získať webové zdroje. V tejto kapitole som definoval aplikačné rozhranie, opísal protokol HTTP a jeho metódy, objektový model dokumentu, potrebný pri spracovaní webových zdrojov a nástroje, ktoré sa dajú použiť pre prístup k týmto zdrojom spolu s obmedzeniami prístupu k ním. Kapitola 4 pojednáva o rôznych existujúcich validátoroch a opisuje ich aplikačné rozhrania. V kapitole 5 rozoberám návrh aplikácie, ktorý sa skladá z výberu nástrojov pre implementáciu frontendu a backendu, popisu jej funkcionality, návrhu užívateľského prostredia, návrhu jej vnútornej architektúry a návrhu zoznamu chýb, ktoré by aplikácia mala byť schopná opraviť. V kapitole 6 opisujem niektoré podrobnosti implementácie aplikácie, ako spôsob využívania validátorov, implementácia opravovania chýb, práce so súbormi alebo užívateľského rozhrania. V kapitole 7 testujem funkčnosť aplikácie na vhodných príkladoch. V kapitole 8 zhŕňam výsledky práce a ponúkam návrhy ďalšieho smerovania vývoja aplikácie.

Kapitola 2

Validované jazyky

V tejto kapitole popisujem programovacie jazyky, ktoré sa používajú pri tvorbe webových stránok. Tieto jazyky by mala byť schopná validovať moja aplikácia.

2.1 Jazyk HTML

Jazyk HTML je štandardný značkovací jazyk. Skratka HTML znamená Hyper Text Markup Language. Informácie o jazyku HTML som čerpal z tohto zdroja [30].

Tento jazyk slúži pre vytváranie štruktúry webových stránok. Jazyk HTML pozostáva z rady elementov, ktoré definujú spôsob, akým je potrebné zobraziť obsah webovej stránky. Elementy sú reprezentované pomocou značiek. Značky označujú jednotlivé časti dokumentu ako napríklad záhlavie, paragraf, tabuľka a ďalšie.

Element v jazyku HTML väčšinou pozostáva z dvoch častí: z otváracej a uzatváracej značky. Medzi nimi sa nachádza obsah. Elementy v jazyku HTML je možné zanárať do seba. Značky v jazyku HTML nie sú citlivé na veľké alebo malé písmená. Dobrým zvykom je používanie malých písmen.

Dokumenty napísané v jazyku HTML majú na začiatku definovaný typ dokumentu. Telo dokumentu sa nachádza medzi značkami <html></html>. Za značkou <head> nasleduje hlavička dokumentu. Programátor tu môže uviesť napríklad názov stránky, autora, skripty, štýly a ďalšie. To čo užívateľ vidí v prehliadači je označené začiatočnou a koncovou značkou <body>.

Značky jazyka HTML môžeme rozdeliť do niekoľkých skupín:

- základné značky,
- formátovacie značky,
- značky definujúce formuláre a vstupy,
- značky definujúce multimédia,
- značky definujúce zoznamy,
- značky definujúce spojenia a hypertextové odkazy,
- značky pre deklaráciu tabuliek,
- značky definujúce metainformácie,

- značky upravujúce štýl,
- značky umožňujúce pridať kód.

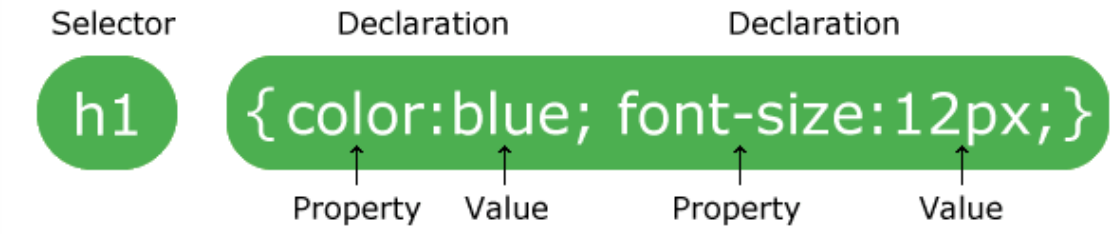
Elementy v jazyku HTML môžu obsahovať atribúty, ktoré poskytujú dodatočné informácia o elemente v jazyku HTML. Atribúty sú špecifikované v otváracej značke. Atribúty väčšinou obsahujú názov atribútu a hodnotu atribútu.

2.2 Jazyk CSS

Jazyk CSS slúži pre popis vzhľadu webovej stránky. Skratka CSS znamená Cascading Style Sheets. Informácie o jazyku CSS som čerpal informácie z týchto zdrojov [29] [26] a [25].

Tento jazyk nám umožňuje meniť fonty, farby, umiestnenie jednotlivých elementov a ďalšie. Jazyk CSS taktiež umožňuje prispôsobiť vzhľad rôznym druhom zariadení, napríklad pre veľké obrazovky, mobilné zariadenia, tlačiarne a ďalšie. Jazyk CSS je nezávislý od jazyka HTML a môže byť použitý s rôznymi značkovacími jazykmi postavených na XML. Pre lepšie udržiavanie webových stránok sa zvykne oddeľovať jazyk HTML od od jazyka CSS. Umožňuje to následne ľahšie zdieľať štýly medzi webovými stránkami a prispôsobiť webové stránky rôznym prostrediam.

V jazyku CSS je vzhľad definovaný pomocou pravidiel. Príklad pravidla sa nachádza na obrázku 2.1. Pravidlá sa skladajú zo selektora a bloku deklarácií. Selektor určuje, ktorému elementu jazyka HTML chceme prispôsobiť štýl. Blok deklarácií obsahuje jednu alebo viacero deklarácií, oddelených bodkočiarkou. Každá deklarácia pozostáva z názvu vlastnosti a hodnoty vlastnosti, ktoré sú oddelené dvojbodkou. Bloky deklarácií sú uzavreté zloženými zátvorkami.



Obr. 2.1: Príklad pravidla v jazyku CSS [26].

Selektory môžeme rozdeliť do piatich kategórií:

- jednoduché selektory - vyberajú elementy na základe mena, triedy alebo identifikátora,
- kombinačné selektory - vyberajú elementy na základe špecifických vzťahov medzi nimi,
- selektory pseudo-tried - vyberajú elementy na základe určitého stavu,
- selektory pseudo-elementov - vyberajú a prispôsobujú štýl časti elementu,
- selektory atribútov - vyberajú elementy na základe atribútov alebo hodnôt atribútov.

2.3 Jazyk JavaScript

Jazyk JavaScript je interpretovaný alebo just-in-time kompilovaný programovací jazyk, kde sú funkcie na prvom mieste. Informácie o jazyku JavaScript som čerpal z [11].

JavaSript je známy ako skriptovací jazyk využívajúci sa pre webové stránky. Tento jazyk sa okrem prehliadača využíva aj v iných prostrediach, ako napríklad Node.js, Apache CouchDB a Adobe Acrobat. Jazyk JavaScript je prototypovo založený, multi-paradigmový, jedno vláknový, dynamický jazyk, podporujúci objektovo orientovaný, imperatívny a deklaratívny štýl programovania.

Štandard tohto jazyka sa nazýva ECMASript. Od roku 2012 všetky moderné internetové prehliadače plne podporujú ECMASript 5.1 a tie staršie podporujú najmenej ECMAScript 3. Organizácia ECMA International v roku 2015 zverejnila šiestu verziu ECMAScript, označovaná ako ECMAScript 6 alebo ES6. Odvtedy štandardy ECMAScript vychádzajú každý rok.

Je potrebné si dávať pozor na zamieňanie jazyka JavaScript s jazykom Java. „Java“ aj „JavaScript“ sú ochranné známky alebo registrované ochranné známky spoločnosti Oracle v USA a ďalších krajinách. Tieto jazyky sú od seba veľmi odlišné. Jazyky majú odlišnú syntax, sémantiku a aj použitie.

Kapitola 3

Vzdialené spracovanie webových stránok

V tejto kapitole píšem o aplikačnom prostredí, hypertextovom prenosovom protokole a jeho metódach. V kapitole ďalej opisujem objektový model dokumentu a jeho využitie v jazyku JavaSript a TypeScript. Na konci kapitoly sa pozerám na možnosti získania webových zdrojov a problematiku CORS.

3.1 Aplikačné rozhranie

Aplikačné rozhranie je zbierka definícií a protokolov. Aplikačné rozhranie sa označuje taktiež skratkou API (Application interface). API umožňuje produktom a službám komunikovať s inými produktami a službami bez znalosti ich implementácie. Programovanie je tak rýchlejšie, jednoduchšie a hrozí menší výskyt chýb [19].

3.2 Hypertextový prenosový protokol

Hypertextový prenosový protokol je webovým protokolom aplikačnej vrstvy. Tento protokol je často označovaný skratkou HTTP (HyperText Transfer Protocol). HTTP je definovaný v RFC1945 a RFC2616. Tieto a aj následujúce informácie boli čerpané z knihy [17], ak nie je uvedené inak.

Protokol je implementovaný v klientskom a serverovom programe. Tieto programy sa spúšťajú na rozličných koncových systémoch. Komunikujú medzi sebou pomocou HTTP správ. HTTP definuje štruktúry týchto správ a spôsob komunikácie. HTTP využíva pre komunikáciu na transportnej vrstve protokol TCP.

Protokol HTTP je bezstavový protokol. Protokol neuchováva na serveri žiadne informácie o klientovi. Ak klient znova požiada o rovnaký objekt, server mu ho znova zašle.

3.2.1 Metódy protokolu HTTP

Hypertextový prenosový protokol definuje niekoľko metód. Niektoré najznámejšie metódy sú:

- GET - slúži na získanie dát identifikovaných URI, používa sa aj na vyhľadávanie,
- HEAD - je podobná ako metóda GET, ale slúži na získanie hlavičky správy,

- POST - slúži na vytváranie nových objektov,
- PUT - slúži na upravovanie celých objektov,
- PATCH - slúži na upravovanie jednotlivých častí objektov,
- DELETE - slúži na mazanie objektov,
- OPTIONS - vracia podporované HTTP metódy, používa sa na otestovanie funkčnosti servera [1].

3.2.2 HTTP cookies

Na uchovávanie aspoň nejakého stavu sa používajú HTTP cookies. Je to malý obnos dát, ktorý zasiela klientovi. Prehliadač si tieto dáta môže uložiť a pri ďalšej požiadavke poslať na server. HTTP cookies sa môžu využiť napríklad na zistenie, či je užívateľ prihlásený alebo pre zapamätanie si obsahu nákupného košíka [9].

3.3 Objektový model dokumentu

Objektový model dokumentu alebo tiež nazývaný DOM (Document object model) je objektová reprezentácia HTML alebo XML dokumentu. Informácie o DOM som čerpal z MDN webovej dokumentácie [10]. Príklad DOM je zobrazený na obrázku 3.1.

Webová stránka je dokument, ktorý je možné zobraziť v prehliadači alebo ako HTML dokument. Aby sme tento dokument vedeli ľahko upraviť, tak sa z neho vytvorí objektová reprezentácia. Obsah a štruktúra dokumentu zostávajú zachované. DOM sa potom dá upravovať pomocou rôznych programovacích jazykov.

Štandardy W3C DOM a WHATWG DOM sú implementované vo väčšine prehliadačov. Veľa prehliadačov implementuje rôzne rozšírenia, preto je potrebné si dať pozor na kompatibilitu pri využívaní týchto rozšírení.

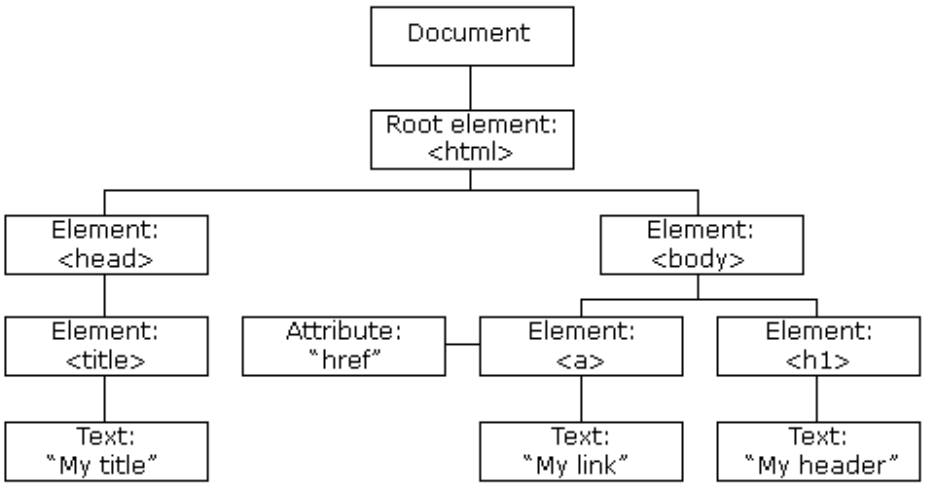
Objektový model dokumentu implementuje jednotlivé HTML elementy ako objekty. Tieto objekty majú vlastnosti a metódy. Pomocou nich vieme pristupovať k jednotlivým atribútom a potomkom elementu HTML, umožňujú nám vkladať, upravovať a odstraňovať tieto elementy, alebo medzi nimi prechádzať.

DOM je navrhnutý, tak aby bol nezávislý od konkrétneho programovacieho jazyka. Každý jazyk s ním môže pracovať svojim spôsobom. Keďže v práci pracujem s jazykom JavaScrip a TypeScript, tak popíšem ako sa konkrétne v nich pracuje s DOM.

3.3.1 DOM a jazyk JavaSript

Na začiatku boli jazyk JavaSript a DOM veľmi prepletené. Neskôr sa oddelili do samostatných entít. Použitie DOM v jazyku JavaSript je veľmi jednoduché. Nie je potrebné sťahovať žiadnu dodatočnú knižnicu. Objektový model dokumentu je dostupný pomocou objektov `document` a `window`.

Objekt `window` reprezentuje okno prehliadača a objekt `document` je koreňom daného dokumentu. Objekt `document` sa skladá z uzlov typu `Node`. V HTML dokumente, uzol reprezentuje HTML element, text alebo atribút HTML elementu. Základným typom uzlu je `element`. Element v HTML dokumente môže reprezentovať HTML element. `NodeList` je pole elementov.



Obr. 3.1: Ukážka občajného DOM dokumentu v jazyku HTML [20].

Pomocou tohto rozhrania môžme napríklad vyhľadávať elementy podľa identifikátora (`document.getElementById(id)`) alebo názvu značky (`document.getElementsByTagName(name)`). Môžeme vytvárať alebo upravovať nové elementy (`document.createElement(name)`). Rozhranie nám umožňuje meniť štýl alebo atribúty elementu (`element.setAttribute()`), či zobraziť si obsah okna alebo posunúť sa na určité miesto dokumentu (`window.scrollTo()`).

Rozhranie objektu `document` umožňuje taktiež ľahký prístup k skriptom pomocou príkazu `document.scripts`.

Manipulácia s objektovým modelom dokumentu v jazyku TypeScript je rovnaká.

3.4 Prístup k vzdialeným webovým stránkam

V tejto sekcii sa pozriem na rôzne spôsoby, akými sa dá pristúpiť k webovým zdrojom, stiahnuť a vytiahnuť potrebné informácie.

3.4.1 Web scrapping

Je to proces získavania dát z webových stránok. Je známejší pod pojmom extrakcia dát. Poskytuje inteligentnú automatizácia umožňujúcu získavanie dát z obrovského množstva zdrojov. Proces extrakcie dát má dve časti: web crawler a web scraper. Web crawler alebo tiež nazývaný pavúk, prehľadáva internet a hľadá užitočné dáta. Web scraper následne tieto dáta extrahuje z webovej stránky [21].

Ďalej by som uviedol niektoré knižnice a frameworky pre web scrapping v jazyku JavaSript.

Angular HttpClient

Angular HttpClinet je zjednodušená implementácia HTTP klienta v prostredí angularu. Táto knižnica implementuje všetky základné metódy HTTP požiadavku. Príklad využitia tohto nástroja sa nachádza na obrázku 3.2.

```
async getSources(url: string): Promise<any> {
  return await this.http.get(url, { responseType: 'text' })
    .toPromise()
    .catch((error) => console.error(error));
}
```

Obr. 3.2: Ukážka použitia knižnice HttpClient vo frameworku Angular.

Request a Cheerio

Pre extrahovanie dát pomocou jazyka Javascript, môžeme použiť dvojicu nástrojov: knižnica Request a Cheerio. Informácie som čerpal z tohto zdroja [18].

Request-Promise je jednoduchý HTTP klient, ktorý umožňuje jednoducho volať HTTP metódy. Pomocou tohto nástroja tak môžeme jednoducho získať zdroje webových stránok. Request je od začiatku roku 2020 oficiálne zastaraný.

Cheerio slúži na spracovanie kódu v značkovacom jazyku. Tento nástroj poskytuje rozhranie pre prechádzanie a manipuláciu v kóde. Cheerio implementuje podmnožinu jadra jQuery a odstraňuje rôzne chyby a obmedzenia z tejto knižnice. Príklad využitia tohto nástroja sa nachádza na obrázku 3.3.

```
var request = require('request');
var cheerio = require('cheerio');

request('https://news.ycombinator.com', function (error, response, html) {
  if (!error && response.statusCode == 200) {
    var $ = cheerio.load(html);
    $('span.comhead').each(function(i, element){
      var a = $(this).prev();
      console.log(a.text());
    });
  }
});
```

Obr. 3.3: Ukážka použitia nástrojov Request a Cheerio [14].

Puppeteer

Puppeteer je Node.js knižnica, ktorá poskytuje rozhranie na kontrolu prehliadača Chrome alebo Chromium pomocou protokolu DevTools. Pupputeer umožňuje vykonávať radu vecí, ktoré sa dajú v prehliadači robiť manuálne, napríklad generovanie snímkov obrazovky, testovanie UI, vyhľadávanie a otváranie webových stránok, web scraping a ďalšie [12].

Príklad využitia tohto nástroja sa nachádza na obrázku 3.4.

Apify SDK

Apify SDK je open-source Node.js knižnica pre extrakciu zdrojov. Apify SDK poskytuje web crawler aj s web scraper. Táto knižnica okrem toho umožňuje automatizáciu webových procesov. Tento nástroj je možné používať u seba lokálne alebo na Apify cloude. Informácie som čerpal z tohto zdroja [23].

Príklad využitia tohto nástroja sa nachádza na obrázku 3.4.

```
const Apify = require('apify');

Apify.main(async () => {
    const requestQueue = await Apify.openRequestQueue();
    await requestQueue.addRequest({ url: 'https://www.iana.org/' });
    const pseudoUrls = [new Apify.PseudoUrl('https://www.iana.org/[.*]')];

    const crawler = new Apify.PuppeteerCrawler({
        requestQueue,
        handlePageFunction: async ({ request, page }) => {
            const title = await page.title();
            console.log(`Title of ${request.url}: ${title}`);
            await Apify.utils.enqueueLinks({
                page,
                selector: 'a',
                pseudoUrls,
                requestQueue,
            });
        },
        maxRequestsPerCrawl: 100,
        maxConcurrency: 10,
    });

    await crawler.run();
});
```

Obr. 3.4: Ukážka použitia knižnice Apify SDK v kombinácií s nástrojom Puppeteer [24].

Porovnanie nástrojov

Knižnica HttpClient vo frameworku Angular slúži na vykonávanie HTTP požiadaviek. Pomocou tejto knižnice je len možné stiahnuť webové zdroje. Kombinácia knižníc Request a Cheerio umožňuje okrem stiahnutia webových zdrojov aj ich spracovanie. Knižnica Puppeteer pracuje hlavne s prehliadačom Chrome. Oproti týmto všetkým knižniciam je Apify SDK plnohodnotnou knižnicou pre web scraping.

3.4.2 CORS

Skratka CORS stojí za Cross-Origin Resource Sharing. CORS je mechanizmus, ktorý umožňuje aplikácií bežiacej na určitej doméne, pristupovať k vybraným zdrojom iného pôvodu (origin). Informácie o CORS som čerpal z tohto zdroja [8].

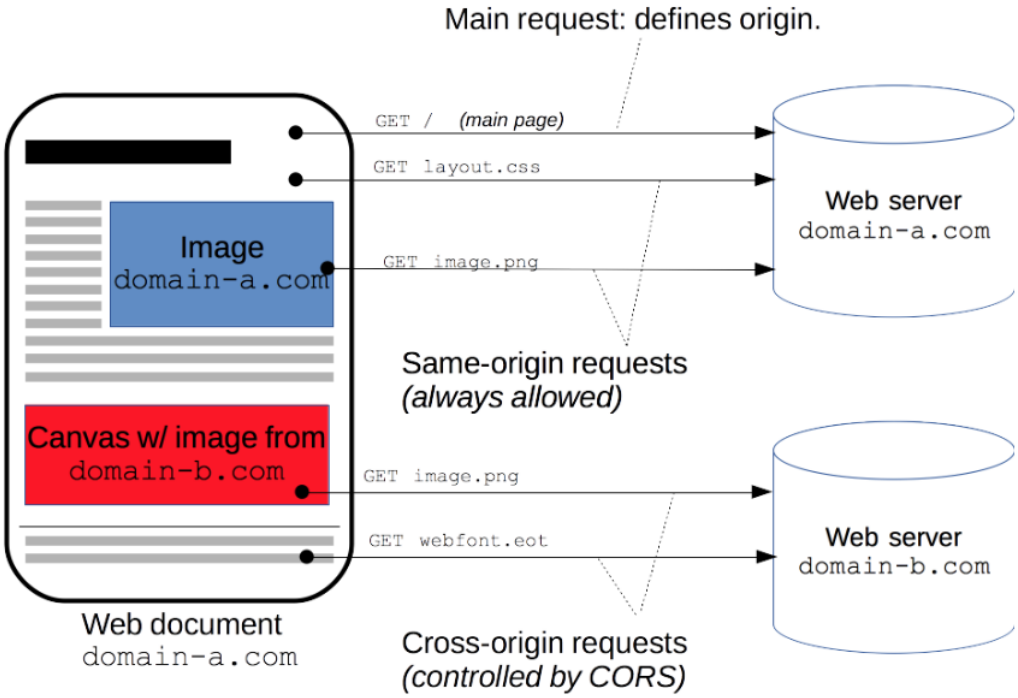
Tento mechanizmus používa na tento účel HTTP hlavičky požiadavkov. Na obrázku 3.5 sa nachádza ilustrácia fungovania mechanizmu CORS.

Pôvod je definovaný protokolom, doménou a portom, napríklad `https://domain-a.com` a `https://domain-b.com` majú rozličný pôvod.

Kvôli bezpečnostným dôvodom prehliadače neumožňujú cross-origin HTTP požiadavky spúšťané zo skriptov. Webové aplikácie tak môžu pristupovať len k zdrojom rovnakého pôvodu na akom beží aplikácia. Ak aplikácia chce pristúpiť k zdrojom cudzieho pôvodu, tak HTTP požiadavka musí obsahovať správnu CORS hlavičku.

Mechanizmus CORS využívajú napríklad HTTP požiadavky po webových fontoch, obrázkoch, WebGL textúrach a ďalšie.

V jazyku JavaScript je možné pre ľahké využitie CORS použiť objekt `XMLHttpRequest`. Príklad použitia je možné vidieť na obrázku 3.6.



Obr. 3.5: Ukážka spôsobu fungovania mechanizmu CORS [8].

```
const xhr = new XMLHttpRequest();
const url = 'https://bar.other/resources/public-data/';

xhr.open('GET', url);
xhr.onreadystatechange = someHandler;
xhr.send();
```

Obr. 3.6: Príklad použitia objektu XMLHttpRequest [8].

Kapitola 4

Validátory webových stránok

Validátor je program, ktorý kontroluje kód po syntaktickej stránke. Existuje obrovské množstvo rôznych validátorov. V práci sa zamerávam na validátory v online prostredí.

Validátory môžu poskytovať validáciu pre jeden typ programovacieho jazyka alebo ponúkajú možnosť validácie pre niekoľko rôznych druhov programovacích jazykoch. Najčastejšie umožňujú validáciu dokumentov, ktoré si užívateľ môže nahrať do validátora, nakopírovať kód do editora, ktorý validátor poskytuje, alebo pre validáciu webových stránok umožňuje validáciu pomocou zadanej adresy URL. Niektoré validátory ponúkajú aj aplikačné rozhranie, pomocou ktorého ich vie programátor použiť priamo vo svojom kóde. V tejto kapitole sa pozriem na validátory, ktoré používam vo svojej práci. Na konci kapitoly uvediem pre porovnanie aj iné druhy validátorov.

4.1 W3C Markup Validation Service

Markup Validation Service je validátor vytvorený organizáciou W3C. Validátor kontroluje webové dokumenty, napísané v jazykoch HTML, XHTML, SMIL, MathML a ďalších. Informácie o tomto validátore som čerpal z jeho dokumentacie [27].

Validátor umožňuje validáciu online dokumentov pomocou zadania URL, nahraných dokumentov, alebo priamo napísaného kódu v editore na stránke validátora. Pred validáciou si užívateľ môže nastaviť parametre validácie, ako napríklad kódovanie znakov, typ dokumentu a jeho verziu.

Výsledok validácie sa zobrazí na samostatnej stránke. Tento validátor zobrazuje nájdené chyby a varovania, ktoré sú farebne zvýraznené na priloženej časti chybného kódu. Validátor zobrazuje užívateľovi pozíciu nájdenej chyby alebo varovania; riadok, stĺpec začiatku chyby a riadok, stĺpec konca chyby. Výsledok validácie je možné filtrovať. Užívateľ si môže vybrať, či chce zobraziť len chyby, alebo varovania.

Validátor je možné využiť online na webovej stránke validátora[1]. Organizácia W3C si umožňuje W3C Markup Validation Service stiahnuť a nainštalovať[2]. W3C Markup Validation Service je k dispozícií za podmienok W3C softvérovej licencie[3]. Organizácia W3C ponúka kód na vzhliadnutie[4].

[1] https://validator.w3.org/
[2] https://validator.w3.org/source/
[3] https://www.w3.org/Consortium/Legal/2015/copyright-software-and-document
[4] https://github.com/w3c/markup-validator/

4.1.1 Aplikačné rozhranie

Okrem vyššie spomenutého, W3C Markup Validation Service poskytuje aj aplikačné rozhranie. Validátor poskytuje HTTP rozhranie, ktoré sa dá volať ako normálna webová služba. Aplikačné rozhranie umožňuje zvoliť si z viacerých druhov vstupov a výstupov. Výslednú správu je navyše možné voliteľne komprimovať.

Vstupom do validátoru môže byť:

- URL validovaného dokumentu zadaná ako parameter HTTP metódy GET,
- telo validovaného dokumentu zadané v tele HTTP metódy POST (odporúčaný spôsob),
- telo validovaného dokumentu zadané, ako položka vo formulári HTTP metódy POST (tento spôsob nie je odporúčaný),
- nahraný validovaný dokument zadaný, ako položka vo formulári HTTP metódy POST (tento spôsob taktiež nie je odporúčaný).

Výstupom validátora môžu byť:

- HTML (predvolený formát),
- XHTML,
- XML,
- JSON,
- chybový formát GNU,
- obyčajný text čitateľný človekom.

Dokumentácia odporúča použiť ako výstupy webovej služby formáty JSON a XML. Dokumentácia odporúča tieto formáty kvôli kompatibilite do budúcnosti a jednoduchosti použitia týchto formátov.

Ako bolo už vyššie spomenuté, aplikačné rozhranie W3C Markup Validation Service podporuje voliteľné komprimovanie správ. Validátor umožňuje komprimovanie požiadaviek a aj odpovedí. Túto možnosť je potrebné správne uviesť v hlavičke HTTP požiadavky.

4.2 W3C CSS Validation Service

W3C CSS Validation Service je validátor vytvorený organizáciou W3C. Validátor kontroluje webové dokumenty napísané v jazyku CSS a SVG. Informácie o tomto validátore som čerpal z jeho dokumentacie [28].

Tento validátor umožňuje validáciu online dokumentov pomocou zadania URL, nahraných dokumentov alebo priamo napísaného kódu v editore na stránke validátora. Validátor umožňuje nastaviť parametre validácie, ako napríklad profil jazyka CSS, zobrazenie chýb len určitej závažnosti, médium na ktorom sa zobrazí obsah a spôsob, akým sa budú validovať rôzne rozšírenia.

Výsledok validácie sa zobrazí na samostatnej stránke. Tento validátor zobrazuje nájdené chyby a varovania, ktoré je možné vidieť na priloženej časti chybného kódu. Validátor

zobrazuje užívateľovi číslo riadku nájdenej chyby alebo varovania. Výsledok validácie nie je možné filtrovať. Chyby a varovania sú zobrazené zvlášť. Užívateľ môže vidieť, v ktorom dokumente sa nachádza tá ktorá chyba, alebo varovanie. Na konci hlásenia sa nachádza zoznam všetkých validovaných súborov aj s kódom.

Validátor je možné využiť online na webovej stránke validátora[5]. Organizácia W3C si umožňuje W3C Markup Validation Service stiahnuť a nainštalovať[6]. W3C Markup Validation Service je k dispozícií za podmienok W3C softvérovej licencie[7]. Organizácia W3C ponúka kód na vzhliadnutie [8].

4.2.1 Aplikačné rozhranie

Okrem vyššie spomenutého, W3C CSS Validation Service poskytuje aj aplikačné rozhranie. Validátor poskytuje HTTP rozhranie, ktoré sa dá volať ako normálna webová služba.

Webovú službu je možné použiť pomocou HTTP metódy GET. Ako parametre metódy môžeme uviesť URL dokumentu (môže sa jednať o dokument v jazyku CSS alebo HTML), ktorý chceme validovať, alebo text dokumentu. Ďalej je možné špecifikovať medzi parametrami rôzne možnosti validácie, ktoré som uviedol v úvode tejto sekcie.

Je taktiež možné špecifikovať druh výstupu validátora ako parameter v hlavičke GET požiadavky. Jedná sa o tieto druhy formátov výstupov:

- text/html (prednastavený formát),
- application/xhtml+xml,
- application/soap+xml,
- text/plain.

4.3 JSHint

JSHint je open-source nástroj, ktorý detekuje chyby a potencionálne problémy v javascriptovom kóde pomocou statickej analýzy kódu. Je ľahko prispôsobiteľný rôznym prostrediam a ponúka širokú ponuku nastavení analýzy kódu. Informácie o tomto validátore som čerpal z jeho dokumentacie [31].

Je to overený nástroj, ktorý používajú inžinieri známych značiek ako Mozzila, Wikipedia, Facebook, Twitter, RedHat, Google a mnoho ďalších.

Je ho možné použiť pomocou príkazového riadku, alebo ako modul prostredia Node.js. Ďalej by som viac rozviedol aplikačné programovacie rozhranie Node.js modulu.

4.3.1 Aplikačné rozhranie

JSHint ponúka javascriptové aplikačné rozhranie, využiteľné v prostrediach ako webové prehliadače alebo Node.js. Aplikačné rozhranie je sprístupnené pomocou funkcie:

```
JSHINT(source, options, predef)
```

.

[5] https://jigsaw.w3.org/css-validator/
[6] https://jigsaw.w3.org/css-validator/DOWNLOAD.html
[7] https://www.w3.org/Consortium/Legal/2015/copyright-software-and-document
[8] https://github.com/w3c/css-validator

Ak sa v kóde zavolá táto funkcia, spustí sa validácia kódu zadaného v parametri `source`. Užívateľ si môže pomocou parametra `options` zvoliť zo širokej ponuky rôzne spôsoby analýzy kódu. Parameter `predef` umožňuje zadať premenné definované mimo súčasného súboru.

Výsledok validácie je možné dostať v premennej `JSHINT.errors` ako pole jednotlivých chýb a varovaní. Použitím metódy `JSHINT.data()` sa vygeneruje správa obsahujúca detaily poslednej analýzy kódu.

4.3.2 Možnosti analýzy kódu

Užívateľ si môže pred spustením analýzy javascriptového kódu zvoliť, čo chce a nechce analyzovať. Príklady niektorých nastavení analýzy kódu:

- nastavenie analyzovanej ECMAStript špecifikácie,
- používanie premenných pred definíciou,
- spôsob zatieňovania premenných,
- varovania pri použití prísneho režimu,
- používanie nedeklarovaných premenných,
- varovania o nevyužitých premenných,
- varovania o používaní deklarácií premenných pomocou kľúčového slova `var`.

4.3.3 Ostatné validátory

Mnoho webových validátorov používa vyššie spomenuté validátory.

Validator.Nu[9]

Validator.Nu je validátor dokumentov napísaných v jazykoch HTML5, XHTML5. Umožňuje validáciu kódu z nahraného súboru, url alebo zadaného kódu v editore na webovej stránke. Tento validátor poskytuje webové aplikačné rozhranie. Ďalšie informácie je možné získať z tohto zdroja [22].

FreeFormatter.com[10]

FreeFormatter ponúka širokú ponuku služieb od validátorov, cez formátovacie nástroje, kódery, dekódery, konvertery a ďalšie. FreeFormatter umožňuje validáciu dokumentov typu HTML, JSON, XML, XPath a ďalšie. Validovaný kód je možné vložiť priamo na stránke pomocou jednoduchého editora alebo nahrať súbor. Tento nástroj poskytuje možnosť nastaviť typ kódovania dokumentu. Výstupom validátora je zoznam chýb. Nie je ho možné filtrovať.

FreeFormatter neposkytuje žiadne aplikačné rozhranie. Je založený na Validator.Nu, čo je validátor od oragnizácie W3C.

[9] https://html5.validator.nu/
[10] https://www.freeformatter.com/html-validator.html

Code Beautify CSS Validator[11]

Code Beautify Css Validator ponúka možnosť validácie dokumentov napísaných v jazyku CSS. Služba Code Beautify ponúka okrem tohto validátora aj množstvo ďalších služieb (konvertéry, editory pre rôzne programovacie jazyky a ďalšie). Ponúka podobné možnosti validácie ako vyššie uvedené validátory. Na rozdiel od nich neponúka žiadne aplikačné rozhranie. Tento validátor funguje čisto ako webová aplikácia.

CSSLint[12]

CSSLint je open-source nástroj pre validáciu dokumentov napísaných v jazyku CSS. CSSLint umožňuje validáciu len pomocou zadaného kódu priamo do editora na stránke. Napriek tomu ale tento nástroj ponúka širokú ponuku rôznych druhov chýb a upozornení na testovanie. CSSLint neponúka žiadne webové API. Nástroj je možné ale stiahnuť a používať ho v príkazovom riadku. Ďalšie informácie je možné získať z tohto zdroja [2].

ESLint

ESLint je nástroj na identifikovanie a hlásenie chýb nájdených v kóde napísanom v jazyku JavaScript. ESLint umožňuje množstvo prispôsobení kontroly kódu ako definície výstupov, špecifikovanie pravidiel a pluginov, hlásenie chýb, automatické opravy chýb a ďalšie. Neposkytuje aplikačné rozhranie. Je ho možné stiahnuť pomocou nástroja npm. Ďalšie informácie je možné získať z tohto zdroja [16].

Esprima[13]

Esprima je nástroj pre lexikálnu a syntaktickú analýzu javascriptového kódu. Tento nástroj je možné použiť v prehliadači, ako Node balíček priamo v kóde, či v prostrediach Rhino a Nashorn. Ďalšie informácie je možné získať z tohto zdroja [15].

[11] https://codebeautify.org/cssvalidate
[12] http://csslint.net/
[13] https://esprima.org/demo/validate.html

Kapitola 5

Návrh webovej aplikácie

V tejto kapitole je popísaný môj návrh komplexného validátora. Tento návrh zahŕňa moju voľbu vybraných nástrojov, pomocou ktorých je aplikácia implementovaná, návrh užívateľského prostredia s vysvetlením výberu a umiestnením jednotlivých prvkov, návrh architektúry aplikácie, jej rozloženie do logických celkov, komunikácie medzi nimi a návrh dátových štruktúr. Kapitola je ukončená návrhom zoznamu chýb, ktoré by moja aplikácia mala byť schopná opraviť.

5.1 Výber nástrojov

Komplexný validátor webových stránok je webová aplikácia. Pre účely implementácie budem preto vyberať z dostupných nástrojov slúžiacich pre vytváranie práve takýchto aplikácií. Aplikácia má svoj frontend a backend, ktoré si vyžadujú rozličné technológie. Tieto technológie budú tvoriť základ mojej aplikácie, od ktorého sa následne bude odvíjať jej architektúra a ďalšie použité nástroje. Kvôli tomuto dôvodu som sa najprv pozrel na ponuku týchto nástrojov a na základe druhu mojej aplikácie a mojich predchádzajúcich skúseností som vybral vhodné nástroje na implementáciu.

Podľa zadania práce by moja aplikácia mala byť schopná validovať tri druhy jazykov: jazyk HTML, CSS a JavaSript. K týmto jazykom som sa preto snažil nájsť vhodné validátory a preskúmať ich aplikačné rozhrania, aby som vedel, aké mám možnosti pri ich použití v kóde.

Pri implementácií kódu som následne narazil na problémy a napadli mi nové nápady, vylepšenia, s ktorými som nerátal v pôvodnom návrhu. Použitie ďalších nástrojov, ktoré som pridal do mojej aplikácie počas samotnej implementácie uvediem v kapitole 6.

5.1.1 Frontend

Medzi najpoužívanejšie a najznámejšie nástroje na vývoj frontendu môžeme zaradiť React, Angular a Vue.js. Každý nástroj má svoje výhody a nevýhody. V skratke predstavím jednotlivé nástroje.

React

React je knižnica od spoločnosti Facebook, slúžiaca pre vytváranie frontendu. Jej špecifickosť spočíva v použití virtuálneho DOM. Táto vlastnosť umožňuje tejto knižnici veľmi rýchlo aktualizovať jednotlivé zmeny bez dopadu na zvyšok aplikácie. Dáta v aplikácií prú-

dia jedným smerom, čo umožňuje vytvárať stabilný kód. Na rozdiel od angularu je React len knižnicou. Umožňuje na jednej strane vysokú flexibilitu, na druhej strane umožňuje len implementáciu pohľadov. Veci, ako napríklad smerovanie, je potrebné riešiť inými externými knižnicami.

Angular

Angular je framework od spoločnosti Google. Angular taktiež slúži na vývoj frontendu. Angular je kompletný framework. Angular je špecifický rozdelením jednotlivých častí aplikácie do komponentov a služieb. Tento framework umožňuje jednosmerný a aj obojsmerný tok dát. Angular používa TypeScript, čo môže byť veľká výhoda.

Vue.js

Vue.js je framework pre vytváranie frontendu. Vue.js má nižšiu popularitu ako knižnica React a framework Angular. Tento framework je ale stále radený medzi najpoužívanejšie nástroje. Vue.js je ako Angular kompletný framework. Vue.js tak isto poskytuje obojsmerný tok dát.

Výber frontend nástroja

Pri výbere nástroja pre implementáciu frontendu som sa rozhodoval na základe niekoľkých kritérií . Pre mňa osobne bolo potrebné, aby mi framework poskytoval čo najviac funkcionalít, aby som nič nové nemusel pridávať. Dôležitá pre mňa bola taktiež moja znalosť nástroja, aby som sa stretol s čo najmenším množstvom komplikácií a bol si vedomý možností daného frameworku. Framowork Vue.js som preto okamžite vylúčil, keďže som s ním nikdy nepracoval. Mal som menšie skúsenosti s knižnicou React, ale len základné. Framework Angular som poznal najlepšie, keďže som z predchádzajúcich projektov a práce mal s ním skúsenosti. Na implementáciu frontendu som si preto vybral framework Angular.

Pre vytváranie vzhľadu som použil knižnice Bootstrap a ng-bootstrap, čo je implementácia/rozšírenie knižnice Bootstrap pre framework Angular.

5.1.2 Backend

Backend mojej aplikácie by mal byť čo najjednoduchší. Pre účely mojej aplikácie nebudem potrebovať žiadnu databázu. Na backend by som chcel preto delegovať úlohy, ktoré nebude možné vykonávať priamo v prehliadači užívateľa, a to prácu so súbormi a iné. Dôležitými kritériami pri výbere nástroja pre backend bola preto jednoduchosť použitia a funkcionalít a taktiež moje predchádzajúce skúsenosti s danou technológiou. Medzi známe nástroje patria Express, Django a Laravel. Nástroje ako Rail alebo Spring som hneď zo začiatku vyradil, keďže s jazykom Ruby a Java nemám veľa skúseností.

Express

Express je Node.js framework. Express umožňuje jednoduchú tvorbu REST API v jazyku JavaSript.

Django

Django je webový framework, ktorý používa Python. Django podporuje MVC architektúru alebo ORM (Object Relational Mapper).

Laravel

Laravel je jeden z najpoužívanejších webových frameworkov. Laravel používa jazyk PHP.

Výber backend nástroja

Ako som uviedol, dôležitými kritériami pri výbere nástroja bola jednoduchosť použitia, a moja znalosť daných technológií. Všetky vyššie spomenuté nástroje nie sú ťažké na implementáciu. Skúsenosti som mal však len s frameworkom Express. Express navyše využíva jazyk JavaScript, s ktorým mám viacej skúseností, ako s jazykmi PHP alebo Python.

5.1.3 Validátory

V kapitole 4 som popísal rôzne druhy existujúcich validátorov. Pre moju webovú aplikáciu som si zvolil následovné: validátor W3C Markup Validaton Service pre validáciu jazyka HTML, validátor W3C CSS Validation Service pre validáciu jazyka CSS a validátor JSHint pre jazyk JavaSript. Práve tieto validátory pre jazyky HTML a CSS som si vybral kvôli tomu, že sú od organizácie W3C, ktorá definuje štandardy pre tieto jazyky. Okrem toho ponúkajú webové aplikačné rozhranie, pomocou ktorého je možné služby týchto validátorov jednoducho volať v aplikácií. Validátor JSHint som zvolil kvôli jeho jednoduchému aplikačnému rozhraniu a širokej možnosti prispôsobenia samotnej validácie.

5.2 Popis vybraných nástrojov

V tejto časti popisujem podrobnejšie vybrané nástroj pre implementáciu ako runtime prostredie Node.js, jeho framework Express, framework Angular a jazyk Typescript. V rámci týchto technológií používam aj programovacie jazyky ako JavaSript, HTML alebo CSS.

5.2.1 Node.js

Node.js je open-source a multiplatformové runtime prostredie jazyka JavaScript. Informácie som čerpal z [6].

Node.js spúšťa javascriptový modul V8, čo je jadro prehliadača Google Chrome, mimo tohto prehliadača. Toto umožňuje, aby bol Node.js veľmi výkonný a dal sa spustiť aj na iných prostrediach.

Aplikácia napísaná pomocou Node.js beží v jednom procese bez toho, aby sa pre každú požiadavku vytvorilo nové vlákno. Node.js poskytuje vo svojej štandardnej knižnici súbor asynchrónnych vstupno-výstupných primitív, ktoré bránia blokovaniu kódu v jazyku JavaScript. Vo všeobecnosti sú knižnice v Node.js napísané pomocou neblokujúcich paradigiem, čo robí blokovacie správanie skôr výnimkou ako pravidlom.

Keď Node.js potrebuje vykonať I/O operáciu, tak namiesto blokovania vlákna a zbytočného plytvania cyklov procesora, Node.js obnoví operácie vrátení odpovedi. Tento návrh umožňuje Node.js zvládnuť tisíce súčasných pripojení k jednému serveru bez toho, aby sa muselo riešiť viacvláknové programovanie, čo by bolo významným zdrojom chýb.

V Node.js je možné používať akékoľvek štandardy ECMASript bez problémov, pretože nemusíte čakať na všetkých svojich užívateľov, aby si aktualizovali svoje prehliadače.

5.2.2 Express

Express je minimálny a flexibilný Node.js webový framework, ktorý poskytuje sadu funkcií pre webové a mobilné aplikácie. Slúži na vytváranie backendu webových aplikácií. Obsahuje metódy protokolu HTTP a middleware. Express poskytuje tenkú vrstvu základných funkcií webových aplikácií bez toho, aby zakrývalo funkcie Node.js [3].

5.2.3 Angular

Angular je framework slúžiaci pre vývoj single-page aplikácií pomocou využitia jazk HTML a jazyka TypeScript. Tento framework implementuje rôzne funkcionality ako sadu typescriptových knižníc, ktoré je možné vkladať do aplikácie. Informácie o frameworku Angular som čerpal z týchto zdrojov [4] a [5].

Architektúra frameworku Angular je postavená na niekoľkých základných konceptoch. Tieto koncepty a ich vzájomné vzťahy sú zobrazené na obrázku 5.1. Jedným z konceptov sú NgModules, ktoré poskytujú kontext kompilácie pre komponenty. Sada NgModules definuje angularovú aplikáciu. Aplikácia má aspoň jeden koreňový modul, ktorý umožňuje samozavádzanie. Typicky okrem tohto koreňového modulu môže mať niekoľko funkčných modulov.

Komponenty definujú pohľady, ktoré sú sadou rôznych elementov zobrazených na obrazovke. Angular si vie medzi nimi vyberať a následne ich meniť podľa logiky programu a dát.

Komponenty využívajú služby, ktoré poskytujú špecifickú funkcionalitu. Táto funkcionalita nemusí priamo súvisieť s pohľadmi. Služby môžu byť preto ľubovoľne vkladané do rôznych komponentov. Týmto sa zabezpečí, že kód je modulárny a znovupoužiteľný.

Komponenty a aj služby sú jednoduché triedy s dekorátormi. Tieto dekorátory definujú ich typ, spolu s metadátami, podľa ktorých Angular vie, ako ich použiť.

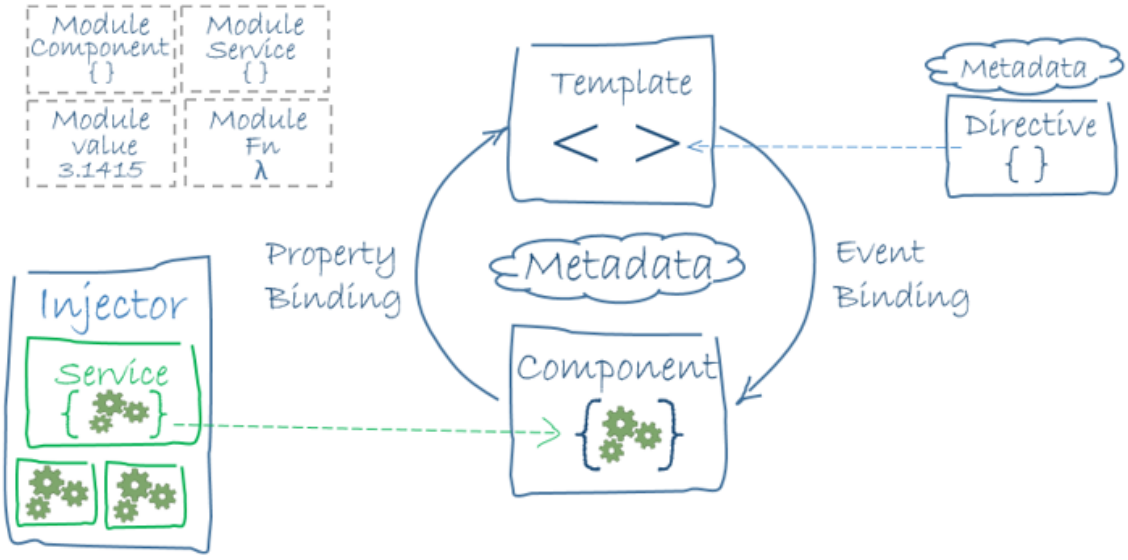
V aplikácií je väčšinou definovaná celá rada komponentov, ktoré sú hierarchicky zoradené. Pre navigáciu medzi jednotlivými komponentami a ich pohľadmi slúži služba Router, ktorú tento framework poskytuje. Angular touto službou tak poskytuje prepracované možnosti v prehliadači (napríklad prechod na určité stránky, len pre autentifikovaného používateľa).

Pohľad komponentu je definovaný pomocou šablóny napísanej v jazyku HTML. Šablóny sú dynamické. Keď ich Angular vykreslí, transformuje DOM podľa pokynov uvedených direktívami. Angular pozná dva druhy direktív: štrukturálne a atribútové.

Štrukturálne direktívy menia rozloženie pohľadu pridaním, odstránením alebo nahradením prvkov v DOM. Používajú sa dva druhy štrukturálnych direktív:

- *ngFor - je to iteračná direktíva; pre každý prvok v zozname vytvorí element v DOM,
- *ngIf - je to podmienková direktíva; zobrazí element alebo vykoná akciu pri splnení podmienky

Atribútové direktívy menia vzhľad, správanie existujúceho prvku alebo slúžia pre previazanie dát. Tieto direktívy vyzerajú ako bežné atribúty HTML. Príkladom môže byť direktíva ngModel, slúžiaca pre jednosmerné alebo obojsmerné previazanie dát.



Obr. 5.1: Znázornenie jednotlivých prvkov frameworku Angular vo vzájomných vzťahoch [4].

5.2.4 Jazyk TypeScript

Jazyk TypeScript je open-source programovací jazyk. Tento jazyk je supersetom jazyka JavaScript. To znamená, že obsahuje všetky funkcionality jazyka JavaScriptu a k ním pridáva rôzne rozšírenia a zlepšenia. V prípade jazyka Typescript je to statické typovanie. Je preto veľmi vhodný pri tvorení väčších projektov. Jazyk TypeScript sa kompiluje na štandardný a čitateľný kód v jazyku JavaSript [7].

5.3 Funkcionalita aplikácie

Webová aplikácia má byť schopná podľa zadania práce validovať kód webových stránok. Webové stránky sú väčšinou tvorené pomocou troch technológií: jazyk HTML, CSS a JavaSript. Komplexný validátor by preto mal byť schopný validovať aspoň tieto tri jazyky.

Aplikácia má výsledky validácie vhodným spôsobom zobrazovať užívateľovi. Aplikácia by ich mala vhodne triediť, napríklad na chyby a varovania, alebo podľa príslušnosti chyby k danému dokumentu.

Aplikácia má byť schopná užívateľovi umožniť rôzne vstupy validácie. Užívateľ si môže validovať nahrané súbory. Tieto súbory si môže nahrať samostatne alebo v celom priečinku s príponou zip. Užívateľ ďalej môže validovať webovú stránku pomocou zadania jej URL. Aplikácia chybné dokumenty stiahne. Následne si ich môže užívateľ opraviť a takto opravené dokumenty stiahnuť (ak je autorom dokumentov). Nakoniec môže validovať kód, ktorý zadá do editoru.

Užívateľ si môže kód pred validáciou alebo priamo po nej prispôsobovať a upravovať v editore.

Aplikácia ponúka návrhy opráv pre sadu základných chýb. Ak užívateľovi návrh opravy chyby vyhovuje, môže ho stlačením tlačidla potvrdiť a opravu vykonať.

Aplikácia bude umožňovať validáciu celej webovej stránky, pomocou stlačenia jedného tlačítka, to znamená, že sa spustí validácia kódu v jazyku HTML, CSS a aj JavaSript. Užívateľ si môže validovať jednotlivé dokumenty samostatne, jednotlivo pre každý ponúkaný druh validátora.

Po ukončení validácie a opráv kódov si užívateľ môže tieto súbory stiahnuť vo formáte zip súboru. Aplikácia ponúka možnosť stiahnuť diff pre patch.

5.4 Návrh užívateľského prostredia

Úplne na začiatku návrhu aplikácie komplexného validátora som vytvoril návrh užívateľského prostredia. Pri navrhovaní som vychádzal zo zoznamu funkcionalít aplikácie. Začal som vytváraním prvotného mockupu pomocou nástroja Balsamiq Wireframes 4. Tento mockup popíšem v tejto sekcii.

Aplikácia je ladená do tmavých farieb, keďže tento štýl je aktuálne veľmi obľúbený a veľa nových aplikácií dostáva nový tmavý vzhľad. V tejto práci nezamýšľam urobiť návrh iných druhov štýlov a farieb. Je to možné dorobiť pri pokračovaní vývoju komplexného validátora. Užívatelia určite v budúcnosti ocenia prepínanie medzi rôznymi grafickými štýlmi.

Na obrázku 5.2 sa nachádza úvodná obrazovka aplikácie. Užívateľ si môže vybrať, ktorý validátor chce použiť a akým spôsobom chce vložiť do validátore zdroje pre validáciu. Úvodná obrazovka neumožňuje vloženie validovaného kódu priamo do editora.

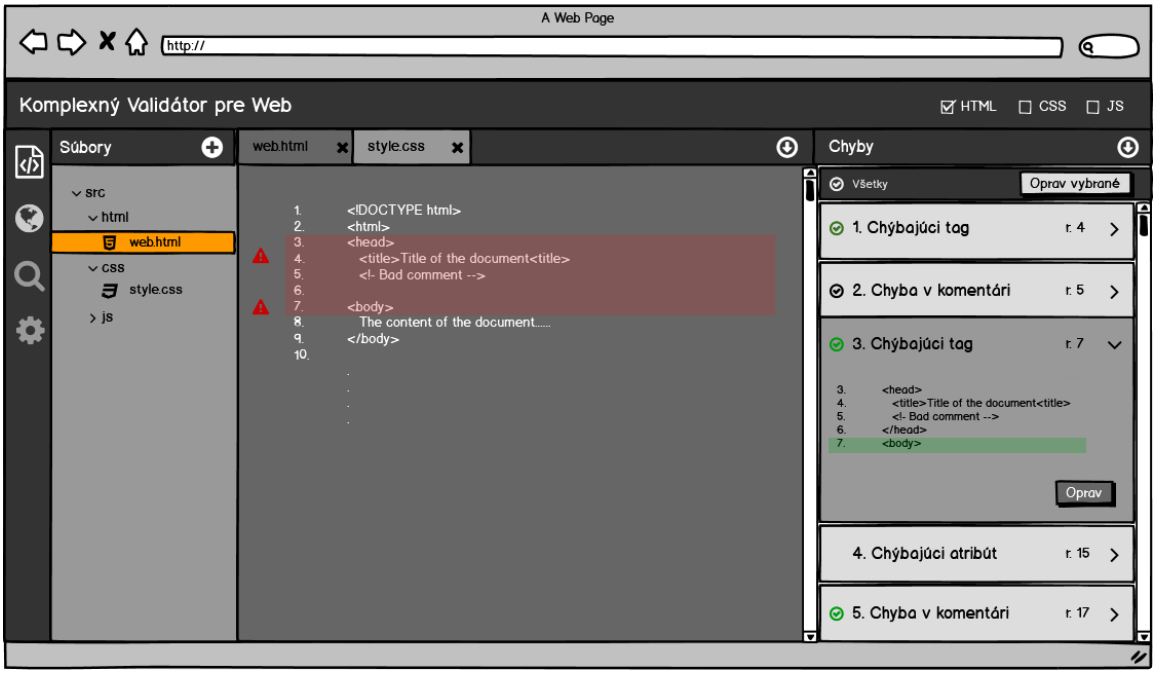
Obr. 5.2: Úvodná obrazovka aplikácie.

Hlavnú časť aplikácie je možné vidieť na obrázku 5.3. Celá aplikácia sa skladá len z tejto jedinej stránky. Jednotlivé časti sa potom následne menia vzhľadom na vybrané módy zadávania vstupov validácie, alebo aktuálne zapnutého validátora.

Návrh užívateľského rozhrania je rozdelený na niekoľko častí. Návrh obsahuje celkovo 4 častí:

- horná lišta, obsahujúca názov a prepínač validátorov,
- bočný panel umiestnený naľavo, obsahujúci prepínač medzi rôznymi vstupmi obsahu pre validáciu, spolu s nastaveniami a vyhľadávaním,
- editor,
- zoznam chýb.

Tieto jednotlivé časti ďalej rozvediem a vysvetlím.



Obr. 5.3: Hlavná časť aplikácie.

5.4.1 Horná lišta

Horná lišta obsahuje názov aplikácie a prepínač medzi validátormi. Detail prepínača je možné vidieť na obrázku 5.4. Umiestnil som ho navrch aplikácie, pretože táto časť je nemenná. Užívateľ tak má prepínač stále na očiach a vidí, ktorý druh validátora práve používa.

Obr. 5.4: Prepínač medzi jednotlivými validátormi, umiestnený v hornej lište.

5.4.2 Bočný panel

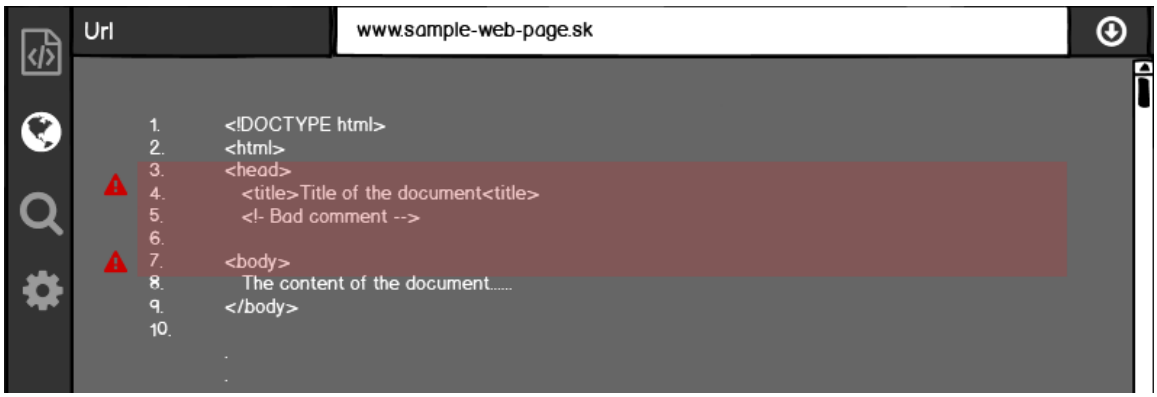
Bočný panel ponúka užívateľovi možnosť prepínať medzi rôznymi módmi vstupov validovaného obsahu. Umožňuje užívateľovi vybrať si medzi validáciou súborov, URL alebo zadaného kódu do editora.

Pôvodný návrh taktiež obsahuje, ako je zobrazené na obrázku 5.5 aj vyhľadávanie medzi súbormi a nastavenia možností validácie.

Po kliknutí na možnosť validácie súborov sa užívateľovi zobrazí okno so súborovým stromom, ako je možné vidieť na obrázku 5.5. Užívateľ si môže nahrať nové súbory. Nahrané súbory sa zobrazia v súborovom strome. Aktuálne zobrazený súbor je farebne zvýraznený.

Obr. 5.5: Bočný panel so zoznamom súborov.

Obrázok 5.6 zobrazuje možnosť validácie pomocou URL. Pri kliknutí na možnosť validácie pomocou URL sa užívateľovi zobrazí lišta, kde môže zadať URL, ktorú chce validovať. Napravo od lišty sa nachádza tlačidlo, pomocou ktorého si užívateľ môže stiahnuť opravené zdroje.



Obr. 5.6: Vzhľad obrazovky pri možnosti validácie URL.

5.4.3 Editor

Editor umožňuje užívateľovi vytvárať, upravovať a zobrazovať kód, ktorý prejde validáciou. Po prejdení kurzorom po chybe v zozname chýb je v editore táto chyba zvýraznená.

5.4.4 Zoznam chýb

Zoznam chýb sa nachádza v pravej časti obrazovky. Pôvodný návrh na obrázku 5.7, zobrazuje zoznam chýb veľmi jednoducho. Nad zoznamom chýb sa nachádza tlačidlo, ktorým užívateľ môže vyvolať validáciu pre aktívny validátor. Každá chyba obsahuje svoj názov a pozíciu v kóde. Na chybu je možné kliknúť, čím sa zobrazí jej detailnejší popis aj s návrhom opravy, ktorý je možné vidieť na obrázku 5.8. Užívateľ vie následne kliknutím na tlačítko chybu opraviť.

Obr. 5.7: Zoznam chýb.

Obr. 5.8: Detail chyby po rozkliknutí.

5.5 Architektúra aplikácie

Návrh architektúry aplikácie najviac ovplyvnil výber technológie pre implementáciu. Angular rozdeľuje jednotlivé funkčné celky do komponentov a služieb. Preto som sa snažil aj ja celú funkcionalitu aplikácie rozdeliť do niekoľkých častí, ktoré neskôr budú reprezentovať tieto komponenty a služby.

Pri definovaní jednotlivých komponentov a ich zodpovedností som si za základ zobral návrh užívateľského rozhrania. Jednotlivé komponenty reprezentujú rôzne časti aplikácie. Napríklad zobrazovanie chýb a varovaní tvorí jeden komponent, detail chyby ďalší, editor je samostatný komponent a tak ďalej.

Komponenty medzi sebou potrebujú komunikovať určitým spôsobom. Framework Angular umožňuje niekoľko rôznych druhov komunikácie medzi komponentmi:

- predávanie dát z rodičovského komponentu na dieťa,
- predávanie dát z dieťaťa na rodičovský komponent,
- predávanie dát pomocou služby, ktorá definuje `Observable`.

Pri návrhu som navyše využil možnosť služby, ktorá uchováva spoločné dáta pre celú aplikáciu, ako napríklad obsah editora, aktuálne používaný validátor, druh vstupu a ďalšie.

Pri navrhovaní aplikácie je kľúčové vytvoriť dátové štruktúry, ktoré uchovávajú dáta. Vytvoril som preto celý rad štruktúr, ktoré uchovávajú výsledky výstupov z validátorov, záznamy o validovaných súboroch, štruktúry uchovávajúce nastavenia validácií a mnoho ďalších.

Celý návrh architektúry aplikácie rozoberiem v nasledovných riadkoch.

5.5.1 Návrh komponentov

Každý komponent zastupuje niektorú časť z návrhu užívateľského rozhrania. Môj návrh aplikácie obsahuje nasledovné komponenty a ich zodpovednosti.

`HeaderComponent` je základným komponentom, ktorý obsahuje všetky ďalšie komponenty. Pozostáva z mriežky, ktorá sa bude dynamicky meniť od vybraného spôsobu zadávania kódu.

`ValidatorsMenuComponent` umožňuje prepínať medzi jednotlivými druhmi validátorov (validátory pre jazyky HTML, CSS a JavaScript) a značí, ktorý validátor sa práve používa. Na tomto komponente sa vhodným spôsobom zobrazuje počet chýb, varovaní a prevedených opráv.

`SettingsComponent` umožňuje nastavenie rôznych možností validácie pre práve používaný validátor.

`SideBarComponent` zobrazuje bočné menu, pomocou ktorého si užívateľ vie nastaviť spôsob zadania vstupu do validátora. Má tri možnosti a to validácia nahraných súborov, validácia URL a validácia kódu zadaného do editora. V skorších návrhoch tento komponent má viacero možností okrem týchto troch. Medzi nimi bolo nastavenie možnosti validácie (premiestnil som ho ku komponentu `ValidatorsMenuComponent`) alebo vyhľadávanie medzi súbormi (z návrhu som ho nakoniec vypustil a nechal, ako možné budúce rozšírenie aplikácie). O ich vyradení z tohto komponentu som sa rozhodol na základe toho, že narušovali význam komponentu a to prepínanie medzi jednotlivými druhmi vstupu validátorov.

`EditorComponent` obsahuje hlavný editor.

`FileManagerComponent` má na starosti súborový strom, jeho vytváranie a manipuláciu s ním. Komponent umožňuje nahrávať súbory pre validáciu a stiahnuť opravené súbory.

`UrlComponent` obsahuje vstupnú lištu pre URL zdroja, ktorý užívateľ chce validovať.

`SpinnerWaitingComponent` zobrazuje sa pri načítavaní súborov alebo počas validácie. Komponent informuje užívateľa o tom, že prebieha načítavanie súborov alebo validácia. Komponent automaticky zmizne po dokončení danej operácie.

`ErrorBarComponent` má na starosti spustenie validácie zdrojov, v závislosti na práve vybranom druhu jazyka, alebo módu vstupu. Komponent zobrazuje celkový počet chýb, varovaní a opráv. Komponent ďalej zobrazuje užívateľovi zoznam chýb, varovaní a opráv s popisom chybovej správy a umiestnenia chyby v kóde. Chyby, varovania a opravy sú roztriedené podľa súborov, v ktorých sa nachádzajú. Komponent umožňuje po opravení chyby zahodenie opravy a vrátenie sa do stavu pred opravou, a taktiež obnovenie takto vrátenej chyby. Proces opravy chýb sa dá preto krokovať.

`ErrorComponent` zobrazuje samotnú chybu, alebo varovanie zvýraznené v kóde. V prípade, ak validátor umožňuje opravu danej chyby alebo varovania, zobrazí aj návrh jej opravy. Ak užívateľ súhlasí s návrhom opravy chyby alebo varovania, tak v takom prípade umožňuje užívateľovi chybu alebo varovanie opraviť.

`FixComponent` zobrazuje informáciu o oprave chyby. Komponent obsahuje spôsob akým chyba bola opravená a stav predchádzajúci oprave. Poskytuje užívateľovi možnosť vrátiť stav pred opravou chyby.

`TestComponent` tento komponent slúži pre testovanie validátora. Komponent obsahuje sadu kódov demonštrujúcich funkčnosť validátora a navrhovania chýb.

`Modálne okná` sú špeciálne komponenty, ktoré obsahujú varovanie a vyžadujú potvrdenie pre pokračovanie. Jedná sa hlavne o situácie, kedy sa mažú výsledky validácie a užívateľ je tak na túto skutočnosť upozornený a vyzvaný aby si zmeny stiahol, ak o ne nechce prísť. Sem patria situácie ako napríklad validácia novej URL adresy, alebo prepínanie medzi módmi vstupu validátorov.

5.5.2 Návrh služieb

Služby poskytujú funkcionalitu pre jednotlivé komponenty. Väčšinou sa jedná o funkcionalitu, ktorá je zdieľaná viacerými komponentmi. Pomocou služieb taktiež riešim komunikáciu medzi komponentmi a riadenie rôznych stavov aplikácie, aby každý komponent mal k nemu jednoduchý prístup a nemusel si tieto informácie získavať priamo z iných komponentov, čo by viedlo k zacykleniu závislostí a vysokej neprehľadnosti kódu. Môj návrh obsahuje následovné služby.

`ApiService` má na starosti implementáciu volaní webových služieb validátorov, backendu mojej aplikácie a získavanie zdrojov vzdialených stránok.

`FixHtmlService` obsahuje funkcie pre opravu chýb jazyka HTML. `FixCssService` obsahuje funkcie pre opravu chýb jazyka CSS. `FixJsService` obsahuje funkcie pre opravu chýb jazyka JavaScript.

`SpinnerWaitingService` implementuje funkcie potrebné pre zobrazenie komponentu `SpinnerWaitingComponent` v popredí aplikácie.

`ZipService` obsahuje funkcie slúžiace pre extrakciu komprimovaných súborov, ktoré sa následne používajú pri načítaní súborov typu zip.

`ComponentCommunicationService` slúži pre komunikáciu medzi jednotlivými komponentmi. Služba obsahuje rôzne stavy validátora alebo jednotlivých komponentov, napríklad práve vybraný mód vstupu validovaných dát, jazyk validátora, alebo aktuálny obsah editora, vybraný súbor v súborovom strome a ďalšie. Služba ponúka jednoduché rozhranie (set, get), pomocou ktorého jednotlivé komponenty môžu tieto stavy upravovať alebo čítať.

Služba taktiež definuje funkcie a premenné slúžiace pre komunikáciu medzi komponentmi pomocou funkcionalít jazyka JavaSript ako `Observable` a `subcribe()`.

5.5.3 Návrh dátových štruktúr

Počas validácie je potrebné si uchovávať množstvo dát a tie treba uložiť do nejakých rozumných štruktúr. Keďže vo frameworku Angular sa používa ako programovací jazyk TypeScript využil som výhodu statického typovania. Na definíciu štruktúr tak používam rozhrania.

Pre potrebu uchovávania výsledkov validácie som pre každý validátor vytvoril vlastnú dátovú štruktúru. Tieto štruktúry uchovávajú výsledok jednej validácie. Každá štruktúra má svoju hlavičku, ktorá obsahuje informáciu o tom, či validovaný kód je validný a zoznam chýb a varovaní. Zoznam chýb a varovaní sa líši v závislosti od použitého validátora, keďže rôzne validátory zasielajú rôzne štruktúry dát s rôznymi informáciami. Tieto dátové štruktúry som nazval `HtmlValidationRes`, `CSSValResult` a `JSValidationRes`.

Pri validovaní je potrebné zadať nastavenia možností validácie. Pre každý validátor som tak znova vytvoril samostatnú štruktúru: `HtmlOptions`, `CssOptions` a `JsOptions`. Tieto štruktúry obsahujú zoznam všetkých poskytovaných možností, ktoré si užívateľ môže nastaviť v aplikácií.

Pre validátor jazyka CSS som naviac vytvoril štruktúru `CSSRequest`, ktorá obsahuje informácie týkajúce sa vyvolania validácie. Štruktúra obsahuje kód na validáciu alebo URL, nastavenia a adresu webovej služby validátora.

Na vedenie záznamu o tom, ktoré chyby a varovania patria ku ktorému súboru som vytvoril dátovú štruktúru `ValidatedFile`. Tá obsahuje identifikátor, meno dokumentu a zoznam chýb a varovaní tvorený poľom dátových štruktúr `HtmlValidationRes`, `CSSValResult` a `JSValidationRes` v závislosti od druhu dokumentu.

Opravenú chybu značím v štruktúre `FixedError`. Táto dátová štruktúra obsahuje identifikátor súboru, kde bola chyba opravená, názov chyby, jej umiestnenie v kóde a obsah chyby, a nakoniec návrh opravy chyby.

Pre vrátené opravy chýb som navrhol štruktúru `RemovedFix`. Táto štruktúra slúži na zaznamenanie si opráv, ktoré boli užívateľom vrátené, aby sa dali neskôr obnoviť. Štruktúra obsahuje identifikátor súboru, ktorý bol opravovaný a chybovú hlášku.

Pri validácií pomocou URL je potrebné vytvárať súbory z validovaných zdrojov, aby si tieto zdroje potom užívateľ následne vedel opraviť. Pre vytváranie nových súborov som navrhol štruktúru `NewFile`, ktorá obsahuje všetky potrebné informácie. Medzi ne patrí identifikátor nového súboru, URL zdroja, názov súboru a obsah.

Navrhol som taktiež zopár pomocných štruktúr, ktoré budú pomáhať pri spustení validácií. Medzi nimi sú `ValidationSettings` a `ValidatorsErrorCount`. Štruktúra `ValidationSettings` obsahuje informácie o zdroji, ktorý sa bude validovať, či sa jedná o validáciu súborov, URL alebo obsahu editora. Následne obsahuje validovaný súbor, URL alebo kód z editora. Štruktúra `ValidatorsErrorCount` obsahuje počet chýb, varovaní a opráv pre jednotlivé validátory.

Nakoniec návrh obsahuje posledné dve štruktúry: `ZipTaskProgress` a `ZipTask`, ktoré boli prebrané z tohto zdroja [13]. Tieto štruktúry majú svoje využitie pri extrahovaní obsahu súboru s príponou zip.

5.5.4 Návrh backendu

Backend komplexného validátora slúži na pokrytie funkcionalít, ktoré nie je možné z rôznych príčin (technických alebo návrhových) implementovať vo frontende. Backend nezahrňuje

nijakú prácu s databázou, keďže si moja aplikácia nepotrebuje ukladať perzistentné dáta. Dáta o validácií existujú len počas samotnej validácie a opravy chýb. Užívateľ má následne možnosť si opravené súbory stiahnuť.

Backend aplikácie je použitý hlavne pre sťahovanie súborov užívateľom. Umožňuje užívateľovi stiahnuť všetky súbory a komprimuje ich do archívu zip. Okrem stiahnutia súborov umožňuje taktiež stiahnuť diff/patch, ktoré zobrazujú rozdiel medzi pôvodným a výsledným dokumentom.

5.6 Návrh zoznamu opravovaných chýb

Súčasťou práce je v prípade chybného kódu poskytnúť aj návrh opravy chyby. V rámci tejto práce nie je možné navrhnúť opravy pre všetky druhy chýb. Preto po konzultácií s vedúcim práce som usúdil, že moja aplikácia umožní opraviť následovné chyby.

5.6.1 Opravované chyby pre jazyk HTML

V prípade jazyka HTML by moja aplikácia mohla byť schopná opraviť chybu týkajúcu sa chýbajúceho typu súboru `<!DOCTYPE html>` na začiatku HTML dokumentu.

Ďalej by to mohlo byť opravenie chyby týkajúcej sa zlého uzavretia elementu jazyka HTML. Na jednej strane by to mohlo byť v prípade chýbajúcej otváracej/uzatváracej značky, napríklad ako v tomto príklade 5.9. V takomto prípade by bolo potrebné spočítať počet značiek, zistiť ktorá z nich chýba a určiť približné miesto, kde môže chýbať. Na druhej strane by to mohlo byť opravenie chybnej uzatváracej značky ako je zobrazené na obrázku 5.10.

```
<div>
    <p>Chýbajúca uzatváracia značka
</div>
```

Obr. 5.9: Chýbajúca uzatváracia značka.

```
<div>
    <span>Chýbajúce lomítko v uzatváracej značke<span>
</div>
```

Obr. 5.10: Nesprávne uzatvorená značka.

Zaujímavou opravou by mohla byť oprava chybného názvu značky. V rámci práce by som sa zameral na malú chybu v názve značky (jedno alebo dve písmená, záviselo by to od dĺžky názvu značky). V takomto prípade by bolo potrebné získať celý zoznam značiek jazyka HTML a pomocou vhodného algoritmu určiť najlepšiu zhodu.

Ďalšou chybou by mohlo byť opravenie chybného formátu komentára, napríklad ako je ukázané na obrázku 5.11.

Pri programovaní sa mi stalo, že som si zamenil poradie zanorenia elementov v zozname, ako je zobrazené na príklade 5.12. Moja aplikácia by opravila toto zlé poradie. Táto oprava by sa následne dala využiť aj na zlé zanorenia v iných elementoch.

```
<!chyba- Chybne otvorený komentár -->
```

Obr. 5.11: Chybný formát komentáru.

```
<ul>
    <a><li>Položka 1, chybné zanorenie</li></a>
    <li>Položka 2</li>
    <li>Položka 3</li>
</ul>
```

Obr. 5.12: Chybné zanorenie elementov v zozname, značka `<li>` musí byť na vonkajšej strane.

Veľmi ľahkými chybami na opravu môže byť hlásenie varovania o neuvedení názvu jazyka v elemente `<html>`, ako napríklad `<html lang=ën»`, chýbajúci titulok stránky uvedený HTML elementom `<title>`, vymazanie nepovinného atribútu `<type>` na javascriptových zdrojoch, vymazanie nepovolených alebo duplicitných atribútov na elementoch.

Ďalšou chybou na opravu môže byť oprava chybne ukončenej hodnoty atribútu spôsobenej chýbajúcou úvodzovkou, napríklad `<a href=ädresa>`.

V prípade obrázkov môže byť aplikácia schopná doplniť chýbajúce atribúty `src` a `alt`. Pravdaže boli by doplnené ako prázdne, zdroj a popis by si užívateľ následne dopísal sám.

5.6.2 Opravované chyby pre jazyk CSS

Pri jazyku CSS moja aplikácia môže byť schopná opraviť základné syntaktické chyby, ako napríklad chýbajúce zložené zátvorky, dvojbodky alebo bodkočiarky.

Náročnejšia oprava môže spočívať v oprave chyby v názve vlastnosti. Napríklad ak sa užívateľ pomýli a napíše namiesto správneho názvu vlastnosti `height` chybný názov `heigt`. Na tento problém bude možné využiť podobný algoritmus ako pri oprave chybného názvu HTML elementu, alebo priamo chybovú správu validátora, ktorá obsahuje správny návrh názvu vlastnosti. V tom prípade by stačilo správny názov len vytiahnuť z chybovej správy.

Ďalšou podporovanou opravou by mohla byť oprava chybných jednotiek (px, rem, vh) alebo názvov farieb (red, blue a ďalšie).

Veľa chýb v jazyku CSS je závislých od sémantiky, toho, čo užívateľ zamýšľa. Validátor môže napríklad nahlásiť chybu, že konkrétna hodnota sa nenachádza medzi možnými hodnotami danej vlastnosti. Takúto chybu ale nie som schopný opraviť, keďže neviem zistiť, ako nejaký element užívateľ chcel zobraziť.

Uvažoval som ešte nad tým, že v prípade dokumentu v jazyku HTML, by som mohol byť schopný opraviť chybný názov selektoru. Bolo by to možné, ak by som si porovnal HTML elementy, triedy a identifikátory v HTML dokumente s tými v štýlovom predpise, a podľa toho by som to bol schopný opraviť. CSS validátor ale nevyhodnocuje chybné názvy selektorov, čo je pochopiteľné.

5.6.3 Opravované chyby pre jazyk JavaScript

Pri jazyku JavaSript bolo najťažšie navrhnúť nejaké opravy chýb. Je to z toho dôvodu, že veľmi veľa vecí spadá do sémantiky jazyka a rôzne návrhy opráv môžu len slepo hádať, čo bolo pôvodným cieľom kódu autora. Podarilo sa mi nájsť zopár chýb alebo varovaní, ktoré by moja aplikácia mohla byť schopná riešiť.

Asi najzákladnejším je pridanie chýbajúcich bodkočiarok. Jazyk JavaSript však nevyžaduje povinné bodkočiarky na konci príkazov, preto toto bude len riešenie varovaní.

Zaujímavejšou možnosťou je pridávanie chýbajúcich deklarácií premenných. Najjednoduchšie by sa to dalo implementovať tým spôsobom, že by sa deklarácia pridávala hneď pred prvé použitie premennej.

Kapitola 6

Implementácia webovej aplikácie

V tejto kapitole opisujem implementáciu komplexného validátora. V jednotlivých podkapitolách vysvetlím princíp implementácie jednotlivých častí a dôvod, prečo som sa preň rozhodol. Ukážem hlavne tie zaujímavejšie časti implementácie.

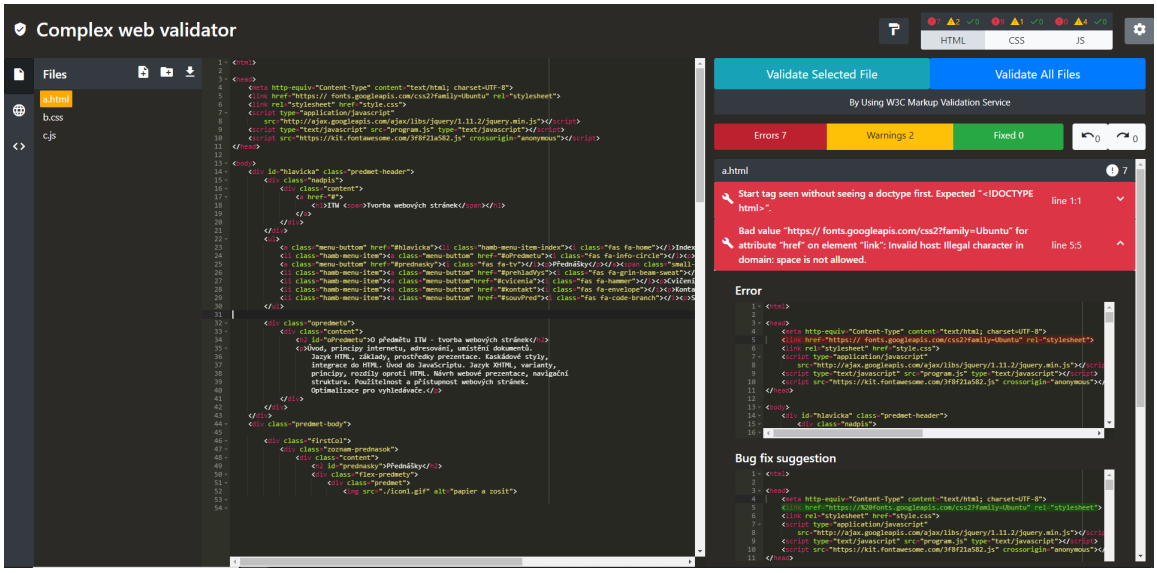
Na začiatku tejto kapitoly popíšem implementáciu grafického rozhrania a to ako sa líši od návrhu. Ďalej vysvetlím aký som použil editor a ako je implementovaný. Následne sa pozriem na proces validácie. Ukážem, ako používam jednotlivé aplikačné rozhrania validátorov. Vysvetlím proces, ktorý sa nachádza za validáciou súborov, URL a obsahu editora. Následne popíšem, ktoré druhy chýb validátor umožňuje opraviť a spôsob akým to robím. Popíšem aj implementáciu krokovania chýb. Ďalej objasním ako pracujem v aplikácií so súbormi: spôsob ich načítania, uloženia v súborovom strome, sťahovanie vzdialených webových stránok a implementáciu sťahovania validovaných a opravených súborov užívateľom.

6.1 Implementácia užívateľského rozhrania

Rozdiel medzi návrhom a implementáciou užívateľského rozhrania je veľký. V tejto časti by som zhrnul všetky zmeny oproti návrhu a ukázal ako vyzerá jeho implementácia.

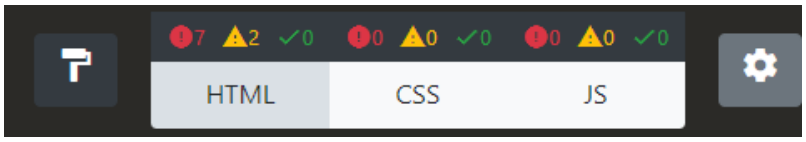
Implementácia užívateľského rozhrania nezahŕňa úvodnú obrazovku z návrhu. Usúdil som, že táto úvodná stránka neposkytuje užívateľovi žiadnu pridanú hodnotu. Neposkytuje žiadnu zvláštnu funkcionalitu, ktorú by nenašiel na stránke samotného validátora. Tento návrh naviac neumožňuje zadanie kódu rovno do editora.

Užívateľské rozhranie je zobrazené na obrázku 6.1. Hlavné rozloženie aplikácie kopíruje bez nejakých väčších zmien pôvodný návrh. Je zachovaný základný tmavý štýl. Niektoré časti aplikácie sú farebnejšie oproti návrhu.

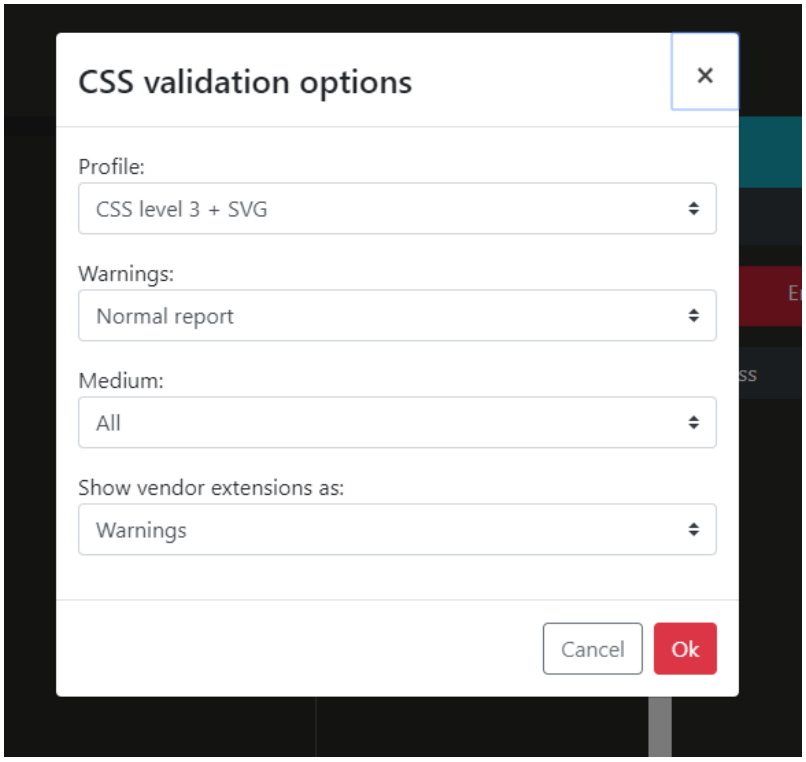


Obr. 6.1: Užívateľské rozhranie aplikácie.

Napravo, v hornej lište sa nachádza prepínač medzi validátormi, podobne ako v návrhu (viz. obrázok 6.2). Prepínač oproti návrhu navyše obsahuje počet chýb, varovaní a opráv pre konkrétny validátor a nástroj na formátovanie kódu. Napravo od prepínača sa nachádza ikonka pre nastavenia. Po jej stlačení sa ukáže okno s obsahom nastavení pre práve používaný validátor (viz. obrázok 6.3).



Obr. 6.2: Prepínač validátorov s počtom chýb a nastaveniami a nástrojom na formátovanie kódu.



Obr. 6.3: Nastavenia validácie pre jazyk CSS.

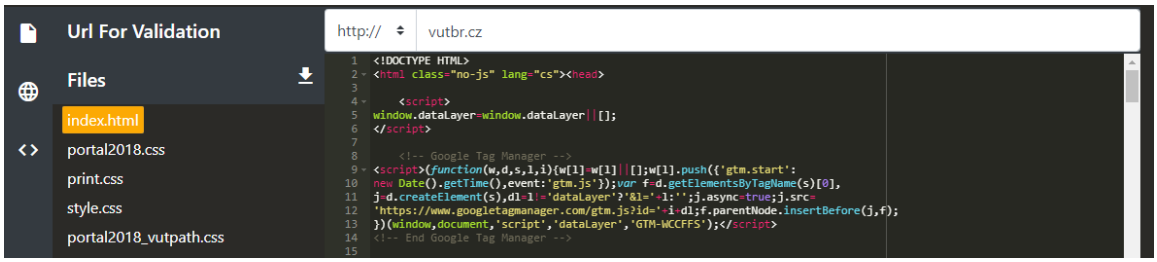
Ľavá lišta obsahuje na rozdiel od návrhu len tri ikonky. Každá je pre jeden druh vstupu validátora: súbory, URL a obsah editora. Pri podržaní myši na danom tlačítku vyskočí názov funkcionality.

Vyhľadávanie medzi súbormi, ktoré obsahoval pôvocný návrh, som sa rozhodol neriešiť v tejto práci. Je to ale zaujímavá funkcionalita, ktorá sa môže implementovať pri ďalšom vývoji aplikácie. Nastavenia som presunul do hornej lišty vedľa prepínača validátorov, pretože sa týkajú samotných validátorov.

Užívateľské rozhranie pre validáciu načítaných súborov je veľmi podobné návrhu. Naviac obsahuje rozdelenie tlačidiel pre načítanie súborov, zip archívu a stiahnutie opravených súborov, ako je možné vidieť na obrázku 6.4.

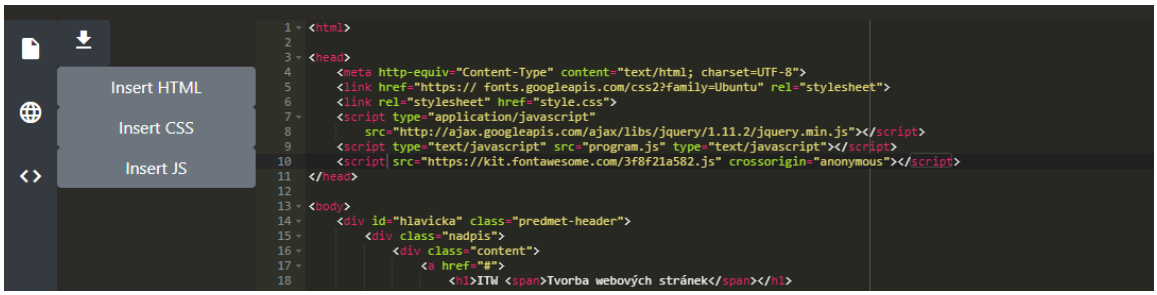
Obr. 6.4: Validácia načítaných súborov.

Implementácia GUI je tiež veľmi podobná návrhu. Oproti návrhu naviac obsahuje súborový strom, so zoznamom chybných zdrojov. Užívateľ si môže vybrať, v akom formáte chce vložiť adresu URL: s predponou http, https alebo bez predpony (viz. obrázok 6.5).



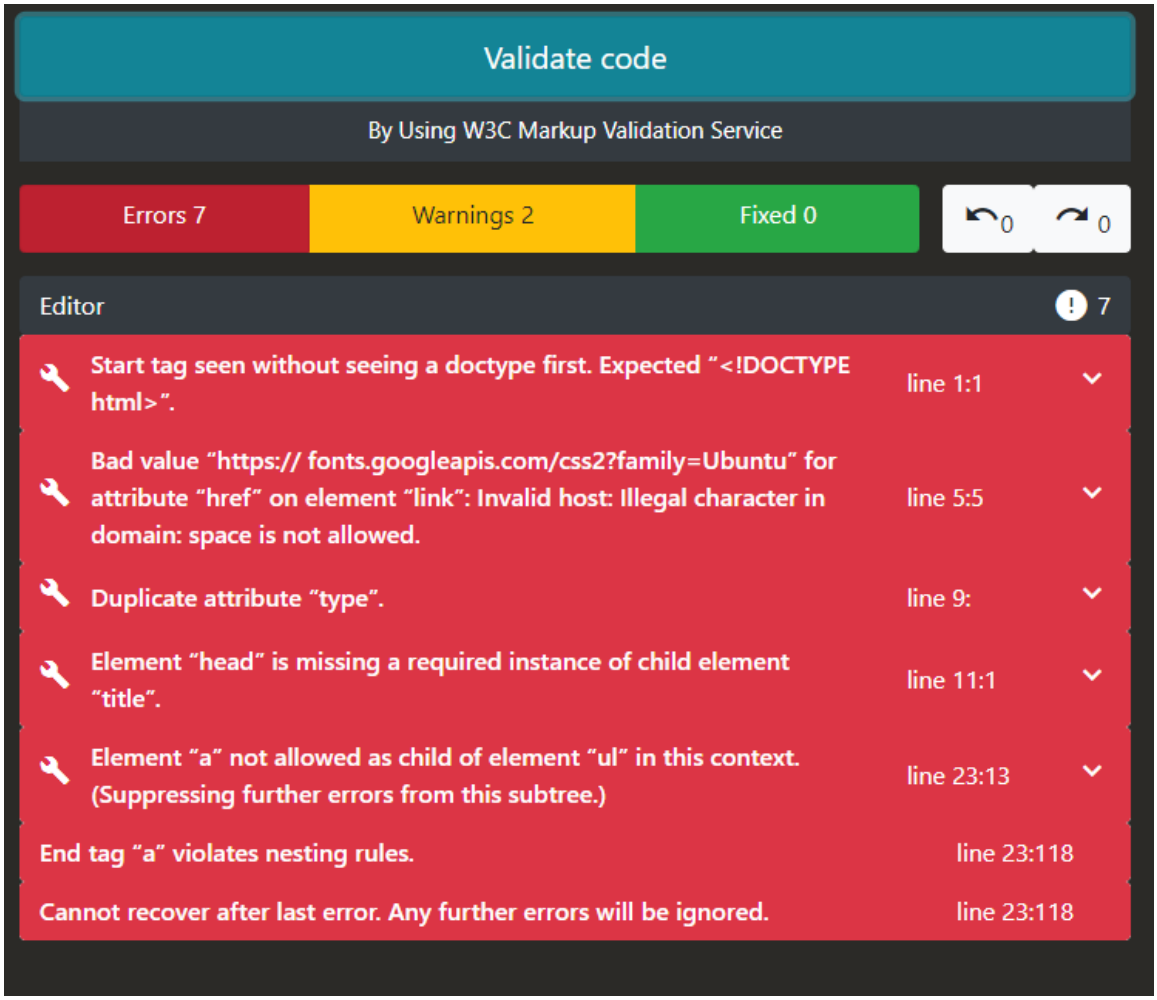
Obr. 6.5: Validácia adresy URL.

Pri validácií obsahu editora sa naľavo od editora nachádzajú tlačítka pre zobrazenie kódu pre testovanie, ktoré som ponechal aj pre oponenta. Pomocou nich je možné v editore zobraziť kód, ktorý demonštruje rôzne návrhy opráv chýb, ktoré aplikácia implementuje. Tieto tlačítka sa v budúcnosti v aplikácií nebudú vyskytovať. Nachádza sa tu taktiež tlačidlo pre stiahnutie validovaného kódu (viz. obrázok 6.6).



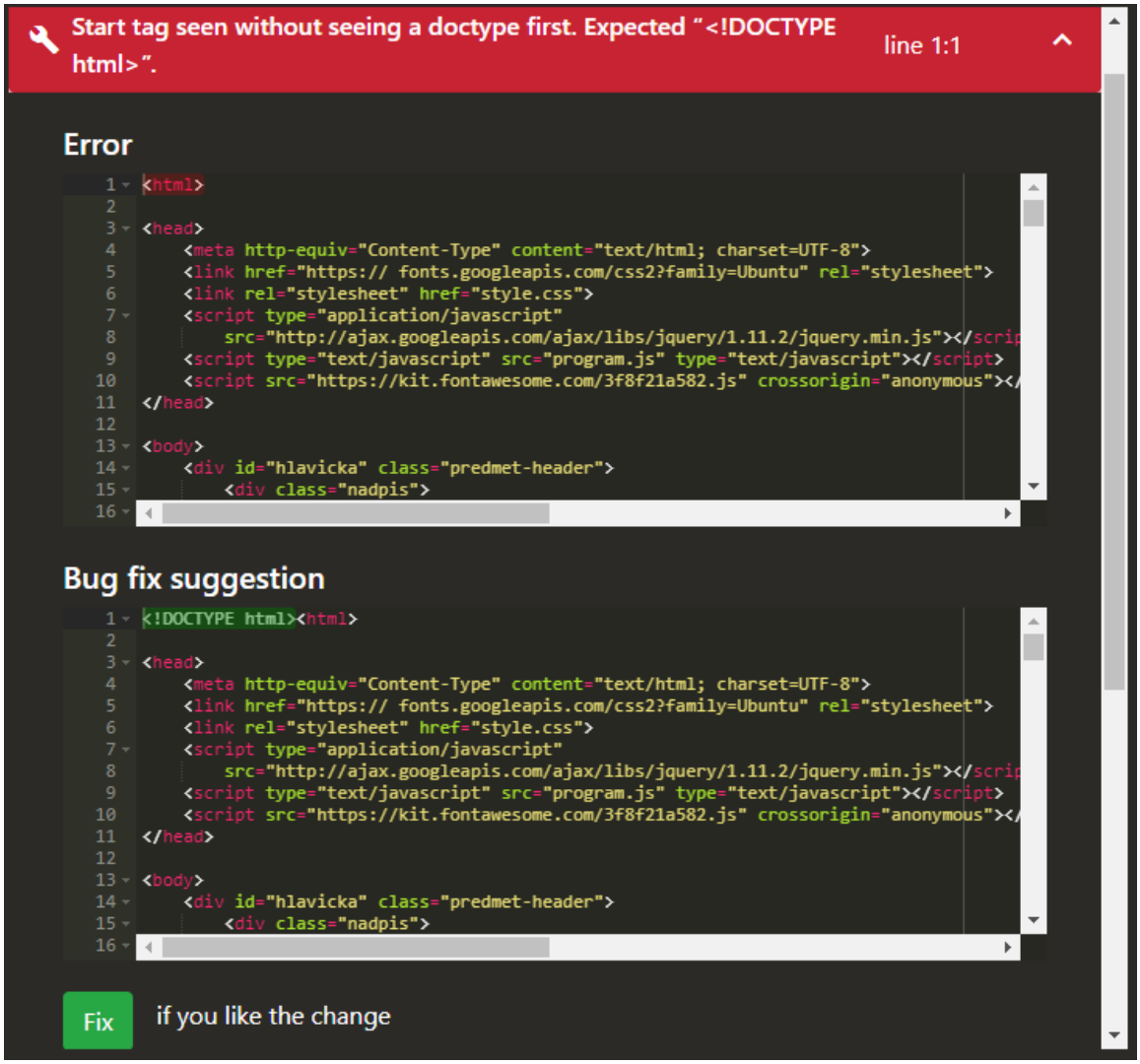
Obr. 6.6: Validácia obsahu editora aj s ponukou kódov na testovanie.

Na pravej strane sa nachádzajú tlačítka pre spustenie rôznych druhov validácie aj s názvom používaného validátora, zoznam chýb, varovaní a opráv ako je možné vidieť na obrázku 6.7. Jednotlivé chyby a varovania sú rozdelené podľa súborov, ku ktorým patria. Vedľa názvu súboru sa nachádza ich počet. Po rozkliknutí sa zobrazí zoznam chýb alebo varovaní. Tak ako návrh, zobrazuje názov chyby aj s jej pozíciou v kóde. Vedľa prepínača medzi chybami, varovaniami a opravami sa nachádzajú dve šípky, umožňujúce krokovanie opravených chýb. Chyby ktoré sa dajú opraviť sú označné príslušnou ikonkou.



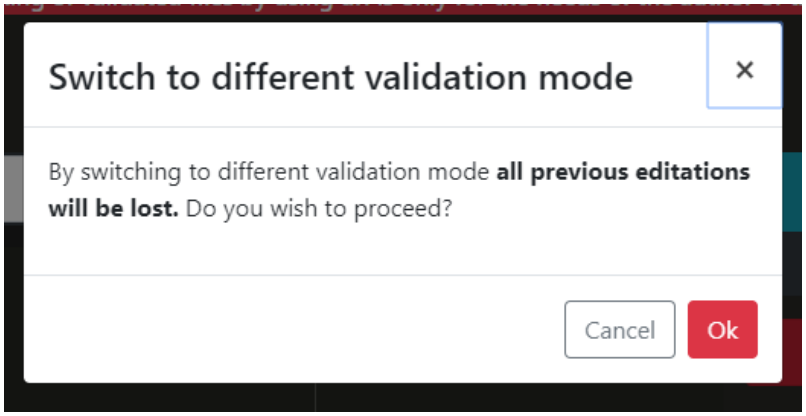
Obr. 6.7: Zoznam chýb, varovaní a opráv.

Po rozkliknutí chyby, varovania alebo opravy sa zobrazí detail chyby. Detail chyby je zobrazený na obrázku 6.8. Ten obsahuje chybu zvýraznenú v kóde a taktiež návrh opravy. Ak je návrh opravy k dispozícií, užívateľ ho vie potvrdiť pomocou tlačítka *Fix*. Oprava chyby obsahuje tie isté editory, ale v opačnom poradí s tlačítkom pre vrátenie opravy chyby.



Obr. 6.8: Detail chyby.

Pri prepínaní medzi módmi vstupu validácie alebo pri validácií novej URL, užívateľa o pokračovaní informujú modálne okná, ktoré vyžadujú potvrdenie. Väčšinou sú použité v prípadoch, keď aplikácia maže predchádzajúce výsledky validácie, aby užívateľ bol o tejto skutočnosti informovaný a mohol si uložiť doterajší progres validácie a opráv. Jeden príklad takého modálneho okna môžete vidieť na obrázku 6.9.



Obr. 6.9: Modálne okno informujúce užívateľa o vymazaní predchádzajúcej validácie.

Aby vedel rodič predávať dáta dieťaťu alebo naopak, je potrebné v komponente potomka definovať vstupy a výstupy. Ako je zobrazené na obrázku 6.10, je to možné urobiť pomocou dekorátorov `@Input()` pre vstupy a `@Output()` pre výstupy komponentu. Pomocou vstupu do komponentu vie rodič posielať dáta svojmu potomkovi a pomocou výstupov vie potomok zasielať dáta svojmu rodičovi.

6.2 Komunikácia medzi komponentmi

Komunikáciu medzi komponentmi zabezpečuje služba `ComponentCommunicationService`. Túto službu využívam pre komunikáciu medzi komponentmi, ktoré nie sú vo vzťahu rodič-dieťa. Pre komunikáciu medzi rodičom a dieťaťom používam rozhranie, ktoré poskytuje framework Angular. V takomto vzťahu sa nachádzajú komponenty `ErrorBarComponent` a `ErrorComponent`. Komponent `ErrorBarComponent` je rodič a komponent `ErrorComponent` je dieťa.

```
@Input() validatedFileContent: string;
@Input() isCollapsed: boolean;

@Output() fixedError = new EventEmitter<FixedError>();
```

Obr. 6.10: Úsek kódu komponentu potomka.

Pri zasielaní dát potomkovi je potom potrebné už len na rodičovi v elemente potomka pridať potrebný atribút (viz. obrázok 6.11) a ako jeho hodnotu priradiť zdroj dát (môže to byť premenná alebo funkcia, ktorá vracia tieto dáta), ktoré sme si definovali na komponente potomka. Dáta sa automaticky uložia na komponente potomka.

Pri zasielaní dát z potomka na rodiča je potrebné najprv dáta odoslať. Toto odoslanie sa následne zaregistruje na rodičovi. Detekuje sa znova pomocou atribútu na elemente potomka v rodičovskom elemente (viz. obrázok 6.11). Výsledok je potom potrebné spracovať pomocou funkcie, ktorá sa zavolá automaticky pri prijatí dát.

```
<app-error [htmlErrorMsg]="error"
    [validatedFileContent]="getValidatedFileContent(file.id)"
    [isCollapsed]="error.isCollapsed" (fixedError)="onFixError($event)">
</app-error>
```

Obr. 6.11: Úsek kódu rodičovského komponentu.

6.3 Editor

Editor je veľmi dôležitou časťou môjho programu. Umožňuje užívateľovi počas validácie upravovať súbory. Užívateľ si tak môže veľmi rýchlo a jednoducho opraviť chyby a nemusí napríklad pri validácií súborov alebo URL najprv opraviť zdroje u seba a následne ich znova nahrať pre validáciu.

Je preto dôležité, aby editor spĺňal niektoré základné požiadavky. Mal by umožňovať zvýrazňovať syntax pre jazyky, s ktorými užívateľ pracuje počas validácie. Veľmi výhodné je, aby podporoval viacero štýlov a tém, aby som ho vedel zladiť s dizajnom aplikácie. Editor by mal byť po implementačnej stránke prispôsobivý. Editor musí poskytovať možnosť vkladať, upravovať a odstraňovať obsah priamo z kódu. Editor musí byť použiteľný s frameworkom Angular. Nakoniec by mal umožňovať zvýrazňovanie častí kódu. Tieto všetky moje požiadavky spĺňal `Ace Editor`[1], ktorý využívam vo svojej práci.

Tento editor využívam v dvoch častiach aplikácie: hlavný editor a menšie editory na komponente detailu chyby/opravy pre zobrazenie chyby a návrhu opravy chýb.

Hlavný editor je spravovaný pomocou komponentu `EditorComponent`. Tento komponent inicializuje hlavný editor. Po inicializácii sa odkaz na novo vytvorený editor ukladá na službu `ComponentCommunicationService`. Pomocou tejto služby je následne možné pristupovať k hlavnému editoru z hociktorého iného komponentu, prispôsobovať ho alebo upravovať jeho obsah. Týmto spôsobom je potom možné meniť napríklad aktuálny jazyk pre editor, ktorý následne umožňuje zvýrazňovanie syntaxu pre daný jazyk, z komponentu `ValidatorsMenuComponent`, ktorý slúži ako prepínač jazykov.

Menšie editory sa nachádzajú na komponentoch `ErrorComponent` a `FixComponent`. S týmito editormi pracujem výhradne len v týchto komponentoch, preto nie sú uložené nikde inde. Obsah týchto editorov užívateľ nevie upravovať. V budúcnosti by aplikácia mohla obsahovať možnosť povoliť úpravu pre editor zobrazujúci návrh opravy chyby a takto užívateľom zadanú opravu aplikovať. Túto opravu chyby by následne aplikácia pri rovnakej chybe už sama odporučila užívateľovi. Aplikácia by sa takto sama učila opravovať nové chyby.

Pri inicializácii týchto editorov je potrebné vedieť, ktorý validátor sa práve používa, aby sa nastavil správny jazyk. Aktuálne používaný validátor si komponenty pri inicializácii editorov získava znova pomocou služby `ComponentCommunicationService`.

Súčasťou editora je nástroj pre úpravu kódu (takzvaný beautify tool), ktorý v aplikácií používam. Jeho funkcionalita sa dá spustiť pomocou príslušného tlačidla v aplikácií.

6.4 Validácia

Validácia súborov je implementovaná v komponente `ErrorBarComponent` a službe `ApiService`. Komponent `ErrorBarComponent` pripravuje všetky potrebné veci pred validá-

[1] https://ace.c9.io/

ciou, po validácii a spúšťa ju. Komponent nastavuje aktuálne používaný súbor pre validáciu, získava možnosti validácie a rozpoznáva druh vstupu validovaných súborov a druh validátora, ktorý sa používa. Komponent zabezpečuje správne spracovanie výsledkov po validácií, ich priradenie k správnym súborov a ukladanie výsledkov do správnych dátových štruktúr. Ak je to potrebné, tak na konci spúšťa vytváranie nových súborov, ktoré má na starosti komponent `FileManagerComponent`.

Služba `ApiService` implementuje volanie webových služieb jednotlivých validátorov, ktoré sa následne volajú v komponente `ErrorBarComponent`.

V následujúcich sekciách by som proces validácie opísal podrobnejšie.

6.4.1 Práca s aplikačným rozhraním jednotlivých validátorov

V aplikácií používam tri validátory: W3C Markup Validation Service pre jazyk HTML, W3C CSS Validation Service pre jazyk CSS a JSHint pre jazyk JavaSript. Každý validátor má rozličné aplikačné rozhranie, ktoré som popísal v kapitole 4. To si vyžaduje aj rozličnú implementáciu.

Pre volanie webových služieb validátorov používam Angular knižnicu jednoduchého HTTP klienta, ktorého som spomenul v časti 3.4.1.

Ako bolo už vyššie spomenuté, implementácia volania služieb validátorov sa nachádza na službe `ApiService`.

Validátor jazyka HTML

Volanie webových služieb validátora W3C Markup Validation Service je implementované pomocou funkcie `validateHtml`. Vstupom funkcie sú nastavenia spôsobu validácie uložené v štruktúre `ValidationSettings`. Funkcia vracia výsledok validácie v dátovej štruktúre `HtmlValidationRes`. Jedná sa o asynchrónnu funkciu.

Vo funkcii sa podľa vstupu, ktorý sa má validovať (táto informácia je uložená v štruktúre `ValidationSettings`), vyberie adekvátne volanie webovej služby validátora. Pre validáciu súborov alebo vstupu editora využívam HTTP metódu POST. Obsah súboru alebo editora sa nachádza v štruktúre `ValidationSettings`. Pri validácií dokumentu pomocou URL používam metódu protokolu HTTP GET. Adresa URL sa nachádza v rovnakej štruktúre ako obsah súboru, či editoru. K adrese webovej služby je nutné v takomto prípade pridať parameter `doc` s hodnotou URL validovaného dokumentu.

Webová služba sa nachádza na adrese `https://validator.w3.org/nu/`. K tejto adrese pri každom volaní pripájam parameter `out` s hodnotou `json`, čo zaručuje, že výsledok validácie dostanem v tomto formáte. K adrese pripájam ešte parameter `level`, ktorého hodnotu si vie užívateľ nastaviť v nastavení spôsobu validácie. Tento parameter určuje, či sa užívateľovi zobrazia len chyby, alebo aj varovania.

Po prijatí výsledku validácie ho spracujem (hlavne oddelím chyby od varovaní) a vložím do novo vytvorenej štruktúry `HtmlValidationRes`.

Pri prvej validácií sa stáva, že vzdialená služba má dlhú odozvu. Nezistil som čím to môže byť spôsobené, chyba sa pravdepodobne nachádza na strane serveru.

Validátor jazyka CSS

Volanie webových služieb validátora W3C CSS Validation Service je implementované pomocou funkcie `validateCss`. Vstupom funkcie sú nastavenia spôsobu validácie uložené

v štruktúre `ValidationSettings`. Funckia vracia výsledok validácie v dátovej štruktúre `CSSValResult`. Jedná sa o asynchrónnu funkciu.

Pri validácií jazyka CSS používam na validáciu URL webovú službu validátora. Služba sa nachádza na adrese `http://jigsaw.w3.org/css-validator/validator`. K adrese webovej služby je nutné v takomto prípade pridať parameter `url` s hodnotou URL validovaného dokumentu. Navyše pri každom volaní služby k tejto adrese pripájam parameter `output` s hodnotou `soap12`, čo zaručuje, že výsledok validácie dostanem v tomto formáte. Nanešťastie tento validátor neposkytuje lepšie a novšie druhy formátov ako napríklad json. Spracovanie výsledkov bolo preto pracnejšie ako v prípade ostatných validátorov.

Pre validáciu súborov a obsahu editora používam nástroj `css-validator`[2], ktorý pracuje práve s validátorom W3C CSS Validation Service. Tento nástroj poskytuje veľmi jednoduché rozhranie. Nástroj obsahuje jednoduchú štruktúru, do ktorej sa dajú zadať vstupy a jednotlivé nastavenia validátora a výstup je vo formáte json. Tento nástroj beží v backende mojej aplikácie a túto službu volám pomocou metódy POST protokolu HTTP. Pre tento nástroj som sa rozhodol práve kvôli jeho jednoduchému použitiu a kvôli tomu, že som mal pri začiatku implementácie problémy s volaním služby W3C CSS Validation Service v prípade validácie súborov a obsahu editora.

Nanešťastie nástroj `css-validator` neidentifikuje, ktorá chyba patrí ku ktorému zdroju. Pri validácií súborov a obsahu editora to nie je problém, keďže pre každý vstup si volám službu validátora zvlášť. Pri validácií URL nastáva ale problém. Zo začiatku som používal len nástroj `css-validator`. Až keď som potreboval triediť chyby podľa súborov, všimol som si tento nedostatok a tak som pre validáciu URL začal používať priamo webovú službu validátora W3C CSS Validation Service, ktorá radí chyby k zdrojom, v ktorých sa vyskytujú. Následne som implementoval spracovanie výsledku vo formáte soap12, do štruktúry `CSSValResult`.

Validátor jazyka JavaScript

Validátor JSHint na rozdiel od predchádzajúcich validátorov nie je webovou službou. Používam ho ako balíček platformy Node.js. Používanie funkcionalít tohto validátora je implementované pomocou funkcie `validateJs`. Vstupom funkcie sú, ako pri ostatných validátoroch, nastavenia spôsobu validácie uložené v štruktúre `ValidationSettings`. Funkcia vracia výsledok validácie v dátovej štruktúre `JSValidationRes`. Používanie validátora si nevyžaduje asynchrónne volania, čo je veľká výhoda práve tohto validátora.

Aplikačné rozhranie som podrobne opísal v časti 4.3. Pre volanie validátora používam funkciu `JSHINT`. Výsledok validácie už veľmi neupravujem. Hlásenia rozdelím akurát na chyby a varovania a uložím do štruktúry `JSValidationRes`.

Nastavenie možnosti validácie

Užívateľ si môže vybrať rôzne nastavenia validácie. Aplikácia zatiaľ povoľuje následovné nastavenia.

Nastavenia pre validátor jazyka HTML:

- zobrazovanie varovaní.

Nastavenia pre validátor jazyka CSS:

[2] `https://github.com/twolfson/css-validator`

- nastavenie profilu, verzia jazyka CSS,
- zobrazovanie varovaní,
- druh zobrazovacieho média,
- ignorovanie rozšírení jazyka CSS.

Nastavenia pre validátor jazyka JavaScript:

- verzia štandardu,
- používanie premenných pred definíciou,
- zatieňovanie premenných,
- povolenie ES5 striktného módu,
- používanie nedefinovaných premenných,
- hlásenie chyby pri nepoužitých premenných,
- zakázanie používania deklarácií premenných pomocou kľúčového slova `var`.

Pri ďalšom rozširovaní aplikácie, by sa mohli pridať nové nastavenia.

Zobrazenie možností aplikácie je implementované na komponente `ValidatorSettingsModalComponent`. Po ich potvrdení uživateľom sa ukladajú na spoločnú službu pre všetky komponenty `ComponentCommunicationService`. Pred každou validáciou sa tieto nastavenia uložia do štruktúry `ValidationSettings`, s ktorou potom pracujú funkcie implementujúce volanie validátorov v rámci služby `ApiService`.

6.4.2 Validácia súborov

Validáciu súborov je potrebné rozdeliť na dve hlavné časti a to validácia všetkých súborov a validácia vybraného súboru. Pri oboch týchto prípadoch sa musí užívateľ nachádzať v móde validácie súborov. Tento mód si vie vybrať v menu na ľavej strane aplikácie. Užívateľ si musí súbory najprv nahrať. Pokiaľ si ich nenačíta, nevie pracovať s editorom a validácia sa nespustí, resp. sa spustí validácia prázdneho súboru. Po úspešnom spustení validácie sa zobrazí obrazovka informujúca užívateľa že súbory sa validujú. Táto obrazovka zmizne hneď ako sa validácia ukončí.

Validovanie všetkých súborov implementuje funkcia `validateAllFiles` na komponente `ErrorBarComponent`. Pred samotnou validáciou funkcia vymaže výsledky predchádzajúcej validácie, nastaví príznak, značiaci, že sa bude jednať o validáciu všetkých súborov a uloží aktuálny obsah editora. Následne sa spustí validácia celého súborového stromu. Algoritmus prejde cez celý súborový strom, identifikuje súbory a podľa ich koncovky spustí príslušný validátor. Podporované koncovky súborov sú: html, htm, dhtml, phtml, jhtml, mhtml, rhtml, shtml, zhtml, css a js. Súbory, ktoré nemajú požadovanú koncovku sú uložené do zvláštnej štruktúry.

Pri validácií vybraného súboru sa v závislosti na zvolenom validátore spúšťajú funkcie `validateSelectedHtml`, `validateSelectedCss` alebo `validateSelectedJs`. Tieto funkcie majú podobnú funkciu a to pripraviť všetko potrebné pred samotnou validáciou. Funkcie využívam aj pri validácií URL alebo obsahu editora. Táto prípravná fáza najprv zabezpečuje

aby v štruktúrach, ktoré sa budú používať, neboli žiadne nadbytočné alebo zlé dáta. Tieto štruktúry sa preto na začiatku prečistia, aby validácia prebehla bez ťažkostí. Ďalej sa určí, ktorý súbor bude validovaný. Buď sa jedná o nový súbor alebo v prípade validácie URL sa môže jednať o nový súbor, ktorý ešte nie je vytvorený. Po tejto príprave sa spustí validácia.

6.4.3 Validácia URL

Validácia URL je asi najkomplikovanejší spôsob validácie. Pri tomto spôsobe validácie je potrebné rozlišovať medzi prvou validáciou novej URL a ďalšími validáciami toho istého zdroja. Ďalej podobne ako pri validácií súborov je potrebné rozlišovať medzi validáciou všetkých zdrojov alebo len častí zdrojov (napríklad validácia len štýlových predpisov).

Kvôli tomuto som tento spôsob validácie v aplikácií rozdelil na dve časti: prvá validácia adresy a následné validácie. Pri prvej validácií sa validuje priamo URL adresa. Túto adresu vkladám ako parameter do validátorov. Validátory následne vrátia chybné súbory, ktoré stiahnem a uložím do súborového stromu. Následujúce validácie tej istej URL adresy pracujú práve s týmto vytvoreným súborovým stromom a používajú sa rovnaké funkcie ako v prípade validácie súborov. Ak užívateľ chce nanovo validovať zdroje na tej istej adrese alebo validovať novú adresu, tak môže validátor resetovať pomocou určeného tlačítka ("New Urlümiesteného v pravom paneli").

Validácia všetkých druhov zdrojov (HTML kód, štýlové predpisy a skripty) je implementovaná vo funkcii `validateAllUrl`. Pred samotnou validáciou všetkých druhov zdrojov sa nastaví príznak pre validáciu všetkých URL zdrojov, príznak, že sa jedná o prvú validáciu skriptov a príznak na zabránenie vytváraniu viacerých súborov s názvom `index.html`. Následne sa spustia prípravy pre jednotlivé druhy validácií pomocou už vyššie spomenutých funkcií `validateSelectedHtml`, `validateSelectedCss` alebo `validateSelectedJs`.

Po validácií sa zoberú validované zdroje pre daný validátor a začne sa príprava pre vytváranie súborov z týchto zdrojov. Aplikácia pripraví pre každý zdroj novú štruktúru `NewFile`. Do štuktúry sa uloží nový identifikátor súboru, adresa zdroja, meno súboru a získa sa jeho obsah. List týchto štruktúr sa následne predá komponentu `FileManagerComponent`, ktorý tieto nové súbory vloží do súborového stromu.

Pri validácií len jedného druhu zdroja sa spúšťajú funkcie `validateSelectedHtml`, `validateSelectedCss` alebo `validateSelectedJs`, v závislosti od aktuálneho validátora. V týchto funkciách sa automaticky rozpozná, že sa jedná o validáciu URL, pripraví sa a spustí validácia.

Validácia HTML a CSS zdrojov je celkom jednoduchá a priamočiara. Pri nej stačí do validátora zadať len validovanú URL. V prípade CSS zdrojov je validátor schopný validovať kód v samostatných súboroch typu CSS alebo aj v rámci CSS kódu v HTML dokumente.

Pri skriptoch je situácia náročnejšia. Najprv sa aplikácia snaží získať základný zdroj HTML (nazývaný často index.html), z ktorého si následne vytiahne pomocou DOM rozhrania všetky skripty. Skripty majú dva druhy formy zápisu: obsah kódu priamo medzi značkami `script` (viz. obrázok 6.12) alebo názov súboru v atribúte `src` (viz. obrázok 6.13)

```
<script>
    console.log("Hello world!");
</script>
```

Obr. 6.12: Obsah kódu priamo v elemente `script`.

```
<script src="helloWorld.js"></script>
```

Obr. 6.13: Názov súboru obsahujúci skript.

Tieto dve formy zápisu rozlišujem pri validácií. V prípade, ak sa priamo v elemente nachádza kód, tak tento kód normálne validujem. Ak sa v elemente nachádza len názov súboru, tak si pred validáciou tento zdroj stiahnem (je to možné keďže názvy súborov majú formu uri). Po validácií, novo stiahnuté zdroje pridám do súborového stromu.

6.4.4 Validácia obsahu editora

Implementácia validácie obsahu editora je najjednoduchšia. Validácia sa spúšťa znova, už vyššie spomenutými funkciami `validateSelectedHtml`, `validateSelectedCss` alebo `validateSelectedJs` v závislosti od vybraného validátora.

6.4.5 Spracovanie výsledkov validácie

Výsledky validácie sa spracovávajú v komponente `ErrorBarComponent`, vo funkciách `validateHtml`, `validateCSS` a `validateJS` v závislosti od druhu jazyka, ktorý sa validuje. V týchto funkciách sa najprv ešte pripravia niektoré dodatočné veci potrebné pre validáciu (príprava možností validácie a druh validácie do štruktúry `ValidationSettings`) a následne sa spustí volanie validátora, ktoré je implementované v službe `ApiService`, ako bolo už vyššie spomenuté. Toto volanie následovne vráti výsledky validácie v dátovej štruktúre.

Tieto výsledky treba následovne, ak sa jedna o validáciu súborov alebo URL, priradiť k správnym súborom. Validovaný súbor reprezentuje dátová štruktúra `ValidatedFile`. Všetky validované súbory sa nachádzajú v poli pozostávajúceho z týchto štruktúr. Tieto štruktúry sú následne použité ako zdroj dát pre vyobrazenie chýb a varovaní užívateľov v aplikácií. Každý validátor má svoj samostatný zoznam validovaných súborov.

6.5 Oprava chýb

Návrhy opráv chýb sa zobrazujú užívateľovi v detaile chyby. Funkcionalita opráv chýb sa spúšťa prostredníctvom komponentu `ErrorComponent`. Po prvom zobrazení detailu chyby sa inicializuje komponent, vytvoria a inicializujú jednotlivé editory a spustí sa funkcionalita, ktorá navrhne opravu danej chyby. Ak oprava existuje, užívateľ ju uvidí v príslušnom editore a zobrazí sa tlačítko pomocou ktorého ju môže nechať opraviť aplikáciou. Keďže celý detail chyby sa inicializuje až vtedy keď je zobrazený, zabraňuje sa tak zahlteniu aplikácie pri väčšom výskyte chýb.

Funkcionalita opráv chýb je implementovaná vrámci troch služieb: `FixHtmlService`, `FixCssService` a `FixJsService`. Každá z týchto služieb obsahuje funkciu, ktorou sa spúšťa proces návrhu opravy chyby. Pre každú chybu sa volá funkcia práve jedenkrát. Na začiatku funkcia spracuje chybovú správu, ktorú dostala z validátora. Správa sa očistí od nadbytočných bielych znakov, odstránia sa nadbytočné medzery zo začiatku a konca správy. Tieto úpravy vykonávajú len v prípade validátora jazyka CSS. Chybové správy ostatných validátorov nepotrebujú takéto úpravy.

Ďalej sa takto upravená chybová správa rozdelí do poľa podľa medzier medzi slovami. Následne sa podľa kľúčových slov správy určí druh chyby a následne sa daná chyba opraví. Spôsob rôznych druhov opráv vysvetlím ďalej. Pre rozpoznanie konkrétnej chyby neporovnávam celé reťazce správ, pretože správy často obsahujú názov elementov, ktorých sa chyba týka. Týmto spôsobom ich viem veľmi ľahko rozpoznať podľa ich pozície v poli.

Ak chybová správa bola rozpoznaná, čo zároveň znamená, že aplikácia poskytuje jej návrh chyby, tak funkcia vráti hodnotu typu boolean. Návrh opravy chyby sa rovno zapíše do konkrétneho editora a zvýrazní.

6.5.1 Opravované chyby jazyka HTML

U jazyka JavaScript aplikácia ponúka možnosť opravy týchto chýb:

- doplnenie chýbajúceho typu dokumentu,
- pridanie atribútu `lang` do elementu `html`,
- doplnenie chýbajúceho elementu `title`,
- chyba v otváracej značke komentára,
- chýbajúca otváracia alebo uzatváracia značka elementu,
- chýbajúce lomítko v uzatváracej značke elementu,
- zlý spôsob zanorenia elementov,
- vymazanie nadbytočného atribútu type, v prípade javascriptového skriptu,
- vymazanie duplicitného atribútu,
- chýbajúca úvodzovka na začiatku hodnoty atribútu,
- chýbajúca medzera medzi jednotlivými atribútmi,
- odstránenie medzery medzi lomítkom a uzatváracou zátvorkou,
- doplnenie chýbajúcich atribútov `src` a `alt` pri elemente `img`,
- doplnenie povinného atribútu určitého elementu,
- odstránenie nepovoleného znaku, medzery, z odkazu v elemente `a` alebo jeho nahradenie escape sekvenciou.

Validátor jazyka HTML určuje pozíciu chyby číslom prvého riadku a stĺpca chyby a jej posledným riadkom a stĺpcom. Pozícia chyby je preto pri mnohých chybových hláseniach veľmi presne určená.

Chyby sú opravované tým spôsobom, že sa nájde pozícia chyby, nahradí sa opravou a oprava sa zvýrazní. Vďaka presnému hláseniu miesta chyby je to v určitých prípadoch jednoduché. Medzi tieto opravy chýb patrí doplnenie chýbajúceho typu dokumentu alebo pridanie atribútu do elementu.

V návrhu som spomínal, že by som chcel opravovať chyby týkajúce sa chybného názvu. Zistil som však, že validátor takéto chyby nehlási, preto som túto opravu vypustil.

Oprava chybnej otváracej značky komentára

Komentár by mal byť správne otvorený pomocou značky `<!--`. Komentár otvorený zlým spôsobom hlási chybu. Aby som bol schopný opraviť takúto chybu, tak v chybnom otvorení komentára musí byť aspoň výkričník (kvôli chybovému hláseniu). V takom prípade si nájdem pozíciu otváracej zátvorky a výkričníka `<!` a od tohto miesta až po koniec chyby všetko nahradím správnym otváracím znakom `<!--`.

Oprava chýbajúcej otváracej alebo uzatváracej značke elementu

Jedná sa o doplnenie chýbajúcej otváracej alebo uzatváracej značky ľubovoľného elementu. Keďže je veľmi ťažké zistiť presnú pozíciu doplnenia chýbajúcej značky, tak túto chybu opravujem tým spôsobom, že značku umiestnim pred začiatok následujúceho elementu (v prípade uzatváracej značky) alebo za koniec predchádzajúceho elementu (v prípade otváracej značky).

Oprava zle zanorených elementov

Pomerne dosť častou chybou, ktorá sa vyskytuje pri validácií jednotlivých stránok, je nepovolené používanie niektorých druhov elementov v iných elementoch. Napríklad, ak jednotlivé položky zoznamu sa začínajú iným elementom ako je element `<li>`. V prípade usporiadaných alebo neusporiadaných zoznamov zaobalím takýto element elementom `<li>`. Pri ostatných prípadoch sa túto chybu snažím riešiť tak, že vymením poradie zanorenia elementov: tie elementy čo sú vo vnútri, premiestnim vonku a tie čo sú vonku premiestnim do vnútra.

Oprava duplicitného atribútu

Ak sa v nejakom elemente nachádza ten istý atribút viackrát, je to hlásené ako chyba. V aplikácií sa tento problém snažím riešiť vymazaním jedného z atribútov. V aplikácií je pevne určené, že sa vymaže posledný takýto atribút. Chyba sa vyskytuje pre každý duplicitný atribút. Pre vyhľadanie atribútu používam regulárny výraz `[ ]*=[ ]*["´]`[\w/;.:=-?]*["´]. Vpredu má ešte pripnutý názov atribútu, získaný zo správy. Podobný regulárny výraz používam aj v ďalších opravách chýb, týkajúcich sa atribútov.

Oprava chýbajúcej úvodzovky na začiatku atribútu

Chybová správa špecifikuje riadok, na ktorom sa nachádza atribút. Pomocou regulárneho výrazu `[a-zA-Z]*[ ]*=[ ]*[\w/;.: =-?]+["´]`, vyhľadám pozíciu tohto atribútu na chybovom riadku a za znamienko -ümiestnim úvodzovku. Typ úvodzovky zistím podľa uzatváracej úvodzovky.

6.5.2 Opravované chyby jazyka CSS

U jazyka CSS aplikácia ponúka možnosť opravy týchto chýb:

- oprava chybného názvu vlastnosti,
- chybný názov absolútnych a relatívnych jednotiek veľkosti[3],

[3] https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units

- doplnenie chýbajúcej bodkočiarky,
- doplnenie chýbajúcej dvojbodky,
- doplnenie chýbajúcich zátvoriek.

Validátor jazyka CSS pozíciu chyby určuje, len číslom riadku. V situácií, keď je veľa kódu na jednom riadku, tak návrhy chýb nemusia pracovať správne. Pri navrhovaní chýb som počítal s možnosťou, že kód bude vyzerať normálne (jeden príkaz na jednom riadku). Bude tak upravený užívateľom alebo určitým nástrojom.

Ako bolo spomenuté už pri návrhu, pri navrhovaní opráv chýb pre tento jazyk, veľké množstvo chýb je závislé od konkrétnych úmyslov a použitých hodnôt autora kódu. Napríklad, ak užívateľ použije chybnú hodnotu, ktorá pre danú vlastnosť neexistuje, tak nie je ľahké alebo vôbec možné určiť, ktorú hodnotu z povolenej množiny hodnôt by mohol chcieť použiť a následne mu ju ponúknuť. Preto som sa obmedzil len na veľmi základné návrhy opráv chýb. Možnosť rozšíriť tento zoznam vidím pri ďalšom vývoji aplikácie.

Oprava chybného názvu vlastnosti

Pri oprave chybného názvu vlastnosti využívam obsah chybovej správy. Ten už obsahuje návrh správneho názvu vlastnosti. Z chybovej správy si ho potom vytiahnem a nahradím ním v editore chybnú správu.

Oprava chybného názvu jednotiek

Opravu chybného názvu veľkostí jednotiek som rozdelil do niekoľkých fáz. Chybová správa presne určuje, u ktorej vlastnosti sa nachádza chyba. Okrem pozície neurčuje nič iné. Najprv si preto získam z chybného riadku chybnú jednotku. Jednotka musí byť umiestnená za číslom (inak sa jedná o iný typ chyby). Následne pomocou regulárnych výrazov zistím, či sa v chybnej jednotke nenachádza preklep spôsobený pridaním nadbytočných znakov. Ak to nie je tak, tak sa pozriem na jednotlivé znaky chybnej jednotky. Ak sedia s niektorým znakom existujúcej jednotky, tak ju ňou nahradím. Veľa jednotiek má znaky na prvom alebo poslednom mieste, ktoré sú jedinečné, takže je dosť možné, že sa presne trafím do jednotky, ktorú autor pôvodne myslel. Pri nejednoznačných znakoch, vyskytujúcich sa u viacerých jednotiek, je preto presne určené, ktorá jednotka nahradí tú chybnú.

Oprava chýbajúcich zátvoriek

Pri dopĺňaní chýbajúcich zátvoriek sa pravá zátvorka dopĺňa na koniec riadka s chybou. Miesto pre chýbajúcu ľavú zátvorku sa snažím nájsť podľa prvého selektoru pre identifikátor alebo triedu, nachádzajúcim sa nad miestom chyby. Opravu ľavej a pravej chýbajúcej zátvorky som oddelil. V jeden čas je možné doplniť len jednú zátvorku. Urobil som tak jednak kvôli architektúre aplikácie, menším problémom pri zvýrazňovaní chýb a hlavne kvôli tomu, aby to neodradilo užívateľa od opravy tejto chyby, ak jedna zátvorka nebude správne opravená.

6.5.3 Opravované chyby jazyka JavaScript

U jazyka JavaScript aplikácia ponúka možnosť opravy troch chýb:

- doplnenie bodkočiarky na koniec chybného riadku,

- nahradenie deklarácií premenných pomocou kľúčového slova `var` za deklaráciu pomocou `let`,
- doplnenie chýbajúcej definície premennej.

Validátor jazyka JavaScript pozíciu chyby určuje len číslom riadku. Preto pri navrhovaní opravy chyby vychádzam z rovnakých predpokladoch ako pri jazyku CSS.

Pri doplnení chýbajúcej definície premennej sa definícia doplní riadok nad chybou.

Pri jazyku JavaSript som sa ocitol vo veľmi zložitej pozícii opravovať vôbec nejaké chyby, keďže ich obrovské množstvo závisí od sémantiky kódu. Snažil som sa umožniť aspoň týchto zopár základných opráv. Priestor pre vytváranie ďalších opráv preto vidím pri ďalšom vývoji aplikácie, zahrňujúcu komplexnú analýzu kódu.

6.5.4 Krokovanie chýb

Aplikácia umožňuje užívateľovi vracať opravy chýb, ktoré vykonal. Ak vrátenie opravy chyby oľutuje, môže tento krok zvrátiť. Aplikácia tak umožňuje krokovanie chýb. Krokovať chyby užívateľ môže pomocou tlačidiel umiestnených nad zoznamom chýb. Vrátiť chybu môže užívateľ navyše z detailu opravených chýb. Implementácia krokovania sa nachádza na komponente `ErrorBarComponent`.

Vrátenie sa do stavu pred opravou chyby - UNDO

Ak sa užívateľ chce vrátiť do stavu pred opravou chyby, zobrazí sa modálne okno, ktoré potrebuje potvrdenie o pokračovaní. Vrátením opravy chyby sa vráti stav celého editora do stavu pred chybou. Všetky najnovšie opravy a editácie sa vymažú. Po potvrdení vrátenia chyby sa v kóde nastavia rôzne prepínače, aktualizuje sa obsah súborov, štruktúr a editora a znovu sa validuje aktuálny súbor alebo obsah editora. Nakoniec sa vytvorí nový záznam typu `RemovedFix` a vloží do príslušného zoznamu, ktorý uchováva všetky vrátené opravy chýb. Tento zoznam sa bude následne využívať pri zvrátení tohto kroku.

Znova uplatnenie vrátenej opravy chyby - REDO

Užívateľ môže jednoducho znova uplatniť vrátený návrh opravy chyby. V takomto prípade sa načíta zo zoznamu vrátených opráv chýb, posledne takto vrátená oprava typu `RemovedFix`. Z tejto štruktúry sa použije znenie chybovej správy, ktorá táto oprava riešila. Pomocou tejto správy sa nájde chyba v zozname chýb. Táto chyba sa potom použije ako vstup do funkcie, ktorá navrhne opravu chyby. Následne sa aktualizuje súbor a obsah editora znova vrátenou opravou chyby.

6.6 Práca so súbormi a ich obsahom

Aplikácia pracuje so súbormi a ich obsahom hlavne v rámci komponentu `FileManagerComponent`. Tento komponent implementuje funkcionality ako načítanie obsahu súborov, ich zobrazenie v súborovom strome a vytváranie nových súborov. Sťahovanie súborov užívateľom implementuje backend aplikácie. Sťahovanie zdrojov vzdialených stránok, potrebných pri validácií pomocou URL je implementované v rámci služby `ApiService`. Implementáciu ďalej detailnejšie opíšem.

6.6.1 Súborový strom

Pre súborový strom používam nástroj `angular2-tree`[4]. Nástroj je kompatibilný s frameworkom Angular, ľahko sa používa a má plno rôznych funkcionalít. Vzhľad súborového stromu je možné rôzne prispôsobiť pomocou tried jazyka CSS, súbory je možné presúvať potiahnutím, je možné nastaviť animácie a ďalšie.

Súborový strom je možné použiť v kóde veľmi jednoducho. Ilustruje to výsek z môjho kódu na obrázku 6.14. Súborový strom je reprezentovaný samostatným HTML elementom. Obsah stromu je reprezentovaný hodnotou atribútu `nodes`, nastavenia sú priradené do atribútu `options`. Nástroj má rôzne spúšťače. Na spúšťač som schopný pripnúť nejakú vlastnú funkcionalitu. Jednou z takých spúšťačov je `activate`. Spúšťa sa ak užívateľ klikne na niektorý z uzlov stromu. V mojom kóde sa pri kliknutí na uzol spúšťa funkcia `showContent`, ktorá uloží aktuálny obsah editora a zobrazí obsah aktuálne vybraného súboru.

```
<tree-root #tree class="tree"
    (activate)="showContent($event.node)"
    [nodes]="fileTreeNodes"
    [options]="options">
</tree-root>
```

Obr. 6.14: Použitie súborového stromu v kóde.

Nástroj poskytuje štruktúru, ktorú je potrebné využiť aby bolo možné správne zobraziť súborový strom. Štruktúra je veľmi jednoduchá, ako je možné vidieť na obrázku 6.15. Štruktúra sa skladá z poľa uzlov, ktoré reprezentujú priečinok alebo súbor. Priečinok aj súbor majú spoločné položky, ako identifikátor a meno. Priečinok má navyše položku `children`, čo je vlastne obsah daného priečinka. Obsah priečinka je znova reprezentovaný poľom uzlov. Pri uzle typu súbor som pridal položku `content`, ktorá uchováva obsah súboru. Táto položka nie je súčasťou pôvodnej štruktúry poskytovanej nástrojom. Je to môj dodatok.

[4] https://angular2-tree.readme.io/docs

```
fileTreeNodes = [
  {
    id: 1,
    name: "dir1",
    children: [
      { id: 2, name: "file1", content: "hello world" },
      { id: 3, name: "file2", content: "hello space" }
    ]
  },
  {
    id: 4,
    name: "dir2",
    children: [
      { id: 5, name: "file3", content: "hello thesis" },
      {
        id: 6,
        name: "dir3",
        children: [
          { id: 7, name: "file4", content: "lorem ipsum" }
        ]
      }
    ]
  }
];
```

Obr. 6.15: Štruktúra reprezentujúca súborový strom.

Súborový strom sa vytvára v komponente `FileManagerComponent` pri načítaní súborov, zip archívu alebo prvej validácií URL. Aby bol súborový strom prístupný pre ostatné komponenty, tak pri každej jeho aktualizácií ho ukladám na službu `ComponentCommunicationService`, ktorá implementuje niektoré rozširujúce funkcionality nad stromom ako získanie, uloženie obsahu súboru, alebo získanie mena súboru na základe jeho identifikátora.

Pri vytváraní súborov na generovanie jedinečných identifikátorov, používam nástroj `uuid`[5].

Načítanie súborov

Užívateľ môže načítať súbory s príponami *html*, *css* a *js*. Taktiež môže načítať celý komprimovaný priečinok typu *zip*. Pre načítavanie súborov používam dve rozličné funkcie: `loadFiles` pre načítavanie súborov a `loadZip` pre načítanie komprimovaného priečinku. Obe funkcie sú implementované v rámci komponentu `FileManagerComponent`.

Pre načítavanie súborov používam objekt jazyka JavaSript `FileReader`. Načítavanie archívov typu *zip* je komplikovanejšie. Aby som ho mohol využiť na frontende aplikácie, tak pre tento účel som prebral nasledovnú implementáciu[6]. Kód umožňuje používať open-

[5] https://github.com/uuidjs/uuid#readme

[6] https://medium.com/@dazcyril/unzip-files-in-the-browser-with-angular-7-fcdc3040f3c1

source knižnicu `zip.js`[7] priamo vo frameworku Angular. Implementácia spočíva v stiahnutí niektorých častí knižnice `zip.js` a zaobalenie ich funkcí v novo vytvorenej službe s názvom `ZipService`.

6.6.2 Sťahovanie zdrojov vzdialených webových stránok

Po validácií adresy URL aplikácia sťahuje chybné zdroje, aby si ich užívateľ mohol opraviť a opravenú verziu nahrať znova na svoj server. Aplikácia sťahuje len chybné súbory a nie všetky webové zdroje. Rozhodol som sa tak kvôli tomu, aby sa moja aplikácia úplne nezahltila a poskytovala užívateľovi len to potrebné.

Kvôli bezpečnostným dôvodom nie je možné pristupovať k vybraným zdrojov cudzieho pôvodu. Pre prístup používam samostatný proxy server. Tento server nie je súčasťou backendu aplikácie. Pri vytváraní proxy servera som použil hotový Node.js proxy server[8]. Tento nástroj pridáva CORS hlavičky do HTTP požiadavkov.

Služby proxy servera volám z frontendu aplikácie. Volanie je implementované v rámci služby `ApiService`, funkciou `getCode`, ktorá je zobrazená na obrázku 6.18. Za adresou proxy serveru sa nachádza adresa zdroja. Proxy server následne pridá CORS hlavičku do požiadavku a vráti požadované webové zdroje. Ukážka proxy servera je na obrázku 6.17, ktorý som prebral z webu [**?**].

```
async getCode(url: string): Promise<any> {
  return await this.http
    .get(environment.proxyServerUrl + url, { responseType: 'text' })
    .toPromise()
    .catch((error) => console.log(error));
}
```

Obr. 6.16: Volanie služby proxy serva z frameworku Angular.

```
var port = process.env.PORT || 8081;

var cors_proxy = require('cors-anywhere');
cors_proxy.createServer({
    originWhitelist: [], // Allow all origins
    requireHeader: ['origin', 'x-requested-with'],
    removeHeaders: ['cookie', 'cookie2']
}).listen(port, function () {
    console.log(`Running CORS Anywhere on http://localhost:${port}`);
});
```

Obr. 6.17: Ukážka proxy servera [**?**].

6.6.3 Sťahovanie upravených súborov užívateľom a súborov diff

Počas chodu aplikácie pracujem len s obsahmi súborov. Aby si užívateľ mohol stiahnuť zmeny, tak je potrebné súbory vytvoriť, komprimovať a stiahnuť. Toto sa odohráva v backende aplikácie. Pri volaní tejto služby, frontend zasiela na backend obsah súborové stromu.

[7] https://gildas-lormeau.github.io/zip.js/
[8] https://github.com/Rob--W/cors-anywhere#readme

Následne sa vytvorí dočasný priečinok, kde sa zo súborového stromu vytvoria skutočné súbory a priečinky. Dočasný priečinok má unikátny názov vygenerovaný pomocou nástroja `uuid`, ktorý používam aj pri generácií identifikátorov pre jednotlivé uzly súborové stromu vo frontende aplikácie. Pomocou nástroja `archiver`[9] sa vytvorí zip archív a ten sa odošle naspäť frontendu, ktorý vyvolá jeho stiahnutie do zariadenia užívateľa. Dočasný priečinok sa nakoniec vymaže. Implementáciu môžete vidieť na obrázku 6.18.

```
router.post('/download-zip', (req, res) => {
    let treeStructure = req.body;
    let uniqueDirPath = path.join(__dirname, '../assets', 'tmpDownloads', uniqueId.v4())
    let srcDirPath = path.join(uniqueDirPath, 'files');
    let zippath = path.join(uniqueDirPath, "complexValidatorOutput.zip");

    fs.mkdirSync(srcDirPath, { recursive: true });
    createFileTree(treeStructure, srcDirPath);
    zipFileTree(srcDirPath, zippath).then( (resolve, reject) => {
        res.sendFile(zippath);
        console.log(resolve);
        console.log(reject);

        rimraf(uniqueDirPath, function () { console.log("done"); });
    });
});
```

Obr. 6.18: Implementácia vytvárania a archivovania súborov.

Pre vytváranie súborov diff pre patch používam nástroj `jsdiff`[10]. Vytváranie týchto súborov prebieha tak isto v backende aplikácie. Na ich stiahnutie používam podobný kus kódu, ako pri sťahovaní súborov.

[9] https://github.com/archiverjs/node-archiver
[10] https://github.com/kpdecker/jsdiff

Kapitola 7

Testovanie funkčnosti aplikácie

V tejto kapitole popíšem spôsob, akým som testoval výslednú aplikáciu a jej výsledok.

7.1 Požiadavky na funkčnosť aplikácie a dáta použité pri testovaní

Aplikácia má byť schopná validovať zdrojové kódy troch rozličných druhov jazykov (HTML, CSS a JavaScript). Aplikácia má umožňovať užívateľovi validovať svoje zdrojové súbory rôznymi spôsobmi (pomocou načítania súborov, URL alebo zadaním kódu do editora) a nastaviť si rôzne možnosti validácie. Aplikácia umožňuje validáciu jednotlivých zdrojov samostatne alebo komplexne (celú webovú stránku naraz).

Výsledok validácie sa má zobraziť vo výpise chýb, roztriedený podľa súborov, chýb a varovaní.

Aplikácia má poskytovať užívateľovi návrhy chýb, ktoré môže využiť pri oprave svojich zdrojových súborov. Po ukončení opravy, aplikácia má umožňovať užívateľovi si tieto zdrojové súbory stiahnuť, poprípade stiahnuť diff pre patch.

Validáciu pomocou URL adresy som testoval na troch rôznych webových stránkach: `https://www.vutbr.cz/`, `https://www.damejidlo.cz/` a `https://koronavirus.mzcr.cz/`. Pre testovanie validácie pomocou súborov a vstupu do editora som použil vlastné zdrojové kódy. Tieto kódy je možné zobraziť v aplikácií, pomocou tlačidiel pri validácií obsahu editora. Pre účely testovania validácie nahraných zdrojových kódov som si z týchto kódov vyrobil súbory, ktoré som následne načítal do aplikácie. Aplikáciu som testoval v prehliadači Google Chrome.

7.2 Výsledky testovania

Najprv som začal s testovaním validácie načítaných súborov. Načítal som si tri súbory, obsahujúce kódy v jazyku HTML, CSS a JavaScript. Načítanie súborov bolo rýchle a úspešné. Po kliknutí na jednotlivé súbory v súborovom strome sa ich obsah objavil v editore. Skúsil som najprv validovať dokument s kódom v jazyku HTML. Validácia prebehla bez problémov a v zozname chýb sa zobrazil správny názov validovaného súboru, s dobrým počtom chýb a zoznamom chýb. Po prejdení myšou po jednotlivých chybách, sa zvýraznili v hlavnom editore. Po rozkliknutí prvej chyby sa správne zobrazili dva malé editory, neumožňujúce editáciu, zobrazujúce chybu a návrh opravy. Keďže chybu je možné opraviť (jedná sa o chýbajúci typ dokumentu), zobrazilo sa zelené tlačítko, umožňujúce opravu. Chyba

sa po stlačení tlačidla správne opravila. Rovnakým spôsobom som opravil ďalšie chyby a varovania.

Opravené chyby sa správne, v poradí v akom boli opravené, zobrazili v príslušnej záložke. Po rozkliknutí detailu opravy sa znova zobrazili dva editory, ukazujúce opravenú chybu a samotnú opravu, farebne zvýraznené. Na konci detailu chyby sa zobrazilo červené tlačítko pre vrátenie opravy chyby. Po jeho stlačení sa stav editora správne vrátil do stavu pred chybou. Všetky vykonané opravy po vrátenej oprave zmizli. Pomocou tlačítka pre vrátenie vrátených opráv sa tieto opravy znova aplikujú.

Pokúsil som sa súbory stiahnuť, pomocou troch dostupných možností. Stiahnutie prebehlo úspešne. Súbor, ktorý som opravoval sa stiahol v opravenej podobe a súbor obsahujúci diff vyzeral byť v poriadku.

Následne som podobným spôsobom otestoval validátor pre jazyk CSS a JavaScript. Skúsil som zmeniť možnosti validácie pomocou tlačidla vedľa prepínača medzi validátormi (obsahuje ozubené koliesko). Zmeny sa prejavili pri validácií (skúšal som možnosť nezobrazovania varovaní pri jazyku HTML a CSS, alebo zakázanie používania nedeklarovaných premenných pri jazyku JavaScript). Testovanie tohto druhu validácie som ukončil validáciou všetkých súborov naraz, ktorá taktiež prebehla v poriadku.

Pri testovaní tejto časti som zistil, že sa nesprávne zobrazoval názov opravenej chyby vo validátore jazyka JavaScript. Ďalej pri prepnutí do iného módu vstupu sa nevymazal obsah štruktúr REDO funkcionality. Tieto chyby som opravil a už sa v aplikácií nevyskytujú.

Podobne som postupoval aj pri testovaní validácie obsahu editora. Všetko bolo funkčné.

Nakoniec som testoval validáciu URL adresy. Otestoval som ju na vyššie uvedených webových stránkach. Aplikácia zdroje zvalidovala a stiahla dotyčné chybné súbory. Pri väčšom množstve chýb sa aplikácia spomalila. Narazil som na problémy, keď aplikácia nechcela zobraziť názov súboru *index.html*, pri validácií pomocou CSS validátora, alebo chybne zobrazila čísla riadkov s chybou pri validácií súboru *index.html* pomocou validátora pre jazyk JavaScript. Tieto chyby som odstránil a pri testovaní sa viac nevyskytli.

Nakoniec som ešte skúsil načítať komprimovaný súbor. Súbor sa načítal správne a vytvoril sa súborový strom.

Na záver testovania som skonštatoval, že aplikácia je funkčná (po opravení chýb funguje na množine testovaných príkladoch). Pri veľkom množstve chýb alebo varovaní (tisíc a viac) sa aplikácia spomalí. Všimol som si, že k spomaleniu aplikácie dochádza, ak si užívateľ zobrazí hlásenia chýb pre daný súbor. Pri ďalšom vývoji by sa preto mohlo zamerať aj na optimalizáciu aplikácie. Validácia jazyka HTML trvá niekedy trochu dlhšie, ale vždy sa úspešne ukončí. Problém vidím u samotnej webovej služby, ktorú volám pri validácií.

Aplikáciu som dal otestovať skutočnému užívateľovi, ktorý hodnotil aplikáciu pozitívne. Užívateľ sa vedel v aplikácií zorientovať a nemal problém ju používať. Oceňoval farebnosť a označenia jednotlivých tlačidiel. Veľmi sa mu páčila funkcionalita, že sa chyba v editore zobrazila na príslušnom riadku a farebne zvýraznila.

Kapitola 8

Záver

Cieľom práce bolo vytvoriť aplikáciu umožňujúcu komplexnú validáciu webovej stránky, pomocou validátorov jazykov HTML, CSS a JavaSript. Aplikácia má tieto chyby užívateľovi rozumne zobraziť, umožniť mu si ich priamo v aplikácií opraviť a následne stiahnuť opravené súbory.

Na začiatku tvorby práce som sa zoznámil s dostupnými webovými validátormi, ich aplikačným rozhraním a s metódami prístupu k vzdialeným webovým stránkam. Následne som urobil návrh užívateľského rozhrania, zvolil si vhodné webové technológie pre implementáciu a navrhol základnú štruktúru aplikácie. Zvolil som si vhodné validátory pre každý jazyk, tak aby zodpovedali čo najlepšie štandardu a zároveň sa dali jednoducho použiť. Navrhol som rôzne druhy chýb, ktoré by aplikácia mohla byť schopná opraviť. Nakoniec som funkcionalitu aplikácie otestoval na vhodných webových stránkach.

Pri tvorbe aplikácie som si prehĺbil znalosti frameworku Angular, ktorý som použil pri implementácií. Zároveň som si vyskúšal pracovať s aplikačným rozhraním validátorov. Zaujímavé bolo nachádzať spôsoby, akým je možné opraviť rôzne chyby v daných jazykoch.

Po prevedení testov a opravení chýb hodnotím, že výsledná aplikácia je funkčná a umožňuje užívateľovi validovať svoje zdrojové súbory. Chyby sa mu jednoducho zobrazia a priamo si ich vie opraviť v editore. K niektorým chybám užívateľovi poskytujem návrhy opravy. Opravené zdrojové súbory si vie následne stiahnuť. Pri väčšom počte chýb sa môže stať, že sa aplikácia spomalí.

V aplikácií vidím veľa možností, akým by sa mohol uberať ďalší vývoj. Je tu priestor pre zlepšenie užívateľského zážitku z používania aplikácie, ako napríklad v pridaní rôznych filtrov pre chyby, alebo väčšia prispôsobivosť aplikácie (rôzne farebné štýly). Zaujímavým vylepšením by mohlo byť pridanie validátorov nových jazykov, napríklad jazyka PHP, TypeScript a iné. Veľký potenciál vidím v rozšírení zoznamu opravovaných chýb a varovaní, alebo vo vylepšení a spresnení súčasných návrhov opráv chýb. Vhodné by sa bolo taktiež pozrieť na optimalizáciu aplikácie, aby sa nespomaľovala pri väčšom množstve chýb a varovaní.

Literatúra

[1] Berners Lee, T. *Methods* [online]. W3C, 1992 [cit. 2020-04-14]. Dostupné z: https://www.w3.org/Protocols/HTTP/Methods.html.

[2] Contributors. *CSSLint* [online]. GitHub [cit. 2020-04-24]. Dostupné z: https://github.com/CSSLint/csslint.

[3] Contributors. *Express* [online]. Node.js Foundation [cit. 2020-04-08]. Dostupné z: https://expressjs.com/.

[4] Contributors. *Introduction to Angular concepts* [online]. Google [cit. 2020-04-08]. Dostupné z: https://angular.io/guide/architecture.

[5] Contributors. *Introduction to components and templates* [online]. Google [cit. 2020-04-08]. Dostupné z: https://angular.io/guide/architecture-components#introduction-to-components-and-templates.

[6] Contributors. *Introduction to Node.js* [online]. OpenJS Foundation [cit. 2020-04-07]. Dostupné z: https://nodejs.dev/introduction-to-nodejs.

[7] Contributors. *TypeScript* [online]. Microsoft, 22. októbra 2019 [cit. 2020-04-16]. Dostupné z: https://github.com/microsoft/TypeScript.

[8] Contributors. *Cross-Origin Resource Sharing (CORS)* [online]. MDN Web Docs, 24. februára 2020 [cit. 2020-04-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[9] Contributors. *HTTP cookies* [online]. MDN Web Docs, 13. apríla 2020 [cit. 2020-04-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

[10] Contributors. *Introduction to the DOM* [online]. MDN Web Docs, 26. januára 2020 [cit. 2020-04-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[11] Contributors. *JavaScript* [online]. MDN Web Docs, 30. marca 2020 [cit. 2020-04-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[12] contributors, P. *Puppeteer* [online]. GitHub [cit. 2020-04-23]. Dostupné z: https://github.com/puppeteer/puppeteer.

[13] Daz. *Unzip Files in the Browser with Angular 7* [online]. Medium, 19. decembra 2018 [cit. 2020-05-06]. Dostupné z: https://medium.com/@dazcyril/unzip-files-in-the-browser-with-angular-7-fcdc3040f3c1.

[14] Georgieva, M. *How To Use node.js, request and cheerio to Set Up Simple Web-Scraping* [online]. DigitalOcean, 16. septembra 2013 [cit. 2020-05-21]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-use-node-js-request-and-cheerio-to-set-up-simple-web-scraping.

[15] Hidayat, A. *Documentation on using Esprima* [online]. Ariya Hidayat [cit. 2020-04-24]. Dostupné z: https://esprima.org/doc/.

[16] JSFoundation. *Getting Started with ESLint* [online]. JS Foundation [cit. 2020-04-24]. Dostupné z: https://eslint.org/docs/user-guide/getting-started.

[17] Kurose, J. F. a Ross, K. W. *Computer Networking: A Top-Down Approach*. 6. vyd. Pearson, 2013. ISBN 978-0-13-285620-1.

[18] Patel, H. *5 Best JavaScript Web Scraping Libraries and Tools* [online]. codementor, 26. februára 2019 [cit. 2020-04-23]. Dostupné z: https://www.codementor.io/@hirenpatel545/5-best-javascript-web-scraping-libraries-and-tools-sicow2rx9.

[19] RedHat. *What is an API?* [online]. Red Hat [cit. 2020-05-21]. Dostupné z: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces.

[20] RefsnesData. *What is the HTML DOM?* [online]. W3Schools [cit. 2020-04-15]. Dostupné z: https://www.w3schools.com/whatis/whatis_htmldom.asp.

[21] Scrapinghub. *What is web scraping?* [online]. Scrapinghub [cit. 2020-04-23]. Dostupné z: https://scrapinghub.com/what-is-web-scraping.

[22] Sivonen, H., Foundation, M. a Pieters, S. *About Validator.nu* [online]. [cit. 2020-04-24]. Dostupné z: https://about.validator.nu/.

[23] s.r.o., A. T. *Apify SDK* [online]. Apify Technologies s.r.o. [cit. 2020-04-23]. Dostupné z: https://sdk.apify.com/.

[24] Technologies, A. *Quick Start* [online]. Apify Technologies [cit. 2020-05-21]. Dostupné z: https://sdk.apify.com/docs/guides/quick-start.

[25] W3C. *CSS Selectors* [online]. W3C [cit. 2020-04-07]. Dostupné z: https://www.w3schools.com/css/css_selectors.asp.

[26] W3C. *CSS Syntax* [online]. W3C [cit. 2020-04-07]. Dostupné z: https://www.w3schools.com/css/css_syntax.asp.

[27] W3C. *Documentation for the W3C Markup Validator* [online]. W3C [cit. 2020-04-13]. Dostupné z: https://validator.w3.org/docs/.

[28] W3C. *Documentation index for the CSS Validator* [online]. W3C [cit. 2020-04-13]. Dostupné z: https://jigsaw.w3.org/css-validator/documentation.html.

[29] W3C. *HTML & CSS* [online]. W3C [cit. 2020-04-07]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss.

[30] W3Schools. *HTML Tutorial* [online]. W3Schools [cit. 2020-04-06]. Dostupné z: https://www.w3schools.com/html/default.asp.

[31] Waldron, R., Potter, C., Pennisi, M. a Page, L. *Documentation* [online]. JSHint [cit. 2020-04-13]. Dostupné z: https://jshint.com/docs/.

Príloha A

Obsah pamäťového média

- `complex-validator` - zdrojové kódy aplikácie
- `validator-thesis.zip` - zdrojové kódy textu práce
- `Komplexný-validátor-webových-stránok.pdf` - pdf verzia práce

Príloha B

Dostupnosť aplikácie a jej spustenie v lokálnom prostredí

Aplikácia je dostupná online, pomocou služby Heroku, na linku https://complex-webpage-validator.herokuapp.com/.

Aplikáciu je možné spustiť aj v lokálnom prostredí. Pre tento účel je potrebné mať nainštalované minimálne tieto verzie programov:

- `npm` vo verzii 6.9.0 a vyššie,
- `Node.js` vo verzii 10.16.0 a vyššie.

Ostatné závislosti sa nainštalujú pomocou nástroja `npm`, ako je uvedené nižšie.
Aplikáciu spustíte u seba na localhoste pomocou týchto krokov:

1. `cd ./complex-validator`
2. `npm install`
3. `ng serve`

Pre spustenie backendu aplikácie a proxy servera postupujte takto:

1. `cd ./complex-validator/server/src`
2. `node ./server.js`
3. `node ./proxyServer.js`

Ak by bol nástroj diff nefunkčný a backend by hlásil chybu, tak je potrebné v súbore `/complex-validator/server/src/api.js` odkomentovať riadok č. 32 a zakomentovať riadok č. 31.