VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÝCH TECHNOLÓGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÝCH SYSTÉMOV
DEPARTMENT OF INFORMATION SYSTEMS

ANALYZÁTOR OPTIMALIZACE A BEZPEČNOSTI WEBOVÝCH APLIKACÍ

ANALYZER OF WEB APPLICATION OPTIMIZATION AND SECURITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **BORIS ŠTRBÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN BEDNÁŘ**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Štrbák Boris**
Program: Informační technologie
Název: **Analyzátor optimalizace a bezpečnosti webových aplikací**
Analyzer of Web Application Optimization and Security
Kategorie: Analýza a testování softwaru

Zadání:

1. Nastudujte problematiku analýzy optimalizace a bezpečnosti webových aplikací. Seznamte se s existujícími nástroji a prozkoumejte možnosti využití těchto nástrojů.
2. Navrhněte analyzátor webových aplikací, navrhněte vlastní metody analýzy optimalizace a bezpečnosti webových aplikací a vyberte vhodné externí služby pro integraci do Vašeho řešení.
3. Analyzátor implementujte jako webovou aplikaci. Implementujte vlastní navržené metody analýzy a zajistěte napojení vybraných externích služeb do analyzátoru.
4. Proveďte experimenty na existujících webových aplikacích a vyhodnoťte výsledky experimentů.
5. Zhodnoťte výsledky a přínos práce a navrhněte možnosti jejího dalšího pokračování.

Literatura:

- PETRŽELKA, Jiří. *Web Site Optimization*. Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Miloš Eysselt, CSc.
- GRIGORIK, Ilya. *High Performance Browser Networking: What Every Web Developer Should Know about Networking and Web Performance*. 2013. ISBN 978-1-4493-4472-6.
- MUELLER, John. *Security for web developers: using JavaScript, HTML and CSS /*. Sebastopol, CA: O´Reilly, 2016. ISBN 978-1-491-92864-6.
- DOVER, Danny a Erik DAFFORN. *Search Engine Optimization (SEO) Secrets*. New Jersey: John Wiley & Sons, 2011. ISBN 978-0-470-55418-0.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2 zadání.

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

Vedoucí práce: **Bednář Martin, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 11. října 2021

Abstrakt

Cieľom tejto práce bolo vytvoriť webovú aplikáciu, ktorá by automatizovala pravidelné analyzovanie jednotlivých stránok webu a získané informácie prezentovala užívateľovi. Výsledkom je Single-Page aplikácia (SPA) vytvorená pomocou JavaScript rámca (*framework*) Vue.js s využitím knižnice Vuetify. Serverová časť, ktorá obsahuje REST API a procesy na skenovanie a analýzu webu, je implementovaná v PHP rámci Laravel. Obe časti sú navrhnuté tak, aby bežali v kontajnerizovanom prostredí. K tomuto účelu je použitý Docker. V prvej časti práce je popísaná problematika optimalizácie stránok, zabezpečenia a SEO. Spomenuté poznatky sa potom využili pri samotnej implementácii.

Abstract

The goal of this project was to create a web application that would automate the regular analysis of individual pages of the site and present obtained information to the user. Final product is a Single-Page application (SPA) created using JavaScript framework Vue.js with the Vuetify library. The server part, which contains REST API and scanning and analysis processes, is implemented in the PHP framework Laravel. Both parts are designed to run in a containerized environment. Docker is used for this purpose. The first part of the thesis describes the issues of site optimization, security and SEO. The mentioned findings were then used in the implementation itself.

Kľúčové slová

HTTPS, PLT, optimalizácia, SEO, Rich snippet, OpenGraph, bezpečnosť, Laravel, Vue.js, Vuetify, SPA, Docker

Keywords

HTTPS, PLT, ooptimization, SEO, Rich snippet, OpenGraph, security, Laravel, Vue.js, Vuetify, SPA, Docker

Citácia

ŠTRBÁK, Boris. *Analyzátor optimalizace a bezpečnosti webových aplikací.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačných technológií. Vedoucí práce Ing. Martin Bednář

Analyzátor optimalizace a bezpečnosti webových aplikací

Prehlásenie

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

.....
Boris Štrbák
6. mája 2022

Poďakovanie

Chcel by som poďakovať vedúcemu práce, Ing. Martinovi Bednářovi, za ochotu ujať sa vedenia mojej vlastnej témy, jeho cenné rady a užitočné konzultácie.

Obsah

Úvod

Od kedy vznikli prvé webové stránky pred zhruba 30 rokmi, prístup k ich tvorbe a designu prešiel mnohými zmenami a inováciami. Spoločnosti z rôznych odvetví si začali uvedomovať, že potrebujú svoj vlastný priestor na internete. Jednou z prvých spoločností, ktorá spustila svoje vlastné stránky, boli americký predajcovia pizze, ktorí takto začali zbierať rezervácie. Postupne online priestor zahltili rôzne internetové obchody, rezervačné alebo informačné systémy, ktoré bojujú o každého jedného návštevníka.

Vstupnou bránou k internetu sa postupne stali vyhľadávače ako Google, Bing, Yahoo alebo český Seznam. Okrem reklám a vyskakovacích okien sú práve oni jedným zo spôsobov ako na Vaše stránky dostať čo najviac ľudí. Každá spoločnosť, každý podnikateľ chce, aby boli jeho stránky súčasťou čo najviac vyhľadávaní a aby sa zobrazovali čo najvyššie. Každý vyhľadávač má svoj vlastný spôsob, ktorým hodnotí webové stránky a podľa toho ich priraďuje k rôznym kľúčovým slovám a radí vo výsledkoch alebo zobrazuje pri našeptávaní užívateľovi.

Dnes už existuje mnoho nástrojov, ktoré pomáhajú ako vývojárom, tak aj majiteľom webov sledovať ako sú na tom ich stránky čo sa týka optimalizácie nie len pre vyhľadávače. Tieto nástroje ale často pracujú iba s jednou URL a neposkytujú prehľadné štatistiky pre celý web. To môže byť problém pokiaľ je potrebné pravidelne sledovať obrovský informačný systém s množstvom podstránok. Práve na toto sme s kolegami natrafili počas práce na internetových obchodoch. Cieľom tejto práce bolo vytvoriť aplikáciu, ktorá umožní skupine užívateľov spravovať webové projekty na základe domény alebo URL domovskej stránky. Aplikácia potom na pozadí celý web zmapuje, a potom bude v pravidelných intervaloch získavať informácie o optimalizácií jednotlivých podstránok, ktoré sa vo forme spätnej väzby spolu s radami na zlepšenie zašlú späť užívateľovi.

V kapitole 1 sa dočítate viac o tom prečo sa optimalizácia pre vyhľadávače vôbec rieši, ako dosiahnuť čo najlepší výkon, čo všetko naň vplýva a aké nástroje pre analýzu stránok existujú.

Ďalšou oblasťou, ktorú som sa rozhodol zakomponovať do výslednej aplikácie je bezpečnosť webových stránok. V kapitole 2 som najskôr zhrnul aké existujú bezpečnostné riziká a ako sa im dá predchádzať. Potom som popísal niektoré existujúce nástroje pre analýzu bezpečnosti webu a ich možnú integráciu do projektu.

V kapitole 3 je popísaná architektúra aplikácie, dáta, s ktorými bude pracovať a návrhy základných prvkov užívateľského rozhrania. Okrem toho sa na základe prvých dvoch kapitol vybrali prvky na analýzu, ktoré boli neskôr implementované, a externé služby alebo nástroje využité vo výslednej aplikácii.

Implementácia serverovej a klientskej časti, spolu s testovaním, sú popísané v poslednej kapitole 4. Okrem iného je v nej popísaná organizácia zdrojového kódu, implementácia jednotlivých analýz a skenovania stránok webu, alebo nastavenie Docker prostredia.

V kapitole 5 je popísané testovanie, ktoré prebiehalo zároveň s vývojom aplikácie.

Kapitola 1

Optimalizácia webových stránok z pohľadu HTML, CSS a JavaScriptu

V rámci tejto kapitoly sa venujem priebehu komunikácie medzi klientom (prehliadačom) a serverom pri otváraní webových stránok za pomoci HTTP protokolu, ktorý je od skorých 90. rokov minulého storočia štandardom a základným protokolom pri prenose dát na internete. Je popísané ako prebieha parsovanie HTML kódu, CSS štýlov a JavaScriptu, kde pri tom vznikajú medzery vyplývajúce na rýchlosť načítavania stránky a ako ich minimalizovať. Ďalej je rozobraná optimalizácia pre vyhľadávače (SEO) a to, ako dosiahnuť aby sa stránky dostali na popredné miesta vo výsledkoch vyhľadávania, a to hlavne z pohľadu vyhľadávača Google, pretože podľa nových štatistík [23] je celosvetovo s prehľadom na prvom mieste. Mnohé spomínané veci ale platia aj pre ostatné vyhľadávače ako Yahoo či Bing. Moderné prístupy k SEO ponúkajú a množstvo spôsobov ako ich analyzovať a ponúknuť napríklad aj vývojárom webov nejakú spätnú väzbu.

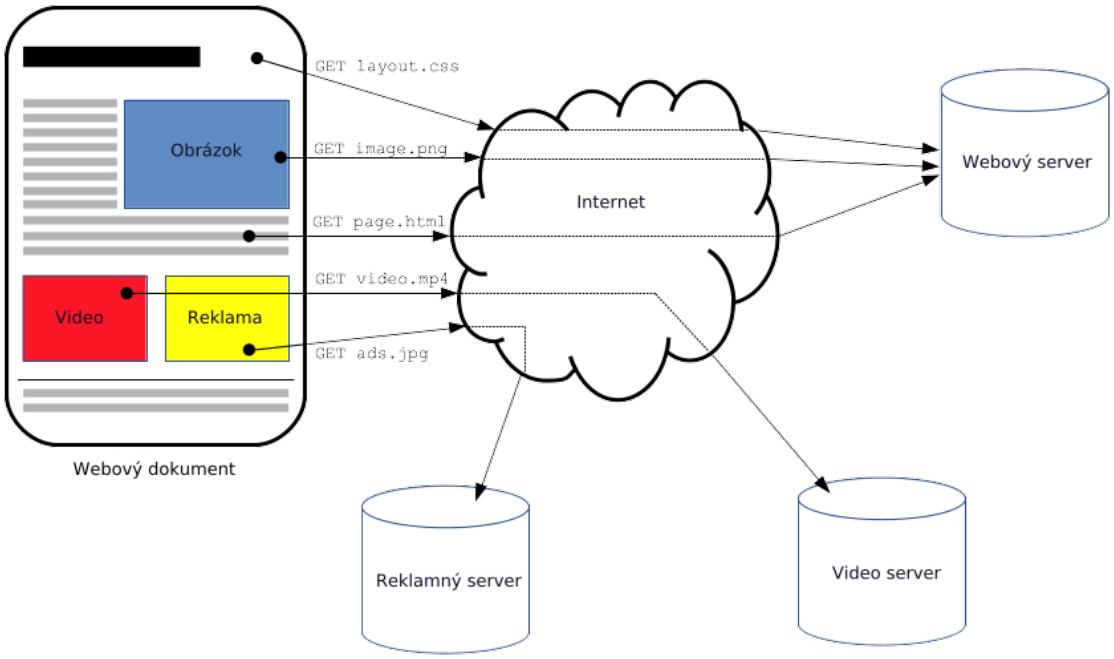
1.1 Úvod k HTTP

HTTP bol navrhnutý ako protokol na prenos zdrojov ako HTML dokumenty, obrázky alebo videá a je základným kameňom prenosu hocijakých dát na internete. Využíva sa pri komunikácií typu klient - server, kde klient zasiela serveru požiadavky (anglicky *requests*) na rôzne dokumenty a server zasiela späť odpoveď (anglicky *response*). Toto je zobrazené na obrázku 1.1. Je to aplikačný protokol odosielaný pomocou TCP, niekedy sa navyše použije TLS zakódované spojenie [4]. V dnešnej dobe sa vo väčšine prípadov používa HTTPS, ktoré pracuje práve nad vrstvou TLS (HTTP over TLS/SSL).

Medzi klientom a serverom v skutočnosti nachádza niekoľko entít, ktoré môžu fungovať napríklad ako cache. Vo všeobecnosti sa nazývajú *proxies*. Klient sa ešte zvykne nazývať termínom *user-agent* a predstavuje hocijaký nástroj, ktorý reprezentuje užívateľa. Môže to byť prehliadač alebo robot, ktorý spracováva stránku kvôli indexovanie pre nejaký vyhľadávač. O nich ale viac v kapitole 1.4.

Aby klient získal od serveru alebo proxy požadovanú stránku, musí spraviť tieto kroky [4]:

1. Otvoriť TCP spojenie. Klient môže vytvoriť nové spojenie, využiť už existujúce alebo otvoriť niekoľko spojení naraz.



Obr. 1.1: **Zobrazenie HTML stránky.** Obrázok v jednoduchosti zobrazuje získavanie dokumentov zo serveru [4].

2. Odoslať požiadavku, ktorá obsahuje kľúčové slovo ako GET alebo POST reprezentujúceho operáciu, ktorú chceme aby server vykonal. Potom nasleduje cesta k zdroju, ktorý od serveru vyžadujeme a nakoniec verzia protokolu HTTP (HTTP/1.1 alebo HTTP/2). Toto sa nachádza na prvom riadku a za ním nasledujú hlavičky. V prípade POST *requestu* môžu za hlavičkami nasledovať ďalšie dáta.

```
GET /index.php HTTP/1.1
HOST: google.com
...
```

3. Spracovať odpoveď serveru, ktorá obsahuje najskôr verziu protokolu HTTP. Potom stavový kód odpovede a nakoniec popis stavového kódu. A podobne ako pri požiadavku je toto na prvom riadku a za ním nasleduje zoznam hlavičiek a potom telo odpovede.

```
HTTP/1.1 200 OK
Server: Apache
Content-Length: 20000
Content-Type: text/html
...
```

4. Uzavrieť spojenie alebo ho použiť na ďalší dotaz.

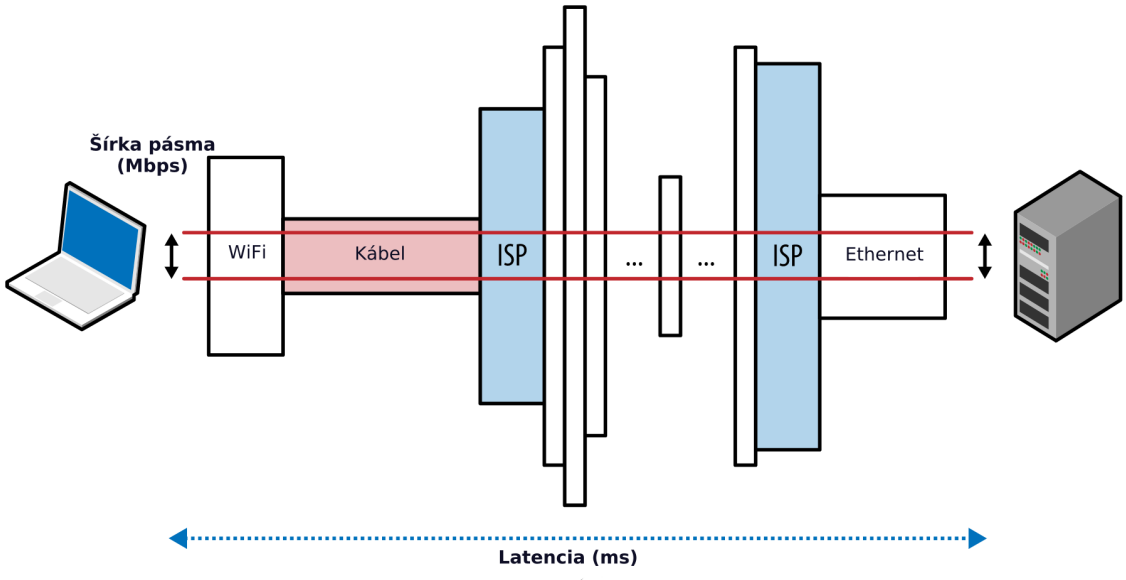
Ide o jednoduchý protokol, ktorý je ľahko čitateľný a umožňuje vývojárom rýchle odhaľovanie chýb. HTTP vo verzii 1.0 umožňuje pomocou jedného spojenia vymeniť iba jednu dvojicu požiadavka - odpoveď. Verzia 1.1 môže byť použitá na spracovanie viacerých takýchto dvojíc jedným spojením [11]. HTTP 2 obaľuje HTTP správy do rámcov, ale princíp fungovania zostáva rovnaký [4].

HTTP je bez-stavový protokol, teda neexistuje žiadne priame prepojenie medzi dvoma rozdielnymi dotazmi. Toto môže spôsobiť problémy pri tvorbe moderných webov. Vyriešené je to pomocou *cookies*, ktoré je možné zadefinovať v hlavičke dotazu alebo odpovede. V dotaze sa vyskytujú v hlavičke „Cookie“. Server môže klientovi indikovať ich vytvorenie v hlavičke „Set-Cookie“. Hlavičky tohto protokolu sú navrhnuté s cieľom spraviť z neho ľahko rozšíriteľný protokol. Klient a server si môžu sami zadefinovať aké hlavičky nad rámec štandardných budú používať a implementovať pomocou tohto nejakú vlastný funkcionalitu.

1.2 Rýchlosť ako vlastnosť aplikácie

Skok v oblasti vývoja softvéru pre web za posledné roky spôsobil zvyšujúce sa nároky na rýchlosť načítavania stránok. Čas načítania webovej stránky (PLT) sa stal kľúčovým faktorom pri posudzovaní výkonnosti stránky a podľa viacerých štúdií vplýva na UX (anglicky *user experience*) a tým ma dopad aj na návštevnosť a profit organizácie. Napríklad Amazon zvýšil zisky o 1 % za každých 100 milisekúnd redukcie v PLT alebo Shopzilla o 12 % redukciou v PLT zo 6 sekúnd na 1.2 sekundy. Z týchto dôvodov je rýchlosť označovaná ako vlastnosť, ktorú treba zákazníkovi dodať a aby bolo toto dodanie úspešné je treba myslieť na viacero faktorov. Práve tým sa budem zaoberať v tejto kapitole.

Prvé dve komponenty (vlastnosti), ktoré spomeniem sú latencia a šírka pásma. Latencia predstavuje čas od odoslania dát (paketu) serverom po prijatie dát klientom. Šírka pásma je zas maximálny počet dát prenášaných po sieti v danú chvíľu. Zobrazené sú na obrázku 1.2.



Obr. 1.2: **Latencia a šírka pásma.** Obrázok zobrazuje latenciu (*latency*) a šírku pásma (*bandwidth*) [32].

Aj keď sú tieto dve komponenty veľmi dôležité, ich analýza v rámci aplikácie, ktorá je cieľom tejto práce by nebola možná. Vyplýva z nich však požiadavka pri tvorbe webových stránok, ktorá sa analyzovať dá, a to je minimalizovanie prenášaných dáta a dodávanie zdrojov (css, obrázky, ...) tak, aby sa čo najmenej blokovali. Ako príklad si môžeme predstaviť, že chceme preniesť 10 Mb súbor cez 1 Mbps linku. Takýto úkon by zabral 10 sekúnd. Keby máme k dispozícií 100 Mbps tak by to zabralo iba 0.1 sekundy.

Vývojári ale vo väčšine prípadov nemajú šírku pásma ako ovplyvniť a teda nezostáva nič iné ako sa pokúsiť čo najmenej zmenšiť veľkosť súborov. Dá sa to dosiahnuť napríklad minifikáciou štýlov a skriptov, odstránením prebytočných znakov (ako napríklad medzery) z HTML kódu alebo kompresiou obrázkov či videí. V dnešnej dobe sa často využíva formát obrázkov webp. Ide o formát, ktorý podporuje stratovú aj bezstratovú kompresiu a dokáže výrazne zmenšiť veľkosť súborov a zrýchliť tak načítavanie stránky. Bezstratové webp obrázky sú oproti png o 26 % a oproti stratovým jpg o 25-34 % menšie. Takisto podporuje transparentnosť a takéto obrázky môžu byť oproti png až 3-krát menšie. Webp je v súčasnosti podporované vo všetkých popredných prehliadačoch ako Google Chrome, Mozilla Firefox, Edge či Opera. Existuje aj kódovacia a dekódovacia knižnica libwebp a nástroje pre príkazový riadok cwebp (kompresia) alebo dwebp (dekompresia).

Minifikácia znamená odstránenie prebytočných prvkov z kódu. Patria sem medzere, komentáre alebo dlhé názvy premenných, ktoré sa nahradia za krátke, často jedno-písmenkové výrazy. Takýto kód je pre ľudí nečitateľný (čo prehliadaču vôbec nevadí) ale poskytuje tú istú funkcionalitu zatiaľ čo dosiahneme zníženie šírky pásma potrebnej na sieťové požiadavky pre stiahnutie súborov. V prípade skriptov môžeme dosiahnuť zníženie veľkosti v krajných prípadoch aj o 60%. Minifikáciou sa stala štandardnou technikou optimalizácie webových stránok a využívajú ju tvorcovia všetkých hlavných JavaScriptových knižníc, ktorý poskytujú okrem štandardných súborov a súbory s príponou min.js. Napríklad takáto verzia knižnice JQuery je o 176 Kb menšia.

Väčšina moderných prehliadačov a serverov podporuje kompresiu gzip, kedy server dokáže automaticky takto zakódovať súbor predtým než ho odošle a prehliadač ho po prijatí dokáže dekódovať. V tabuľke 1.1 môžeme vidieť ako sa líšia pôvodné, minifikované a komprimované verzie niektorých CSS knižníc.

Názov	**Verzia**	**Bez kompresie**	**S minifikáciou**	**S gzip**
Bootstrap	v4.0.0	187 KB	147 KB	20 KB
Material Design Lite	v1.3.0	350 KB	137 KB	21 KB
Semantic UI	v2.2.6	730 KB	550 KB	95 KB
Pure CSS	v0.6.2	80 KB	17 KB	3.8 KB
Milligram	v1.3.0	10 KB	8 KB	2 KB

Tabuľka 1.1: Tabuľka veľkostí CSS knižníc bez kompresie, minifikovaných a s gzip kompresiou. Viac príkladov dostupných na adrese `https://gist.github.com/primaryobjects/64a4e7e3351c646f51eee07949215ad4`.

1.3 Anatómia moderných webových aplikácií

Vývojom v oblasti webových technológií vznikli pri najmenšom tri základné druhy obsahu [32]:

1. **Hypertextové dokumenty**. V dnešnej dobe už takmer neexistujúce, no na počiatku internetu a www išlo o čisto textové stránky s nejakou základnou štylizáciou a podporou pre hypertextové odkazy.

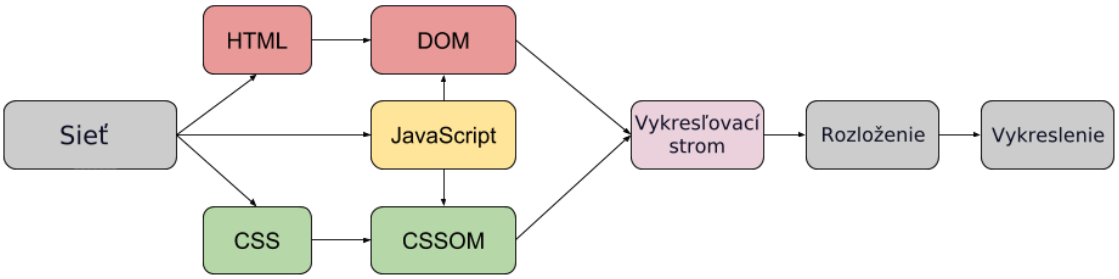
2. **Webové stránky**. Ide o rozšírenie hypertextových dokumentov o podporu obrázkov, videí a iných médií. Umožňujú vytárať rôzne vizuálne bohaté ale zväčša neinteraktívne stránky.

3. **Webové aplikácie**. Ide o pridanie JavaScriptu, dynamického HTML (DHTML) a Ajaxu k jednoduchým webovým stránkam. Takéto aplikácie už dokážu poskytnúť spätnú väzbu pre akcie užívateľa okamžite. Takýto vývoj umožnil vznik prvých plnohodnotných webových aplikácií ako napríklad Outlook Web Access (vznik XMLHTTP podpory v IE5).

`HTTP 0.9` podporovalo požiadavku na jeden dokument, ktorý sa prenášal cez jedno TCP spojenie. Hlavnou metrikou rýchlosti stránok bola preto doba načítania dokumentu.

`HTTP 1.0` a `HTTP 1.1` priniesli spolu s inými inováciami potrebu načítať viacero dokumentov, ktoré tvorili výslednú stránku. To prinieslo možnosť otvoriť viacero TCP spojení a hlavnou metrikou sa stala doba načítania stránky (PLT - *page load time*). PLT je v jednoduchosti čas, za ktorý sa *spinner* indikujúci načítavanie stránky prestane točiť. Pod povrchom ale ide o čas kedy sa spustí `onload` udalosť v prehliadači. Tá sa spustí po tom, čo sa načítajú všetky dokumenty, potrebné na zobrazenie stránky [32].

Táto skutočnosť priniesla potrebu riešiť komplexné grafy závislostí skriptov, štýlov a HTML kódu.



Obr. 1.3: **Vykresľovacia sada inštrukcií (anglicky *pipeline*) prehliadača** [7].

Na obrázku je vidieť z akých procesov sa skladá parsovanie HTML dokumentu. Samotný dokument je sparsovaný do DOM (Document Object Model). Ide o multiplatformové a jazykovo nezávislé rozhranie, ktoré reprezentuje HTML a XML dokumenty v stromovej štruktúre, v ktorej každý uzol a objekt reprezentuje nejaký prvok dokumentu. DOM umožňuje programátorský prístup k prvkom stromu a každý uzol obsahuje aj manažér udalostí.

Zo štýlov a pravidiel v nich obsiahnutých sa tvorí CSSOM (CSS Object Model). Ide o niečo podobné ako DOM, ale v tomto prípade sa manipuluje z CSS namiesto HTML.

Oba tieto prvky sú spojené aby vytvorili vykresľovací strom. Ten obsahuje všetky prvky viditeľné na základe DOM a CSSOM. Napríklad pokiaľ nejaký paragraf bude odpovedať pravidlu `display: none;`, tak sa nezobrazí [33]. Z vykresľovacieho stromu sa môže vypočítať skutočné rozloženie prvkov a stránka sa môže vykresliť.

V tomto momente prichádza do hry JavaScript. Vykonávanie skriptov môže spustiť synchrónne operácie a blokovať tak parsovanie DOM. Podobne môžu skripty požadovať vypočítané štýly objektov, a teda môžu tiež byť blokované čakaním na CSSOM. Parsovanie DOM nemôže pokračovať kým sa nespustí skript, a ten nemôže byť spustený kým sa nedoparsuje CSSOM. Preto sa musia štýly načítať skôr ako samotné skripty. Dosiahne sa to umiestením CSS súborov v HTML kóde na začiatku (najlepšie do hlavičky) a JavaScript súborov alebo skriptov na konci tela stránky.

Alebo sa môže použiť kľúčové slovo `defer`. Keď naň prehliadač narazí, tak pokračuje v parsovaní HTML a skript spracuje na pozadí a spustí až po spracovaní celého HTML. Relatívne poradie takýchto skriptov je zachované a sú spustené tak ako sa vyskytujú v HTML kóde.

Podobný kľúčovým slovom je `async`. Takýto skript sa takisto načíta na pozadí a neblokuje parsovanie DOM, ale na rozdiel od defer je spustený asynchrónne ihneď v okamihu ako sa načíta alebo stiahne. DOM a ani ostatné skripty nečakajú na spustenie takéhoto skriptu, a ani on nečaká na nič iné.

Nasledujúci príklad ukazuje ako sa async a defer používajú:

```
<script defer src="https://example.com/script.js"></script>
<script async src="https://example.com/script.js"></script>
```

Projekt *HTTP archive* [8] zhromažďuje dáta o tom, ako je digitalizovaný obsah na webe konštruovaný a poskytovaný. Je to repozitár dát o výkone webových prvkov ako stránky, neúspešné dotazy alebo použité technológie. Podľa týchto štatistík mala v druhej polovici roku 2021 priemerná desktopová stránka (jedna URL) tieto vlastnosti:

- Na získanie všetkých dokumentov sa prenieslo **2173 KB**.
- Dokopy sa odoslalo **74 požiadaviek**.
- Na **obrázky** sa odoslalo **25 požiadaviek** a prenieslo **967.5 KB**.
- Na **JavaScript** sa odoslalo **22 požiadaviek** a prenieslo **492.3 KB**.
- Na **štýly** sa odoslalo **7 požiadaviek** a prenieslo **73.9 KB**.

Lenivé načítanie obrázkov

Obrázky ako také neblokujú parsovanie DOM ani CSSOM, a teda nemajú vplyv na rýchlosť vykreslenia stránky. Majú ale vplyv na PLT a pokiaľ ich je na stránke veľa, tak môže byť tento vplyv zásadný. V dnešnej dobe azda najpoužívanejšou technikou ako zredukovať čas potrebný na načítanie všetkých obrázkov je tzv. `lazy-loading`. Slovíčko lazy (lenivý) predstavuje taký prístup k nejakej činnosti, pri ktorom ju chceme vykonať až v momente, keď je to naozaj potrebné. V tomto prípade chceme obrázok načítať keď ho užívateľ vidí. Na zabezpečenie tejto funkčnosti sa do nedávna používali rôzne triky za použitia HTML atribútov a skriptov. Väčšina moderných prehliadačov v najnovších verziách (s výnimkou Safari) podporujú atribút `loading` značky `img`, ktorý s hodnotou `lazy` indikuje prehliadaču, že má obrázok načítať, až keď sa reálne zobrazí na obrazovke.

Optimalizácia pre prehliadače

Moderné prehliadače sú viac než len akýsi vykresľovač HTML kódu a manažér soketov. S ich pribúdajúcim množstvom sa výrobcovia snažia dosiahnuť čo najlepší výkon ich prehliadača a poskytujú množstvo optimalizácií, ktoré môžu viesť k rýchlejšiemu načítaniu stránky. Mnoho z týchto prvkov prehliadač vykonáva za nás, bez toho aby sme o tom vôbec vedeli. Je ale niekoľko spôsobov ako mu môžeme pomôcť so zrýchlením našich stránok a identifikovaním kritického obsahu. Existujú štyri techniky, ktoré implementujú skoro všetky prehliadače [32]:

- **Pre-fetching a priorizácia**. DOM, CSSDOM a JavaScript parser-y môžu poskytnúť viac informácií o priorite jednotlivých dokumentov. Blokujúce dokumenty majú priradenú vyššiu prioritu aby sa načítali čo najskôr.
- **DNS pre-resolve**. Často používané doménové mená sa načítajú vopred aby sa vyhlo DNS latencií budúcich HTTP požiadaviek.
- **TCP pre-connect**. Prehliadač môže otvoriť TCP spojenie pokiaľ predpokladá, že bude použité na budúce HTTP požiadavky. Pokiaľ je tento predpoklad úspešný, tak sa môže eliminovať latencia spojená s roundtripom (TCP handshake).
- **Pre-render stránky**. Niektorým prehliadačom môžeme napovedať ďalšiu možnú destináciu a ony si tak môžu túto stránku celú načítať dopredu.

Na zlepšenie identifikácie dokumentov, ktoré potrebujú využiť niektorú z týchto techník existuje tag `link` a jeho atribút `rel`, ktorý môže mať tieto hodnoty:

- dns-prefetch (použitý pokiaľ chceme DNS pre-resolve),
- prerender (pre-render stránky),
- prefetch (pred-načítanie dokumentu pre jeho budúce použitie alebo navigáciu),
- preconnect (použitý na TCP pre-connect).

Tieto hodnoty sú použité iba ako našepkávač prehliadaču a on nezaručuje, že s tým bude niečo robiť. Pokiaľ s nimi ale aj nič neurobí tak sú neškodné a nezhoršia kvalitu ani výkonnosť webu. Príklad ich použitia:

```
<link rel="dns-prefetch" href="//hostname.com">
<link rel="prefetch" href="/velky_obrazok.jpeg">
<link rel="prerender" href="//example.com/clanky/dalsi_clanok">
<link rel="preconnect" href="//example.com">
```

1.4 Vyhľadávače

Keby máme zadefinovať, čo by mal spĺňať ideálny vyhľadávač, tak by sme od neho očakávali aby bol rýchli a poskytoval relevantné informácie.

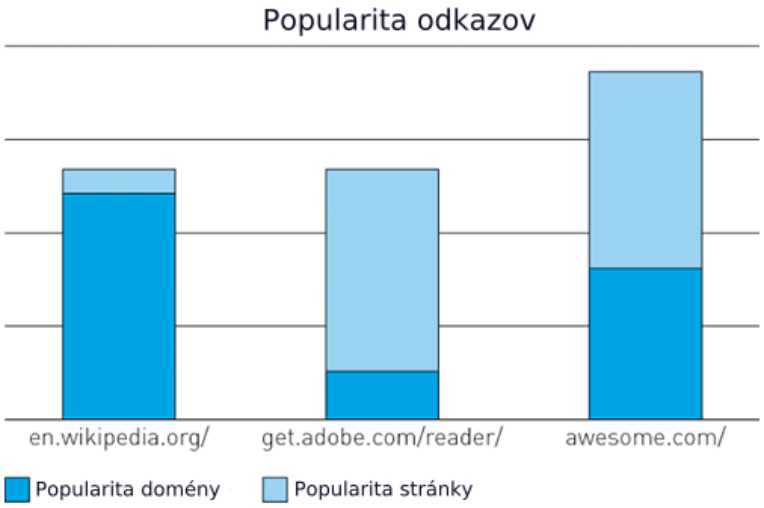
Pokiaľ by sme chceli, aby poskytoval najrelevantnejšie výsledky, tak by vyhľadávač musel mať tým ľudí dostatočne veľký na to, aby prehľadal každý jeden dokument dostupný na

internete, preštudoval ho a zaradil do kategórií. Potom by musel vrátiť presné výsledky pre každý dotaz od užívateľa.

Najrýchlejší vyhľadávač by zas musel byť schopný prehľadať každú URL v momente ako je publikovaná na internete, zaradiť ju do svojho indexu a sprístupniť ju tak pre vyhľadávanie.

Cieľom moderných vyhľadávacích nástrojov ale nie je dať dohromady obrovský tým ľudí, ktorí budú bez prestávky sedieť za počítačom a prezerať obsah internetu. Ich cieľom je vytvoriť algoritmus, ktorý by perfektne kombinoval rýchlosť prehľadávania a indexácie a určoval relevantnosť obsahu. Dnes najpoužívanejší vyhľadávač Google používa algoritmus, ktorý vyvinul jeho zakladateľ Larry Page. Dostal meno po ňom a nazýva sa PageRank. Tento algoritmus najviac využíva silu odkazov. Majiteľ webu síce dokáže ovplyvniť všetko na svojich stránkach (paragrafy, kľúčové slová, vnútorné odkazy, ...), nedokáže však ovplyvniť odkazy na svoju stránku z iných stránok. Táto skutočnosť robí z odkazov výbornú metriku na hodnotenie stránok. Samozrejme algoritmy vyhľadávačov toho robia oveľa viac, toto je jeden zo základných kameňov ich funkčnosti.

Pri hodnotení každej stránky berú vyhľadávače ohľad na vlastnosti samotnej stránky ale aj danej domény. Dvomi faktormi (zo stoviek ďalších), ktoré vplývajú na hodnotu odkazu sú doménová popularita a popularita stránky. Tie sú potom skombinované do jedného faktoru, popularity, ako ukazuje graf na obrázku 1.4. Pre predstavu sa môžeme pozrieť na Wikipédiu, ktorá ma tendenciu byť vo výsledkoch vyhľadávania na prvých miestach pre rôzne kľúčové slová. To je spôsobené tým, že má vysokú doménovú popularitu. Na druhú stranu sa môže stať, že sa na vrchu vyhľadávania objaví nejaká stránka z neznámej domény. Dôvodom toho je zas vysoká popularita samotnej stránky [31].



Obr. 1.4: Graf ukazujúci kombináciu stránkovej a doménovej popularity [32].

Google síce púšťa PageRank stránok do sveta, ale treba brať ohľad na to, že táto hodnota predstavuje iba malú zložku celého algoritmu. Stránka s hodnotou 5 nemusí dostať prednosť pred stránkou s hodnotou 4. Inžinieri z Googlu nechcú aby bol ich algoritmus ľahko prehliadnuteľný pomocou reverzného inžinierstva. Preto sa asi nikdy nedozvieme definitívne metriky. Poznáme iba praktiky, ktoré nám pomôžu zvýšiť PageRank a relevantnosť našich stránok.

Roboti

Predtým než Google alebo akýkoľvek iný vyhľadávač pristúpi k indexácií a hodnoteniu nejakej stránky ju musí objaviť. Robot plní úlohu toho človeka, ktorý by sedel za počítačom, manuálne preklikával stránky a nahadzoval ich do databázy.

Robot, ktorého vypúšťa spoločnosť Google sa nazýva Googlebot. Existuje Googlebot Desktop a Googlebot Smartphone a aký je medzi nimi rozdiel je jasné z ich názvu. Identifikujú sa však rovnako a nedá sa zakázať jedného a druhého povoliť. Úlohou robota je získať obsah dokumentu a predať ho nejakému centrálnemu stroju k analýze. Najskôr je vypustený do sveta tým, že začne prehľadávať nejaké stránky (ich výber). Snaží sa hľadať odkazy na obrázky, videá alebo iný obsah. Dôležité sú preň ale dôkazy na iné stránky, pretože tie sú cestou k tomu, aby objavil nový obsah internetu.

Existujú spôsoby ako zakázať robotom prehľadávať konkrétne stránky alebo im napovedať, či majú nasledovať nejaký odkaz. Ale o tom a ďalších technikách ako optimalizovať web pre vyhľadávače (hlavne pre Google) sa píše v nasledujúcej sekcii.

1.5 SEO

SEO (Search Engine Optimization, po slovensky optimalizácia pre vyhľadávače) je proces, pri ktorom sa snažíme zlepšiť viditeľnosť stránok, keď ľudia používajú Google, Bing alebo iný podobný nástroj na objavenie obsahu, ktorý sa priamo či nepriamo týka nejakého produktu alebo služby v rámci našich stránok. Čím lepšia je viditeľnosť, tým skôr získame pozornosť potenciálnych zákazníkov. V tejto kapitole sa pokúšam prejsť cez rôzne techniky ako dosiahnuť tento cieľ.

Kanonické URL

Jednou z najčastejších SEO chýb, ktoré sa vyskytujú je chýbajúca kanonizácia. V tomto kontexte to predstavuje koncept, kedy si vyberieme autoritatívnu verziu URL a propagujeme jej použitie pred ostatnými variantami [31]. Predstavme si tieto adresy, ktoré by mohli predstavovať tú istú stránku:

```
https://www.example.com/articles?id=123
https://www.example.com/articles/123
```

Prečo by sme sa chceli tomuto vyhnúť? V sekcii 1.4 sme sa venovali popularite odkazov. V tomto prípade je problémom konkrétne popularita stránok, ktorá sa rozdeľuje medzi dve adresy. Lepšie výsledky by sme dosiahli použitím jednej formy, ktorá by mala súčet týchto dvoch popularít.

Jedným so spôsobov ako sa tomuto problému vyhnúť je použiť presmerovanie. Existujú dva typy presmerovaní:

- 301 indikuje permanentné presmerovanie. V praxi je toto presmerovanie používanejšie, pretože predáva tzv. ranking power odkazu. Teda vyhľadávače je to silnejšia indikácia kanonizácie.
- 302 indikuje dočasné presmerovanie. Toto sa používa iba za predpokladu, že sa v budúcnosti presmerovanie odstráni a adresa bude predstavovať samostatný obsah.

Presmerovanie môžeme dosiahnuť pomocou pravidiel v `.htaccess` súbore, napríklad v PHP kóde, JavaScript kóde alebo môžeme použiť HTML tag
`<meta http-equiv="refresh" ...>`:

```
<meta http-equiv="refresh" content="0; url=https://example.com/">
```

Posledné dve možnosti sa ale nedoporučujú pretože pri nich prehliadač nedokáže garantovať permanentné presmerovanie [18]. Štandardom W3C je odporúčané nepoužívať meta tag.

Ďalším riešením kanonizácie je použiť kanonický tag. Tento tag vyhľadávaču napovie, ktorú adresu chceme zobraziť vo výsledkoch vyhľadávania [9]. Ide o HTML tag `link`, s ktorým sme sa už stretli. V tomto prípade by mala atribút `rel="canonical"`. Teda napríklad:

```
<link rel="canonical" href="https://example.com/">
```

Treba dávať pozor aj na URL adresy, ktoré vyzerajú veľmi podobne:

```
https://www.google.com
https://www.google.com/
https://google.com
https://google.com/
http://google.com
http://google.com/
http://www.google.com
http://www.google.com/
```

Aj v tomto prípade treba dbať na kanonizáciu, no tu je najlepšie použiť presmerovanie. Jednoznačne treba presmerovať http na https, a to nie len z bezpečnostných dôvodov, ale aj kvôli skutočnosti, že Google už v dnešnej dobe preferuje stránky s týmto protokolom. To či presmerovať na www alebo bez www, s lomkou alebo bez nej je v podstate jedno. Hlavne si treba vybrať jednu formu a tej sa držať na celom webe.

Robots.txt

Súbor `robots.txt` je jeden zo základných krokov, ktoré môže webmaster urobiť, keď pracuje s vyhľadávačmi. Ide o textový súbor, ktorý by mal byť v koreňovom adresári projektu a čitateľný z vonku. Je založený na informačnom protokole, ktorý hovorí robotom, ktoré adresáre a súbory sú im sprístupnené, a ktoré nie. Predtým než robot začne spracovávať hocijaký dokument z webu, tak vyskúša stiahnuť tento súbor a nájsť pravidlá, ktoré platia pre neho a pre daný dokument. Pravidlo, ktoré zakáže všetkým robotom prehľadávať adresár `tmp` môže vyzerať takto:

```
User-agent: *
Disallow: /tmp/

Sitemap: http://www.example.com/sitemap.xml
```

Prvý riadok identifikuje robotov, pre ktorých toto pravidlo platí. V tomto prípade zahŕňame všetkých robotov. Keby chceme iba Googlebota, tak namiesto znaku `*` použijeme `Googlebot`. Potom musí nasledovať aspoň jedno pravidlo (`Disallow` alebo `Allow`), ktorým zakážeme alebo povolíme adresár alebo súbor. Existuje aj pravidlo `Sitemap`, ktoré obsahuje kompletnú URL súboru so sitemapou 1.5. O robotoch sa môžete dočítať viac tu [12].

Sitemapa

Mapa stránok obsahuje všetky stránky, videá, obrázky a iné dokumenty webu a vzťahy medzi nimi. Vyhľadávače tento súbor využívajú na efektívnejšie prehľadávanie webu. Obsahuje informácie o tom, ktoré stránky sú dôležité a malo by sa na ne zamerať. Taktiež záznamy v sitemape môžu obsahovať informácie o tom kedy bola stránka naposledy aktualizovaná alebo v akom je jazyku, dĺžku videa, typ obsahu a podobne. Sitemapa je výhodná v týchto prípadoch:

- Ide o rozsiahli web a mohlo by sa stať, že robot niektoré novovytvorené alebo aktualizované stránky prehliadne.
- Web obsahuje množstvo stránok, ktoré nie sú dobre odkazované.
- Web je nový a nie je odkazovaný z externých zdrojov.
- Web obsahuje množstvo médií.

Naopak menej výhodná alebo až nepotrebná je v týchto prípadoch:

- Ide o malý web (do 500 stránok, ktoré chceme aby boli indexované).
- Web je dobré „prelinkovaný“. To znamená, že sa dá z domovskej stránky dostať na každú podstránku.
- Web neobsahuje množstvo médií.

Viac o mape stránok sa môžete dočítať tu [14].

Odkazy a štruktúra webu

URL jednotlivých podstránok by mali byť čo najjednoduchšie a mali by čo najlepšie vystihovať obsah a hierarchickú štruktúru webu. Namiesto číselných ID sa odporúča používať nejaké popisné kľúčové slovo. Neodporúča sa používať veľa *query* parametrov v adrese, pretože môže vzniknúť veľa stránok s podobným obsahom, robot tak strávi veľa času indexovaním takmer totožných stránok a môže vynechať iné, dôležitejšie.

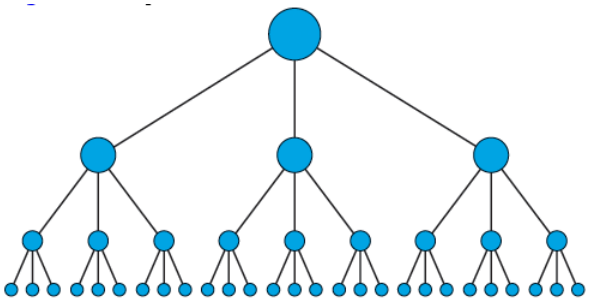
```
Zlá URL:
http://www.example.com/summer_clothing/filter?color_profile=dark_grey

Dobrá URL:
http://www.example.com/summer-clothing/dark-grey
```

Pre niektoré odkazy by sme mohli chcieť Googlu nešeptať, aký je náš vzťah s odkazovaným dokumentom. K dosiahnutiu tohto sa používa `rel` atribút značky `<a>`. Využite to má hlavne v prípade odkazov na iný než nás web.

- `<a rel="sponsored" ...>` označuje odkaz na sponzorovaný obsah alebo nejakú platenú reklamu. Hodnota *nofollow* je tiež akceptovateľná ale odporúča sa použitie *sponsored*.
- `<a rel="nofollow" ...>` označuje odkaz, pri ktorom nechceme aby ho robot nasledoval.

- `<a rel="ugc" ...>` označuje užívateľom vygenerovaný obsah (*user-generated content*). Jedná sa napríklad o odkazy v komentároch na diskusnom fóre.
- `<a rel="noreferrer" ...>` dáva prehliadaču inštrukciu aby z ďalšieho dotazu odstránil hlavičku `Referer` a neposkytol tak informácie o sprostredkovateľovi odkazu.
- `<a rel="noopener" ...>` dáva prehliadaču inštrukciu aby pri navigácií na novú stránku neposkytol informácie o kontexte dokumentu, z ktorého sa daný odkaz otvoril. V novom otvorenom okne nebude dostupný parameter `window.opener`. Podobný cieľ sa dá dosiahnuť nastavením `target="_blank"` atribútu odkazu.
- Hodnoty sa dajú aj kombinovať. Napríklad `<a rel="noopener noreferrer" ...>`.

Pokiaľ chceme robotom zabrániť prechádzaniu interných odkazov, mali by sme použiť `robots.txt`. Vyhľadávač počíta pri hodnotení stránky aj interné odkazy, aj keď nezvyšujú PageRank tak ako ich externé verzie. Mali by sme sa aj tak snažiť aby každá stránka obsahovala čo najviac odkazov na iné stránky v rámci webu. Dôležitý je odkaz na domovskú stránku v hlavičke a naopak, aby na domovskej stránke bolo čo najviac odkazov na podstránky. Pri tomto treba myslieť aj na architektúru webu, kedy by mala v ideálnom prípade vzniknúť akási pyramídová štruktúra, kde na vrchole je práve domovská stránka. Podobne ako na obrázku 1.5.



Obr. 1.5: Optimálna štruktúra webu [31].

Pokiaľ je web dostatočne veľký tak by mal byť jeho obsah rozdelený do podkategórií. Tieto by mali byť odkazované z domovskej stránky. Ak to veľkosť webu umožňuje tak by mal byť rozdelený ešte aj do podkategórií a tie by mali byť odkazované minimálne s nadradenej kategórie. Na konci pyramídy by mali byť obsahové stránky. Stránky kategórií a podkategórií by mali spĺňať nasledujúce:

- byť užitočné pre užívateľa,
- odkazovať na priame podkategórie,
- mať dostatok obsahu aby boli dobre indexované.

Stránky s obsahom

Obsahové stránky sú hlavným lákadlom pre robotov. Sú dôvodom prečo ľudia navštevujú náš web a mali by v nich zanechať dojem, že im poskytli to, čo hľadali. Tieto stránky by mali byť špecifické a relevantné pre danú tému.

V hlavičke by mali obsahovať značku `<title>`, v ktorej by mal byť dostatočne unikátny názov stránky vzhľadom na celý obsah webu. Zároveň by mal dostatočne vystihovať podstatu stránky. Titulok je jedným z najdôležitejších prvkov a zároveň najjednoduchších SEO nástrojov. Navyše sa jeho text nepoužíva iba vo výsledkoch vyhľadávania, ale aj pri odkazovaní cez rôzne sociálne siete. Vyhľadávače väčšinou odstrihnú tento text na 65 znakov. Odporúčaná dĺžka je 35 až 40 znakov [31].

Samotný textový obsah by mal byť dostatočne dlhý (aspoň 150 slov) a obsahovať čo najviac a čo najčastejšie kľúčové slová vystihujúce stránku. V dobe písania tejto práce už boli prehliadače schopné detegovať zneužívanie kľúčových slov a dávať takýmto stránkam nižie ohodnotenie.

Pri písaní obsahu je dôležité dodržať hierarchiu viacúrovňových nadpisov. Nadpisy úrovne 1 až 6 boli v minulosti základom štruktúry celej stránky. Vývojom v oblasti webu ale stratili na sile a vyhľadávače im ani nedávajú veľkú hodnotu. Stále sú však odporúčané pretože zlepšujú čitateľnosť a orientáciu užívateľom.

Dôležitým prvkom obsahových stránok sú média v podobe obrázkov, videí, IFramov alebo Flash prvkov. Obrázky sú jedným zo spôsobov ako môžu ľudia vizuálne objaviť nejaký web. Pridanie kontextu k nim môže viesť k väčšej návštevnosti. Obrázok na stránke by mal vystihovať jej obsah. Mali by byť v dobrom rozlíšení, názov súboru (*filename*) by ich mal vystihovať a mali by sa v texte nachádzať blízko relevantného obsahu. `Alt` text obrázku zlepšuje prístupnosť pre ľudí, ktorí ich nemôžu vidieť (používajú čítačku alebo majú pomalé pripojenie). Taktiež pomáha Googlu ich lepšie pochopiť a poskytnúť pri vyhľadávaní lepši kontext. Dobrý alt text by mal obsahovať kľúčové slovo alebo slová, ktoré popisujú čo sa na obrázku nachádza. Napríklad:

```
<img src="pes.jpg" alt="pes"/>
```

Najlepšou hodnotou je stručný ale výstižný popis toho čo obrázok zachytáva:

```
<img src="pes.jpg" alt="Aportujúci pes s~palicou v~zuboch."/>
```

Pozor ale na fakt, že Google indexuje HTML obsah stránky, nie CSS. Nie je teda dobrou praktikou umiestňovať dôležité obrázky pomocou divu a CSS parametru `background-image`. Google podporuje obrázky typov BMP, GIF, JPEG, PNG, WebP, a SVG. Taktiež nie je zlé použiť Data URI obsahujúcu do *base64* zakódovaný obrázok. Takéto obrázky dokonca aj znižujú počet HTTP dotazov, no zvyšujú veľkosť samotného HTML dokumentu:

```
<img src="[data]">
```

Málo kedy sa už v dnešnej dobe stane, že sa nenavrhne web responzívne. Existujú dve možnosti ako zadefinovať obrázky, ktoré sa prispôsobia veľkosti okna. Použiť atribúty `srcset` a `sizes` alebo tag `<picture>` [10].

```
<img srcset="example-320w.jpg 320w,
   example-480w.jpg 480w,
   example-800w.jpg 800w"
sizes="(max-width: 320px) 280px,
   (max-width: 480px) 440px,
   800px"
src="example-800w.jpg" alt="...">
```

Srcset zadefinuje zoznam obrázkov, ktoré môže prehliadač použiť a ich veľkosť. Sizes zase definuje, ktorý sa má použiť pri určitej veľkosti okna.

```
<picture>
 <source type="image/svg+xml" srcset="example.svg">
 <source type="image/webp" srcset="example.webp">
 <img src="example.png" alt="...">
</picture>
```

Tag `<picture>` je výhodný pokiaľ máme k dispozícií obrázok v rôznych formátoch a niektorý prehliadač alebo zariadenie by niektorý z nich nemuseli podporovať. Zoznam obrázkov je v tagoch `<source>`. Odporúča sa použiť tag `<img>`, ktorý obsahuje fallback obrázok.

Na popularite v poslednej dobe získavajú aj videá. Aby bolo video maximálne viditeľné, Google odporúča vytvoriť preň samostatnú stránku [24]. Aby sme zabezpečili čo najlepšiu indexáciu je dobré špecifikovať atribút `poster` pokiaľ je použitý tag `video`. Ide o obrázok, ktorý sa zobrazí ako náhľad videa. Potom napríklad v sitemape špecifikovať tag `<video:thumbnail_loc>`.

Použitie posledných dvoch spomenutých spôsobov použitia médií na webe (IFrame a Flash) sa už v dnešnej dobe neodporúča. Google nezaručuje, že obsah IFramov bude indexovaný. Ak je ich použitie nevyhnutné, je dobré na stránke použiť aj textový odkaz na obsah daného IFramu. Flash nie je odporúčaný z toho dôvodu, že už dlhšiu dobu nie je podporovaný. Namiesto neho by sa mali využiť napríklad funkcie HTML5 [20].

Meta značky

Meta značky sú skvelým spôsobom ako vyhľadávačom poskytnúť viac informácií o stránke. Nie všetky meta značky sú podporované každým vyhľadávačom a nie každý vyhľadávač podporuje rovnaké meta značky. Z predchádzajúceho obsahu sa dá vydedukovať, že sa zameriam na tie, ktoré sú podporované Googlom [16] a zároveň na tie, ktoré ešte neboli spomenuté v širšom kontexte. Meta značky by mali byť v sekcii `<head>`.

`<meta name="description" content="..."/>` sa používa na krátke opísanie obsahu stránky. V niektorých prípadoch sa tento popis zobrazuje vo výsledkoch vyhľadávania a môže ľuďom pomôcť rozhodnúť sa či stránku navštívia alebo nie. Dobrou praktikou je aby nebol dlhší ako 150 znakov a mal aspoň 120 znakov. To by malo zaručiť, že sa zobrazí v peknej forme ako na desktope tak na mobile.

`<meta http-equiv="Content-Type" content="...; charset=..." />` alebo `<meta charset="..." >` určujú typ obsahu dokumentu alebo sadu znakov. Odporúča sa používať `Unicode/UTF-8` všade, kde sa dá.

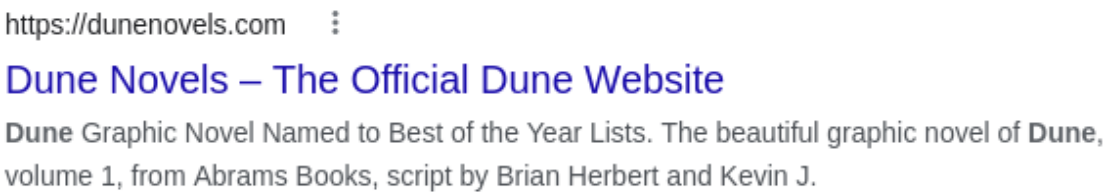
`<meta name="viewport" content="...">` hovorí prehliadaču ako má vykresliť stránku na mobile.

`<meta name="keywords" content="...">` obsahuje čiarkou oddelené kľúčové slová. V minulosti sa táto značka zneužívala na spam slov, ktoré s obsahom stránky nemali nič spoločné. Viacero vyhľadávačov ako Google, Yahoo či Bing preto túto značku vo väčšine prípadov ignorujú.

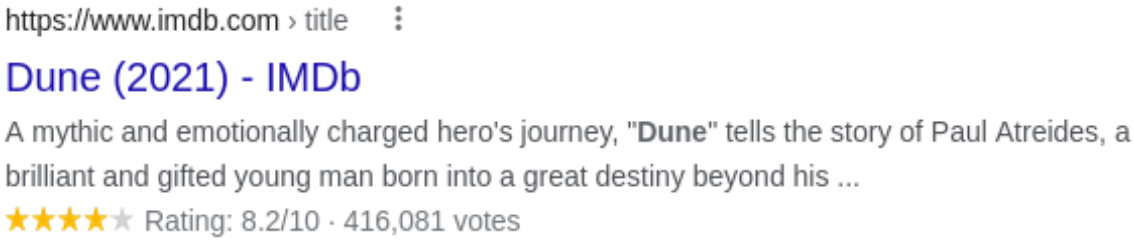
Rich snippets, *rich results* a štruktúrované dáta

Výsledok vyhľadávania sa nazýva *snippet*. V podstate je to modrý odkaz s názvom stránky a tmavým popisom. Pokiaľ sa k tomu pridá nejaký obrázok, recenzie, dĺžka trvania videa,

filmu alebo prípravy receptu, tak to pomenúvame názvom *rich snippet*. V skratke ide je to klasický *snippet* obohatený o nejaké ďalšie dáta. Rozdiel medzi nimi je viditeľný na obrázkoch 1.6 a 1.7.



Obr. 1.6: **Klasický snippet**, ktorý je výsledkom hľadania pre film Duna.



Obr. 1.7: **Rich snippet**, ktorý je výsledkom hľadania pre film Duna.

Na druhom obrázku môžeme vidieť celkové hodnotenie filmu, počet hodnotení a nad názvom stránky taktiež *breadcrumbs* (sekundárna navigácia, ktorá pomáha s orientáciou v rámci webu).

Google už ale nie je len vyhľadávač, ktorý zobrazí zoznam odkazov s krátkym popisom (alebo ako sme už zistili aj s dodatočnými dátami). Na stránke vyhľadávania dokáže zobraziť recepty, rôzne kartičky napríklad osobností, filmov, produktov alebo podnikov, FAQs a podobne. Tieto výsledky sa nazývajú *rich results*. Väčšinou majú krajší a lákavejší grafický dizajn, a preto sú veľkým zdrojom návštevníkov.

Aby vyhľadávače dokázali takýmto spôsobom zobraziť obsah nášho webu, musíme použiť `Schema`. Je to slovník, ktorý dokáže robotom lepšie opísať stránku pomocou štruktúrovaných dát. Pomocou toho potom dokáže vyhľadávač vytvoriť *rich result*. Definícia takýchto dát môže vyzerať takto [21]:

```
<script type="application/ld+json">
   {
     "@context": "https://schema.org",
     "@type": "Article",
     "mainEntityOfPage": {
       "@type": "WebPage",
       "@id": "https://yoast.com/"
      },
      "headline": "WordPress SEO: The definitive guide",
      "image": "https://yoast.com/wordpress_seo_definitive_guide.png",
      "datePublished": "2020-02-05T08:00:00+08:00",
```

```
      "dateModified": "2020-02-05T09:20:00+08:00",
      "author": {
        "@type": "Person",
        "name": "Joost de Valk"
      },
       "publisher": {
        "@type": "Organization",
        "name": "Yoast",
        "logo": {
          "@type": "ImageObject",
          "url": "https://yoast.com/yoast_logo.png"
    }
  }
}
</script>
```

Treba si všimnúť tag `<script type="application/ld+json">`, ktorý sa umiestňuje do `<head>` HTML dokumentu. Obsah tagu je potom JSON obsahujúci dôležité informácie o objektoch na stránke. Môžeme takto reprezentovať celú stránku, článok, obrázky, videá a oveľa viac. Nebudem sa púšťať do opisu jednotlivých parametrov a ich hierarchie, pretože to by vydalo aj na samostatnú prácu. Ak vás zaujíma viac môžete navštíviť dokumentáciu na adrese `https://schema.org/docs/documents.html`.

OpenGraph

OpenGraph protokol umožňuje stránke stať sa rich (bohatým) objektom. To znamená, že by táto stránka mohla mať na Facebooku rovnakú funkcionalitu ako iné objekty Facebooku. Aby sme toto dosiahli musíme vložiť ďalšie meta značky do hlavičky HTML kódu. Najčastejšie používané značky sú:

- `og:title` je titulok objektu a v podstate je to názov stránky. Môže to byť to isté ako obsah značky `<title>`.
- `og:type` je typ objektu. Špecifikuje aký typ obsahu je na stránke.
  Napríklad `video.movie`.
- `og:image` obrázok, ktorý by mal vystihovať obsah stránky.
- `og:url` by mala byť kanonická adresa stránky.
- `og:description` krátky popis obsahu stránky. Môže to byť to isté ako meta description.

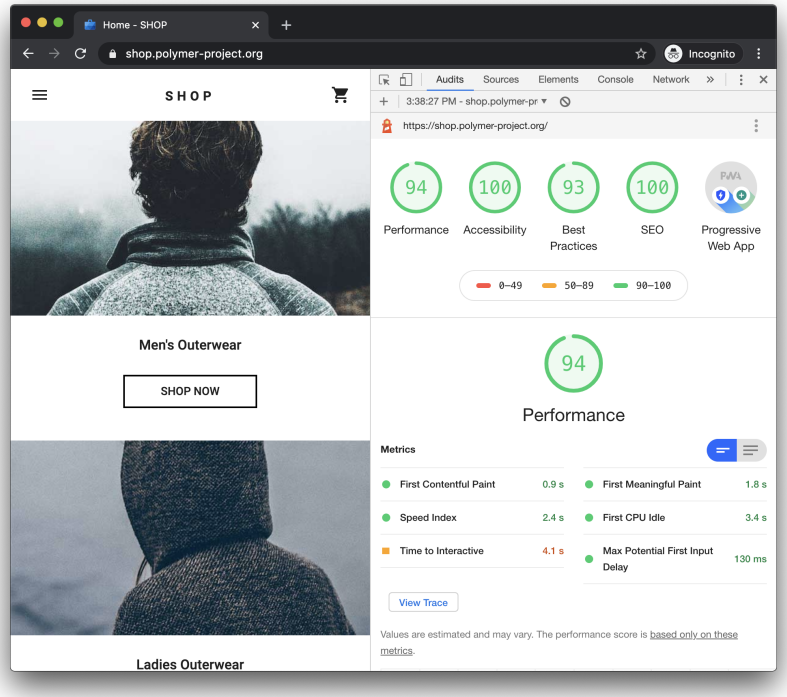
Táto sekcia sa síce netýka priamo SEO, no v dnešnej dobe, keď svetu vládnu sociálne siete, nemôžu ani vyhľadávače ignorovať tieto meta značky a ich použitie môže viesť aj k lepšiemu hodnoteniu stránok. Ďalšie značky a viac o ich použití nájdete na stránkach `https://ogp.me/`.

1.6 Existujúce nástroje

V tejto sekcii sú priblížené nástroje, ktoré sa dajú využiť na analýzu stránok z hľadiska výkonnosti a SEO. Vybral som dva od spoločnosti Google, ktoré sú vhodné na zakomponovanie do výsledného produktu práce.

Lighthouse

Lighthouse je open-source nástroj na zlepšovanie kvality stránok. Zahŕňa testy výkonnosti, prístupnosti, SEO a ďalších užitočných vecí, ako napríklad testy pre progresívne webové aplikácie (PWA). Je to dostupný v prehliadači Google Chrome v sekcii Chrome DevTools, dá sa spustiť v terminály alebo ako Node modul. Spracováva zadanú URL, na ktorej spustí sériu auditov. Výsledok ukazuje ohodnotenie aplikácie v daných oblastiach a vysvetlenie nedostatkov, prípadne ako ich vyriešiť. Existujú viaceré analytické aplikácie, ktoré tento nástroj využívajú, napríklad `https://pagespeed.cz/`.



Obr. 1.8: Výsledný report Lighthouse v Chrome DevTools

Pre spustenie testov v terminály je potrebné nainštalovať Google Chrome a potom Lighthouse ako Node balíček.

```
npm install -g lighthouse

#spustenie
lighthouse <url> --output json
```

Možnosťou `--output json` nastavíme, že chceme výsledný report vo forme JSON. Z toho je potom možné vyčítať rovnaké dáta ako sa zobrazujú v prehliadači. Viac sa o tomto nástroji dočítate na stránkach Google [15].

Search Console

Ide o ďalší nástroj od spoločnosti Google. Je zameraný na zber dát o výsledkoch vyhľadávania pre zadanú doménu. Základné dáta, ktoré zobrazuje sú:

- Počet kliknutí na stránku vo výsledkoch vyhľadávania.
- Počet zobrazení stránky vo výsledkoch vyhľadávania. Ide o zobrazenie v HTML kóde. Teda užívateľ nemusí na konkrétny obsah zaskrolovať ale musí sa aspoň dostať na stránku, na ktorej je odkaz. Teda pokiaľ bude odkaz na druhej strane a užívateľ si ju nezobrazí, tak sa to nezapočíta.
- CTR (Click Through Rate). Ide o číslo od 0 do 1, ktoré vyjadruje ide o počet kliknutí vydelený počtom zobrazení. Pokiaľ je toto číslo nízke, zrejme treba zapracovať na reprezentácií stránok.
- Priemernú pozíciu vo výsledkoch vyhľadávania.

Všetky tieto dáta sa dajú získať pomocou API, ktorá umožňuje filtrovanie podľa dátumu, krajiny, zariadenia, kľúčových slov a ďalších parametrov. Na jej použitie je ale potrebné najskôr zaregistrovať doménu v aplikácií a potom zaregistrovať aplikáciu v Google API Console. Všetky potrebné informácie sú opäť dostupné na stránkach Googlu [22].

Okrem agregácie dát o vyhľadávaní, Search Console dokáže pracovať aj s mapu stránok. Taktiež je možné vynútiť indexovanie konkrétnej stránky. API spomínané vyššie podporuje aj prácu s mapami stránok (nahrať, zmazať, vypísať). Taktiež si môžeme vypísať všetky weby (domény), ktoré ma užívateľ v Search Console zaregistrované.

Ďalšou užitočnou funkčnosťou je kontrola, či je stránka na zadanej URL optimalizovaná pre mobilné zariadenia (anglicky *mobile-friendly*). K tomuto taktiež existuje aj API *endpoint*, ktorý kontroluje zadanú URL [17]. Výstupom je JSON, ktorý obsahuje informáciu, či je stránka *mobile-friendly*. Pokiaľ nie je, tak aj napovie, aké problémy sa na nej vyskytujú:

- MOBILE_FRIENDLY_RULE_UNSPECIFIED (neočakávaná chyba),
- USES_INCOMPATIBLE_PLUGINS (používajú sa pluginy nekompatibilné s mobilným zariadením),
- CONFIGURE_VIEWPORT (viewport nie je zadefinovaný pomocou meta značky),
- FIXED_WIDTH_VIEWPORT (viewport je zadefinovaný na fixné rozmery),
- SIZE_CONTENT_TO_VIEWPORT (obsah stránky nesedí do viewportu),
- USE_LEGIBLE_FONT_SIZES (veľkosť fontu je malá pre čítanie na mobile),
- TAP_TARGETS_TOO_CLOSE (elementy, na ktoré sa dá klikať prstom sú príliš blízko pri sebe).

Tento API endpoint je zatiaľ iba v BETA verzii, no aj tak môže priniesť zaujímavé informácie.

Search Console toho dokáže ešte viac, ako napríklad kontrolovať štruktúrované dáta, ktoré sú základom pre bohaté výsledky. Spomenul som však hlavne veci, ku ktorým sa podľa dokumentácie [22] dá pristupovať pomocou API.

Kapitola 2

Bezpečnostné riziká webových aplikácií

Webové aplikácie ponúkajú užívateľom prístup k množstvu služieb a dát. Ich popularita rastie okrem iného aj vďaka pohodlnému prístupu z hocikade, multiplatformovej kompatibilite a rýchlemu vývoju. Mnohé moderné technológie a prístupy k vývoju tomu taktiež napomáhajú. Toto však láka aj mnohých hackerov, ktorý sa chcú dostať ku kritickým dátam alebo zneužiť nejaké služby. Takéto narušenie môže mať ťažké ekonomické, legálne, etické alebo mnohé iné následky. Webové aplikácie sa stali zdrojom najviac bezpečnostných narušení v oblasti IT [36].

Testovanie alebo analýza zabezpečenia webových aplikácií je preto jedna z kľúčových vecí. V dnešnej dobe sa dá využiť množstvo online nástrojov. Niektoré z nich sú zadarmo, niektoré platené a iné zas zadarmo poskytnú informácie o bezpečnostných nedostatkoch a za poplatok vám ich pomôžu vyriešiť. Treba mať na pamäti, že žiadny nástroj nezaručí 100 % presnosť. Toto platí najmä pre tie, ktoré nebežia v prostredí, z ktorého by mali priamy prístup na server, ktorý má byť testovaný (toto platí aj pre tento projekt). Skener síce slúži ako užitočný nástroj, no nikdy neobjaví všetko. Tiež sa môže stať, že skener objaví nejaký bezpečnostný nedostatok, ktorý v skutočnosti neexistuje [35].

Predtým, než sa posunieme ďalej, je potrebné ujasniť tri základné pojmy súvisiace všeobecne s IT bezpečnosťou.

Aktívum (Asset)

Aktívum je niečo, čo sa snažíme ochrániť. Môže ísť o ľudí, majetok, či všeobecne o nejaké informácie. Tie zahŕňajú databázu, zdrojový kód, kritické záznamy spoločnosti, dáta o užívateľoch a podobné nehmotné položky.

Hrozba (Threat)

Hrozba je hocičo, čo dokáže využiť nedostatky (zraniteľné miesta) systému, aby sa zmocnilo nejakého aktíva, poškodilo ho alebo úplne zničilo. Teda je to niečo, proti čomu sa chceme chrániť (hacker, užívateľ systému, nestabilita systému).

Zraniteľnosť (Vulnerability)

Zraniteľnosť je slabé miesto v implementácií, dizajne alebo celkovo vo fungovaní systému, ktoré môže byť využité hrozbou na získanie neautorizovaného alebo neoprávneného prístupu k aktívam.

Riziko (Risk)

$$Risk = Asset + Threat + Vulnerability$$

Riziko je spojenie predošlých pojmov dokopy. Teda je to udalosť, kedy nejaká hrozba využije zraniteľnosť aby sa zmocnila aktíva. Dá sa povedať, že hrozba môže existovať, ale pokiaľ neexistuje zraniteľné miesto, tak by nemalo existovať riziko. Podobne to môže byť aj opačne, keď existuje zraniteľné miesto, ale neexistuje hrozba.

2.1 Hrozby pre moderné webové aplikácie

Hrozieb pre webové aplikácie je množstvo a rôzne zoznamy sa dajú nájsť hocikde na internete. Vždy je ale ťažké zahrnúť všetky. Problém je v tom, že autor takéhoto zoznamu nedokáže presne odhadnúť štruktúru aplikácie, ktorá by mala byť v ohrození. Preto aj analyzátor, ktorý má pracovať mimo prostredia aplikácií a má byť univerzálne použiteľný na rôzne systémy, nedokáže predchádzať všetkým hrozbám, ktoré existujú pre konkrétnu aplikáciu. Pokiaľ chcem otestovať nejaký softvér čo najviac do hĺbky, je potrebné oprieť sa o viacej zoznamov hrozieb a zahrnúť rôzne spôsoby ich testovania alebo rôzne nástroje. Cieľom tohto projektu taktiež nie je vytvoriť nástroj, ktorý vám zaručí, že testovaná aplikácia nemôže byť nijako napadnutá, ale zanalyzuje vašu aplikáciu z pohľadu najčastejších hrozieb, ktorým sa dá pomerne jednoducho predchádzať. V tejto sekcii prejdem cez niektoré z týchto hrozieb [35].

Buffer overflow

Ide o jedu z najznámejších hrozieb v oblasti softvérovej bezpečnosti. Nastáva, keď sa program pokúša uložiť do vyrovnávacej pamäte viacej dát, ako sa tam zmestí alebo keď sa snaží uložiť dáta do pamäte mimo bufferu [35]. Zapisovanie mimo bloku alokovanej pamäte môže viesť k poškodeniu dát, spadnutiu programu alebo spusteniu škodlivého softvéru.

Táto zraniteľnosť sa dá ťažko odhaliť a vo všeobecnosti je ju aj ťažké zneužiť. Na úrovni zdrojového kódu sa často jedná o chyby v programátorovej práci, keď sa nesprávne pracuje s pamäťou. Napríklad pri zlom použití funkcie `strncpy` v jazyku `C\C++` [29]. V oblasti webov môže ísť o nesprávnu kontrolu limít rôznych vstupov.

Najlepším spôsobom ako tomuto predchádzať je kontrolovať rozsah a veľkosť dát, s ktorými aplikácia pracuje, či už ide o spomínané vstupy alebo nahrávané súbory a podobne.

Code injection

Typ útoku, pri ktorom útočník vloží do aplikácie kód, ktorý je potom vykonaný. Využíva zlé spracovanie a validáciu vstupných dát. Útočník môže kód vložiť aj do toku dát medzi klientom a prehliadačom podobne ako pri *man-in-the-middle* útoku [29].

V niektorých prípadoch sa môže stať, že cieľ útoku uvidí vložený kód ako súčasť stránky. Samozrejme to tak nemusí byť vždy a kód iba čaká v pozadí aby spôsobil nejakú škodu.

Dobrým spôsobom ako predchádzať tejto hrozbe je používať šifrované dátové toky, HTTPS protokol a iné typy verifikácie dát [35].

Cross-site scripting (XSS)

Pri tomto type útoku sa do inak dôveryhodnej stránky vloží škodlivý skript. Používa sa, keď chce útočník poslať škodlivý kód koncovému užívateľovi pomocou webovej aplikácie. Prehliadač nedokáže zistiť, že ide o skript, ktorý na stránke nemá byť a spustí ho. Ten potom získa prístup ku koláčikom (cookies), rôznym premenným uloženým v relácii (session) alebo iným citlivým dátam prehliadača. Môže taktiež zmeniť HTML štruktúru stránky alebo presmerovať užívateľa na iný web.

Škodlivý obsah býva vo forme JavaScriptu, Flashu, HTML alebo hocijakého iného typu kódu, ktorý dokážu prehliadače spracovať [29].

Cross-site request forgery (CSRF)

CSRF je typ útoku, ktorý sa pokúša prinútiť obeť odoslať nechcené požiadavky. V mnohých prípadoch prehliadač odosiela dotazy, ktoré obsahujú údaju identifikujúce užívateľa, ako napríklad cookie, IP adresu a podobne [35]. Teda pokiaľ je užívateľ autorizovaný, kód na pozadí stránky nemá ako rozoznať nechcený dotaz od útočníka a normálny dotaz od obete. Cieľom je zmena stavu (najčastejšie dát) na serveri, zmena emailu alebo hesla, vytvorenie objednávky. Získať dáta v tomto prípade nemá pre útočníka žiadny význam, pretože on nevidí odpoveď, ktorú vidí jeho obeť.

Nahrávanie súborov

Každý nahratý súbor predstavuje potenciálne riziko. Ten môže po nahratí obsahovať právomoci serveru, takže dokáže spustiť programy, ktoré by sa nám nemuseli páčiť. Najlepším spôsobom ako tomu predísť je kontrolovať nahraté súbory alebo povoliť iba určitý typ. Nemôžete sa ale spoľahnúť iba na čierny zoznam koncoviek súborov (extension). Hacker by mohol nahrať súbor, ktorý vyzerá byť určitého typu ale v skutočnosti sa chová ako inak [35].

Nesprávne alebo žiadne šifrovanie

Pri prenášaní dát medzi dvomi uzlami, je vždy dôležité používať nejaký typ šifrovania, aby sme predišli odpočúvaniu našich dát. Použitie šifrovania vždycky sťaží hackerom prácu pri narúšaní bezpečnosti systému.

Manipulácia parametrov v URL

Útočník sa pokúša meniť parametre posielané v hlavičkách dotazu alebo v URL. Treba dbať na šifrovanie parametrov posielaných medzi prehliadačom a serverom. Najlepšie je použiť protokol HTTPS.

Session hijacking

Tento útok sa snaží zneužiť mechanizmy na správu sedenia, čo sú väčšinou žetóny (*token*). HTTP komunikácia pozostáva z množstva rôznych TCP spojení a webový server potrebuje nejaký spôsob ako ich od seba odlíšiť. Najpoužívanejšia metóda pozostáva z poslania žetónu užívateľovi po tom, ako sa prihlási. Ten môže byť použitý v URL, v hlavičke HTTP

alebo v tele HTTP odpovede [29]. Session hijacking ho potom zneužíva tým, že sa ho snaží ukradnúť alebo odhadnúť a získať tak neautorizovaný prístup k dokumentom na serveri [29]. Najlepšou prevenciou je použitie HTTPS protokolu a žetónov, ktoré nie je jednoduché odhadnúť.

SQL injection

SQL injection pozostáva z vloženia SQL dotazu pomocou vstupných dát od klienta do aplikácie. Úspešný útok dokáže vykonávať CRUD (Create/Read/Update/Delete) operácie nad dátami z databázy, vykonať administratívne operácie nad databázou a v niektorých prípadoch aj poslať príkazy operačnému systému [29].

Denial of Service

DoS útok sa snaží zneprístupniť softvér pre vykonávanie účelu, na ktorý je navrhnutý. Existuje viacero spôsobov ako toho dosiahnuť. Pokiaľ systém obdrží veľký počet požiadaviek, môže sa stať, že spadne a jeho funkčnosť bude obmedzená alebo úplne zastavená. Podobne sa to môže nastať aj v prípade, že útočník využije spôsob fungovania programu alebo zraniteľné miesta v zdrojovom kóde.

Obranou proti hrozbe takéhoto útoku môže byť napríklad použitie kvalitného sieťového hardvéru alebo firewallu.

2.2 Ako minimalizovať zraniteľnosť

Zoznam z predchádzajúcej sekcie by sa dal rozširovať aj na trojnásobok a stále by nebolo obsiahnuté všetko. Jedným zo zdrojov, kde sa dá dočítať oveľa viac nie len o hrozbách ale aj o tom ako testovanie bezpečnosti zakomponovať do procesu vývoja webových aplikácii, je projekt OWASP (Open Web Application Security Process) [29]. V mnohých prípadoch je oprava dier v ochrane softvéru pred hrozbami jednoduchá a jednou opravou dokážeme vyriešiť viac než len jeden problém [35]. Keď sa pozrieme na spomínané hrozby, zistíme, že mnohé z nich sa dá minimalizovať jednoducho použitím HTTPS protokolu a SSL/TLS certifikátov.

Metód ako testovať bezpečnosť systému a odhaliť zraniteľné miesta existuje niekoľko druhov. Môže to byť manuálna inšpekcia, revízia kódu alebo penetračné testy. Niektoré techniky vyžadujú hlbšiu znalosť systému aby mohli vykonať relevantné testy. V tejto sa zameriam na spôsoby prevencie pred hrozbami, ktorých použitie je možné analyzovať bez priameho prístupu ku zdrojovým súborom a bez znalosti komplexnej štruktúry testovaného systému ale iba pomocou analýzy HTTP komunikácie medzi klientom a serverom.

HTTPS a SSL/TLS certifikáty

HTTPS je protokol, ktorý sa používa na šifrovanú komunikáciu cez počítačovú sieť. Dáta sú šifrované pomocou TLS alebo SSL. Certifikáty sa používajú na overenie totožnosti klienta a serveru, teda toho, že pri otvorení stránky prehliadač naozaj komunikuje so serverom, s ktorým by mal komunikovať. Stali za základom dôveryhodnej komunikácie na internete a pokiaľ nie sú prítomné, prehliadač označí stránku ako nedôveryhodnú a bude varovať všetkých návštevníkov. SSL certifikáty majú expiračnú dobu. Najčastejšie som sa stretol s dobou troch mesiacov ale niektoré zdroje uvádzajú maximálnu dobu niečo cez 2 roky. Čím častejšie sa ale obnovujú tým je to bezpečnejšie. Dôležité je dávať pozor na to aby certifikát

nevypršal bez toho, aby sa nahradil za nový a vznikli tak zraniteľné miesta. Najčastejšou hrozbou, pred ktorou chránia sú útoky typu *man-in-the-middle* (napríklad *session hijacking* spomínaný vyššie), odpočúvanie komunikácie alebo vloženie škodlivých dát do nej.

Overiť, či stránka používa nejaký certifikát sa dá napríklad pomocou nástroja `curl`.

```
curl --insecure -vvI https://www.google.com
```

HTTP hlavičky

Množstvu hrozieb sa dá predchádzať použitím niektorých HTTP hlavičiek. Tu je zoznam niektorých dôležitých [29].

- **HTTP Strict Transport Security**

  Ide o protokol, ktorý chráni pred tzv. *protocol downgrade* útokmi alebo *cookie hijacking*. Webový server týmto špecifikuje, že prehliadač by s ním mal komunikovať výhradne prostredníctvom HTTPS a nikdy nie cez HTTP.

  Môže mať dve hodnoty. `Max-age` je čas v sekundách, po ktorý by si mal prehliadač pamätať, že tento web by mal byť sprístupnený iba cez HTTPS. `IncludeSubDomains` značí, že toto pravidlo platí aj pre subdomény.

  ```
  Strict-Transport-Security: max-age=31536000 ; includeSubDomains
  ```

- **X-Frame-Options**

  Táto hlavička zlepšuje ochranu proti clickjackingu. Inštruuje prehliadač o tom, či obsah stránky môže byť zobrazený v `iframe` elemente. Podobnú funkciu má aj hodnota `frame-ancestors` hlavičky `Content-Security-Policy` (CSP). Pokiaľ sú zadefinované obe, CSP má prednosť a `X-Frame-Options` sa ignoruje.

  Môže nadobudnúť hodnoty `deny` (nepovolí vykresľovanie vôbec), `sameorigin` (nepovolí vykresľovanie z iného zdroja) a `allow-from` (povolí vykresľovanie z určitých zdrojov).

  ```
  X-Frame-Options: deny
  ```

- **X-Content-Type-Options**

  Predchádza tomu, aby prehliadač spracovával súbory ako iný MIME typ, než ten čo je špecifikovaný v `Content-Type` hlavičke. Napríklad aby textový súbor spracoval ako css súbor.

  Hodnota `nosniff` zabraňuje MIME-sniff-ingu.

  ```
  X-Content-Type-Options: nosniff
  ```

- **Content-Security-Policy**

  CSP vyžaduje opatrné a precízne nastavenie. Pokiaľ sa použije, tak má veľký dopad na to, ako prehliadač renderuje stránky. Predchádza širokej škále útokov (napríklad Cross Site Scripting a Cross Site Injections). Alternatívne môže byť použitá meta značka:

```
<meta http-equiv="Content-Security-Policy"
content="default-src 'self'">
```

Hlavička zas môže mať takúto formu, špecifikujúcu, že chceme aby prehliadač načítal iba skripty z domény danej stránky:

```
Content-Security-Policy: script-src 'self'
```

Zoznam všetkých možností je na stránkach MDN [2].

- **Referrer-Policy**

  Táto hlavička naznačuje, ktoré informácie o sprostredkovateľovi (referrer) odkazu na stránku by mali byť zahrnuté v hlavičke `Referrer`.

  ```
  # žiadne informácie o~sprostredkovateľovi nebudú odoslané v~hlavičke
  Referrer-Policy: no-referrer
  ```

  Zoznam všetkých hodnôt je dostupný na stránkach MDN [5]. `Referrer-Policy` môže byť špecifikované aj v HTML kóde vo forme tagu `<meta name="referrer" ...>`, ako atribút `referrerpolicy` tagov `a`, `area`, `img`, `iframe`, `script` alebo `link`, alebo ako atribút `rel` tagov `a`, `link` či `area`.

- **Cross-Origin-Embedder-Policy**

  Predchádza načítaniu cross-origin dokumentov, ktoré nemajú explicitne nastavené oprávnenia [29]. Používa sa v kombinácii s hlavičkou `Cross-Origin-Resource-Policy` (CORP).

  Hodnota `unsafe-none` povoľuje dokumentu načítať bez povolenia cez CORS (Cross Origin Resource Sharing) protokol alebo CORP hlavičku. `require-corp` značí, že dokument môže načítať iba iné dokumenty z rovnakého zdroja alebo dokumenty, ktoré majú explicitne povolené načítanie z iného zdroja.

  ```
  Cross-Origin-Embedder-Policy: require-corp
  ```

- **Cross-Origin-Opener-Policy**

  Táto hlavička zaisťuje, že hlavný dokument nezdieľa kontext prehliadača s cross-origin dokumentmi. COOP izoluje dokument a potenciálny útočník nedokáže pristúpiť ku globálnemu objektu pri otvorení v rámci vyskakovacieho okna. Predchádza tak sade cross-origin útokov (XS-Leaks).

  Pokiaľ je stránka otvorená v novom okne, dokument, ktorý ju otvoril nebude mať prístup napríklad k atribútu `window.opener` nového okna [3].

  Hodnota `unsafe-none` umožňuje dokumentu, aby bol pridaný do kontextu toho, kto ho otvoril. Jedine že by ten, kto ho otvoril používal COOP.

  `Same-origin-allow-popups` udržiava odkazy na novootvorené okná alebo záložky, ktoré buď nenastavujú COOP alebo ho nastavujú na `unsafe-none`.

  `Same-origin` izoluje kontext iba v rámci zdroja.

```
Cross-Origin-Opener-Policy: same-origin
```

- **Cross-Origin-Resource-Policy**

  CROP povoľuje definovať pravidlá, ktoré povoľujú webovým stránkam brániť sa proti určitým dotazom z cudzích zdrojov (napríklad pri `script` a `img` tagoch). Predchádza sa tak rôznym Side-Channel, Cross-Site Script Inclusion a podobným útokom [29].

  `Same-site` hodnota povoľuje čítanie dokumentu iba dotazom z rovnakej domény.

  `Same-origin` zas iba tým z rovnakého zdroja (schéma + host + port).

  `cross-origin` povoľuje čítanie hocikomu.

  ```
  Cross-Origin-Resource-Policy: same-origin
  ```

- **Cache-Control**

  Táto hlavička obsahuje informácie o tom akým spôsobom chcem cachovať dotazy a odpovede. Môže nadobúdať rôzne hodnoty a ich kombinácie [1]. Funguje iba s protokolom HTTP verzie 1.1. pre verziu 1.0 sa spätná kompatibilita zabezpečuje hlavičkou `Pragma`. Nasledujúci príklad demonštruje povolenie cachovania dotazu na dobu jedného týždňa.

  ```
  Cache-Control: public, max-age=604800
  Pragma: no-cache
  ```

Nasledujúce hlavičky nijakým spôsobom nepredchádzajú rizikám, ale ich nesprávne použitie k nim môže viesť.

- **Set-Cookie**

  Používa sa na odoslanie cookie zo servera klientovi, aby ich ten mohol potom poslať naspäť. Pri nastavovaní v hlavičke odpovede môže `Set-Cookie` obsahovať viacero príznakov (flags), ktoré pomáhajú predchádzať niektorým hrozbám (XSS, CSRF, man-in-the-middle útoky) [6]:

  - `Secure` naznačuje, že cookie bude odoslaná serveru iba v prípade, kedy sa použije HTTPS protokol.
  - `HttpOnly` predchádza prístupu ku cookie pomocou JavaScriptu. Napríklad cez atribút `Document.cookie`. Ide ale o relatívne novú funkcionalitu a nie všetky prehliadače ju podporujú.
  - `SameSite` kontrolue či bude cookie poslaná v cross-origin dotazoch.

  Príklad použitia:

  ```
  Set-Cookie: id=123; Secure; HttpOnly; SameSite: lax
  ```

- **X-Powered-By a Server**

  Z týchto hlavičky je ľahké vyčítať rôzne informácie o prostredí, v ktorom aplikácia beží. Útočník by ich potom mohol na ľahší útok. Preto je lepšie ak sa tieto hlavičky v odpovedi nenachádzajú.

Príklad hlavičiek vrátených stránkou `https://facebook.com`.

```
$ curl -I https://www.facebook.com
HTTP/2 200
vary: Accept-Encoding
set-cookie: fr=0IF6qSGHmZZhgvKjh..Bh10bL.kK.AAA.0.0.Bh10bL.AWUt7ZTpnZM;
    expires=Wed, 06-Apr-2022 19:45:14 GMT; Max-Age=7775999;
    path=/; domain=.facebook.com; secure; httponly
set-cookie: sb=y0bXYS1simMZdSTBDaWFl8Q_;
    expires=Sat, 06-Jan-2024 19:45:15 GMT; Max-Age=63072000;
    path=/; domain=.facebook.com; secure; httponly
x-fb-rlafr: 0
document-policy: force-load-at-top
pragma: no-cache
cache-control: private, no-cache, no-store, must-revalidate
expires: Sat, 01 Jan 2000 00:00:00 GMT
x-content-type-options: nosniff
x-xss-protection: 0
x-frame-options: DENY
strict-transport-security: max-age=15552000; preload
content-type: text/html; charset="utf-8"
x-fb-debug: wbLiYSu6O3mh713q3Le1qhTtWPlrvHT/pUnb7lxLVykM0oQKdzxGlFr9ZEspLD/
    MVuEAzQtgiu3iPftIWAYVxA==
date: Thu, 06 Jan 2022 19:45:15 GMT
alt-svc: h3=":443"; ma=3600, h3-29=":443"; ma=3600
```

HTTP metóda TRACE

Táto HTTP metóda sa používa na debugovacie účely. Jej funkcia je, že vráti späť klientovi to, čo poslal v dotaze na server. Táto metóda bola pôvodne považovaná za neškodnú ale neskôr sa objavilo, že môže byť použitá na Cross Site Tracing útok [29]. V produkčnom prostredí je odporúčané tuto metódu zakázať.

Pomocou nástroja curl a metódy OPTIONS sa dá zistiť aké metódy server podporuje. Napríklad takto `curl -X TRACE https://google.com -I`. Výstup potom môže obsahovať podporované metódy v hlavičke `Allow` alebo `Access-Control-Allow-Methods`. Pokiaľ ani jedna nie je prítomná, môžeme odoslať priamo metódu TRACE a skontrolovať, ako server odpovie. Ak metóda neni povolená tak zvyčajne vráti kód 405 (method not allowed).

Web Application Firewall

WAF je aplikačný firewall pre HTTP aplikácie. Aplikuje rôzne techniky na HTTP komunikáciu, ktorými sa snaží chrániť pred útokmi ako XSS, SQL Injection, Code Injection, DoS alebo Buffer overflow. Zvyšuje robutnosť aplikácie čo sa týka útokov z vonku [30]. Existuje mnoho dodávateľov firewallu ako Citrix Netscaler, Varnish alebo Wordfence pre Wordpress. Môžu byť identifikovateľné viacerými spôsobmi ako napríklad pomocou cookie, alebo špecifickej HTTP hlavičky. Niekedy môže rýchla odpoveď serveru s kódom 408 (Request Time-out) značiť prítomnosť firewallu (ale táto metóda nie je veľmi spoľahlivá).

Prevencia proti SQL injection

Čo je to SQL injection už bolo vysvetlené. Existuje mnoho techník ako odstrániť zraniteľné miesta, ktoré by k tomuto útoku viedli. Napríklad dôkladná validácia vstupných dát alebo používanie `prepare` príkazu SQL jazyka.

Odhaliť možnosť tohto útoku sa dá pomocou odhalenia vstupných polí a pokúsiť sa vložiť také dáta, ktoré by spôsobili nejakú SQL chybu alebo podobné chovanie. Niekedy na to stačí vložiť znak `'` alebo `;` do testovaného poľa. Pokiaľ to nie je správne filtrované, vytvorí sa nesprávny SQL dotaz a aplikácia vráti túto chybu. Výrazov, ktorými je toto možné testovať je viac. Môže to byť napríklad SQL výraz podobný ako `... AND 1=1`. V tomto prípade by už ale na správnu analýzu bola potrebná lepšia znalosť testovanej aplikácie. Takýto test môže vyzerať takto:

```
http://example.com/items.php?id=2 and 1=2
http://example.com/items.php?id=' and ;
```

2.3 Nástroje na testovanie zabezpečenia

Nástrojov, ktoré sa dajú využiť na analýzu zraniteľnosti webových aplikácií je nespočetné množstvo. Dobrým začiatkom sú určite stránky OWASP [29], kde sa dajú nájsť informácie o zabezpečení aplikácií (aj o tom ako to testovať) na rôznych úrovniach a rôznymi technikami. Takisto sa tam dá nájsť zoznam nástroj, ktoré v tom môžu pomôcť. Niektoré sa venujú analýze zdrojového kódu, iné ponúkajú aj možnosť penetračných testov.

Ďalším nástrojom s GUI, ktorý sa dá použiť na rôzne testy zabezpečenia ako SQL injection, XSS a podobne je w3af (`https://w3af.org/`).

Jedným z takýchto nástrojov je OWASP ZAP (Zend Attack Proxy), čo je open-source desktopový nástroj, ktorý bol vyvinutý aj ako analyzátor zraniteľných miest a aj ako penetračný nástroj. Funguje ako man-in-the-middle proxy, teda zachytáva komunikáciu medzi prehliadačom testera a serverom a preskúmava zaslané správy, modifikuje ich obsah, posiela pakety na inú destináciu a podobne. Pre záujemcov je k dispozícií dokumentácia na adrese `https://www.zaproxy.org/docs/`. Podľa slov ľudí pracujúcich na tomto nástroji ide o svetovo najpoužívanejší skener webových aplikácií.

SQLMap je nástroj pre príkazový riadok, ktorý vykonáva penetračné testy a snaží sa nájsť miesta náchylné na SQL injection. Ide o nástroj napísaný v jazyku Python a podporuje testy pre širokú škálu databázových systémov. Na stránkach `https://sqlmap.org/` je o ňom napísané viac.

Grabber (`https://rgaucher.info/beta/grabber/`) je nástroj, ktorý je navrhnutý na skenovanie malých webov, pretože skenovanie veľkých aplikácií by trvalo moc dlho a dokázalo by to zahltiť sieť. Dokáže odhaliť zraniteľné miesta útokov XSS, SQL injection, analyzovať JavaScriptový kód, kontrolovať zálohovacie súbory a vykonávať black box (skenovanie aplikácie) aj white box (skenovanie zdrojového kódu) analýzu.

Existuje aj viacero online skenerov zraniteľností webových aplikácií. Nástroj `https://securityheaders.com/` skúma HTTP hlavičky a upozorňuje na možné nedostatky. SiteCheck (`https://sitecheck.sucuri.net`) okrem toho kontroluje či sa na stránkach nenachádza malware alebo spam a snaží sa odhaliť prítomnosť firewallu. Za príplatok ponúka aj pravidelné skenovanie webu, monitorovanie SSL certifikátov a ďalšie možnosti.

Wafw00f (`https://github.com/EnableSecurity/wafw00f`) je python nástroj, ktorý sa snaží pomocou rôznych HTTP dotazov zistiť WAF použitý na webe. V detegovaní firewallu

je lepší ako SiteCheck. Je to open-source nástroj, ktorý sa dá použiť z terminálu napríklad takýmto spôsobom:

```
$ wafw00f https://example.org
...
[*] Checking https://example.org
[+] The site https://example.org is behind
Edgecast (Verizon Digital Media) WAF
```

Toto bola len špička ľadovca. Podobných open-source alebo platených nástrojov je omnoho viac. Pri voľbe konkrétneho sa treba zamyslieť nad tým čo chceme testovať (alebo analyzovať) a ako to chcem robiť.

Kapitola 3

Návrh aplikácie

Výsledná aplikácia by mala za využitia poznatkov získaných v kapitolách 1 a 2 analyzovať web a poskytovať spätnú väzbu o prípadných nedostatkoch. Určená bude najmä pre vývojárov pracujúcich na nejakom webovom projekte, prípadne pre správcov internetových stránok (*webmaster*), ktorým by uľahčila pravidelné sledovanie stránok a odhaľovanie chýb. V tejto kapitole je popísaný návrh základnej funkcionality aplikácie, dátového modelu, architektúry a spôsobu analyzovania celého webu a jeho jednotlivých stránok.

3.1 Základná funkcionalita

Užívatelia budú rozdelení do dvoch rolí:

- administrátori a
- bežní užívatelia.

Administrátor bude môcť pridávať, upravovať a mazať projekty identifikované názvom a domovskou URL. Pri vytváraní projektu bude mať možnosť prihlásiť sa cez svoj Google účet, ktorý bude použitý na napojenie aplikácie na *Google Search Console*. Bude musieť zadať názov projektu a domovskú URL.

Bežný užívateľ bude vidieť zoznam projektov, mať možnosť otvoriť si ich detaily a prezerať vykonané analýzy.

Po vytvorení projektu začne aplikácia skenovať mapu stránok, pokiaľ bude nejaká existovať, a hľadať v nej odkazy na stránky webu. V inom prípade začne od domovskej URL a bude hľadať odkazy na stránky v rámci rovnakej domény. Tie sa uložia do databázy a vložia do fronty na skenovanie. Takto by sa malo postupne objaviť čo najviac stránok a predísť prílišnému spamovaniu servera požiadavkami.

Samotné analyzovanie webu sa bude skladať z troch častí. Prvou bude vyššie spomínané napojenie na *Google Search Console*. Odtiaľ si aplikácia vytiahne štatistiky návštevnosti (spomínané v podkapitole 1.6) objavených stránok, ukladať ich do databázy a zobrazovať užívateľovi vo forme grafu (podľa dní), kde bude môcť sledovať zmeny hodnôt v čase.

Druhou časťou bude pravidelné spúšťanie nástroja Lighthouse (podkapitola 1.6), konkrétne meranie rýchlosti načítania jednotlivých stránok. Hodnoty budú taktiež ukladané do databázy a zobrazované v grafe (taktiež podľa dní). Zoznam metrík nástroja aj s podrobnejším vysvetlením sa dá nájsť na stránke `https://web.dev/lighthouse-performance/`. Spustený môže byť napríklad týmto spôsobom:

```
lighthouse https://example.com --only-categories=performance
--output=json --screenEmulation.disabled
--quiet --no-emulated-user-agent
```

.

Detail projektu bude ďalej obsahovať záložku Analýza jednotlivých stránok, kde budú zobrazené objavené stránky webu. Každá stránka sa bude dať otvoriť a v danom detaile človek uvidí spustené audity ich výsledky, ktoré sa budú dať otvoriť pre získanie viac informácií.

Grafické návrhy podstatných celkov webu popísaných v tejto sekcii sa nachádzajú v prílohe B.

3.2 Priebeh a metódy analýzy

Pri ukladaní objavenej stránky do databázy sa bude taktiež ukladať titulok, *meta description*, kanonická URL, *OpenGraph titulok* a *OpenGraph popis*. Tieto hodnoty budú použité na kontrolu samotnej existencie, optimálnej dĺžky a unikátnosti v rámci webu.

Ďalej sa prostredníctvom hlavičiek a obsahu stránky získaných pomocou nástroja *curl* bude kontrolovať presmerovanie HTTP na HTTPS, návratový kód, podpora HTTP verzie 2, podpora kompresie a presmerovanie rôznych foriem URL (rozdielne v tom či začínajú s www alebo nie alebo, či končia s lomkou alebo nie) na jednu unikátnu pomocou trvalého presmerovania (HTTP kód 301). Z formulárov s metódou GET (metóda POST by mohla spôsobiť modifikáciu dát v testovanej aplikácií) a každej URL obsahujúcej nejaké parametre bude zostavených niekoľko URL tak, aby otestovali všetky *inputy* alebo parametre. Tie sa potom použijú na testovanie metódou „blind SQL injection“.

Analýza pomocou *Chrome DevTools protokolu*[1] bude prebiehať komunikáciou cez socket prepojenie na prehliadač Chrome alebo Chromium v tzv. „headless“ prostredí[2]. Toto bude použité na kontrolu, či konzola v prehliadači neobsahuje chybové hlášky, či pri načítaní stránky existuje nejaký dotaz vracajúci chybový HTTP kód. Taktiež sa bude kontrolovať prítomnosť TLS/SSL certifikátu, správnosť domény, pre ktorú je certifikát platný a samotná platnosť certifikátu.

3.3 Dátový model

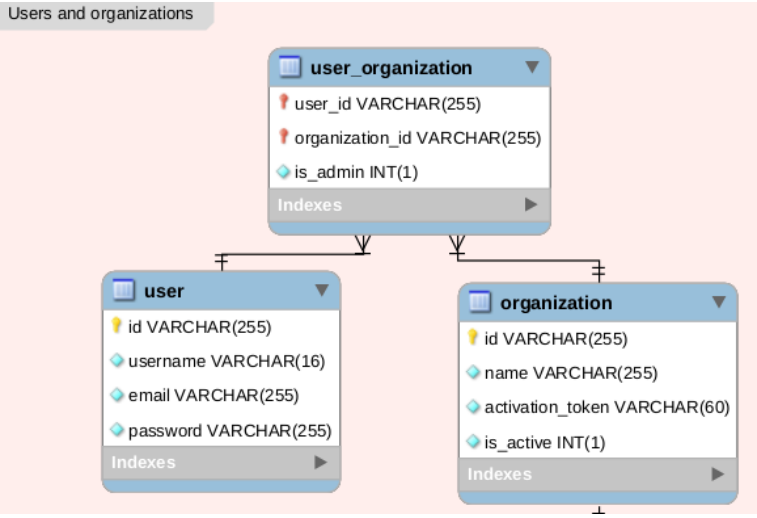
V tejto sekcii sú popísané jednotlivé dáta, s ktorými bude pracovať výsledná aplikácia, a vzťahy medzi nimi.

Užívatelia a role

Užívatelia budú identifikovaný pomocou kľúča, užívateľského mena a emailovej adresy. Každý môže mať niekoľko rolí. Diagram je zobrazený na obrázku 3.1.

[1] Umožňuje *debug* a profilovanie v prehliadačoch Chromium a Chrome. Dokumentácia je dostupná na adrese `https://chromedevtools.github.io/devtools-protocol/`.

[2] Ide o spôsob spustenia prehliadača Chrome bez grafického rozhrania. Ako to dosiahnuť je popísané na adrese `https://developers.google.com/web/updates/2017/04/headless-chrome`



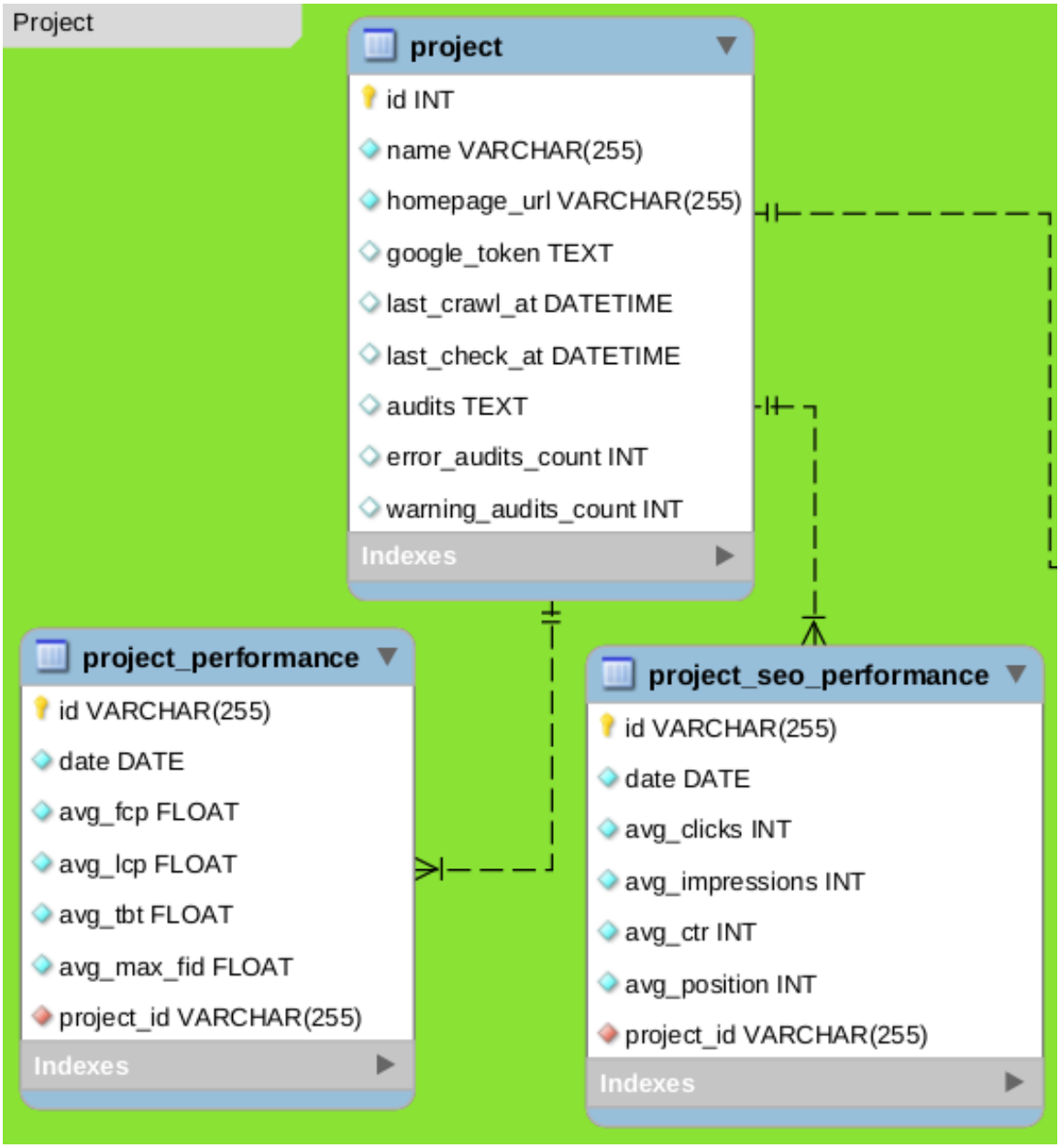
Obr. 3.1: Diagram závislosti entít (ERD) užívateľa a organizácie.

Projekt

Projekt predstavuje analyzovaný web. Každý bude mať názov, adresu domovskej stránky a token (`google_token`), ktorý sa v prípade použitia *Google Search Console* použije na prihlásenie do tejto služby. Každý projekt bude mať audity uložené uložených vo formáte JSON zakódovanom do formátu *Base64*. Ďalej bude táto entita obsahovať informácie o poslednom prehľadávaní odkazov na hocijakej stránke (`last_crawl_at`), poslednom auditovaní nejakej stránky (`last_check_at`) a celkový počet chýb a varovaní objavených v rámci celého projektu. Tabuľka *project_performance* bude obsahovať priemerné hodnoty metrík výkonu v priebehu určitého dňa (za daný deň musí prebehnúť aspoň jedna kontrola výkonu hocijakej stránky v rámci daného projektu). Do tabuľky *project_seo_performance* sa budú ukladať dáta získané zo služby *Search Console*. Bude to počet klikov, počet zobrazení, CTR (počet klikov / počet zobrazení) a priemerná pozícia vy výsledkoch vyhľadávania za určitý deň. Tieto vzťahy sú zobrazené na obrázku 3.2.

Stránka

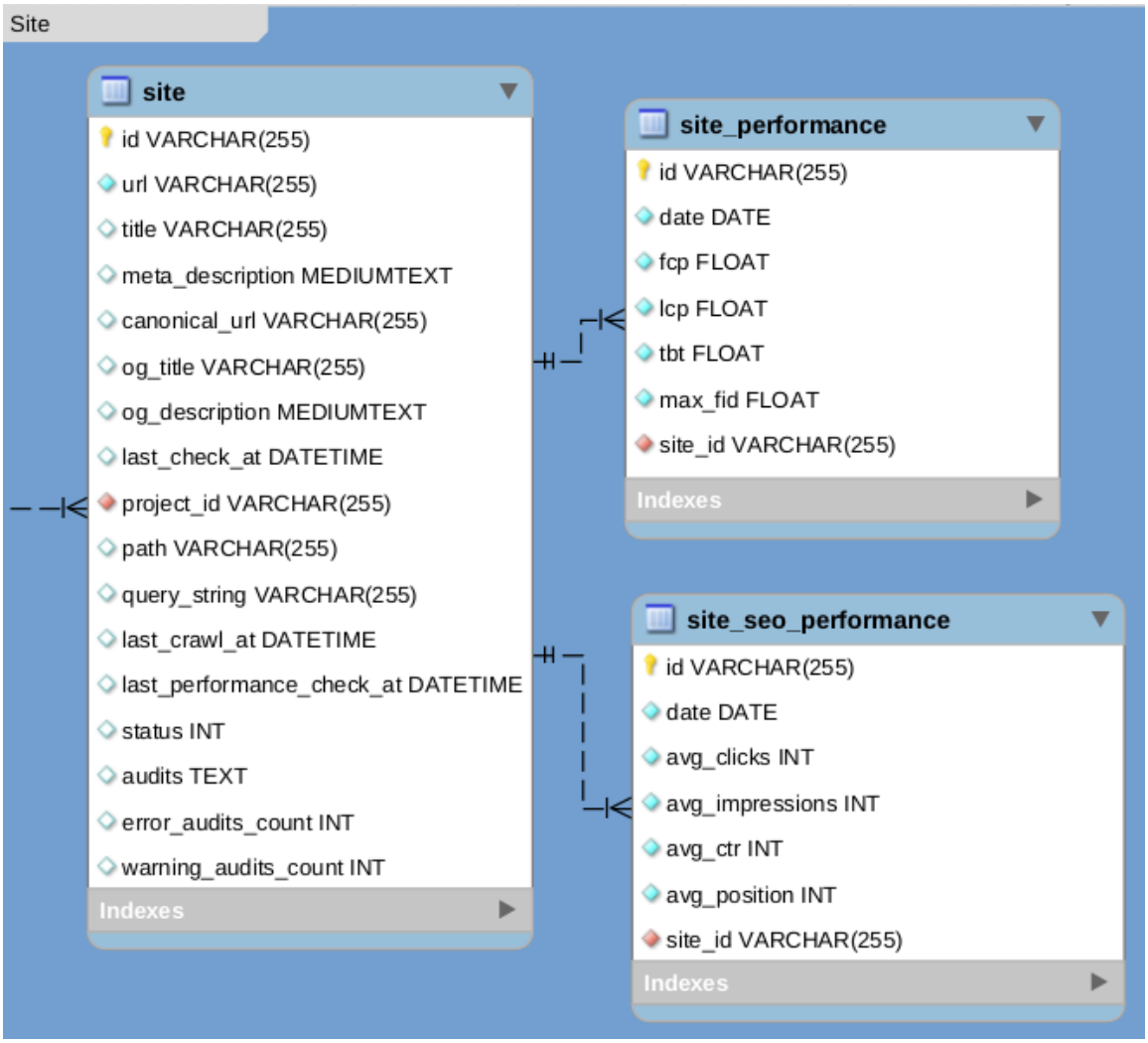
V rámci jedného projektu bude uložených niekoľko stránok (vzťah 1:N). Každá stránka bude mať uložené informácie z HTML kódu ako titulok, meta popis, *OpenGraph* titulok, *OpenGraph* popis a kanonickú URL. Tie budú využité pri analýze, napríklad na kontrolu unikátnosti. Taktiež sa bude ukladať URL (`url`, `path` a `query_string`), dátum poslednej analýzy (`last_check_at`), dátum posledného skenovania stránky (`last_crawl_at`) alebo dátum poslednej analýzy výkonu (`last_performance_check_at`). Po každom audite sa budú aktualizovať hodnoty samotných auditov (`audits`), HTTP kód stránky („status“), počet chybových auditov (`error_audits_count`) a počet varovaní (`warning_audits_count`). Táto entita bude vo vzťahoch 1:N s podobnými tabuľkami ako samotný projekt. Tú sa budú ale ukladať dáta pre samotnú tabuľku. *Site_performance* a *site_seo_performance* sú v tomto poradí ekvivalentmi *project_performance*, a *project_seo_performance*. Diagram je na obrázku 3.3.



Obr. 3.2: ERD entity projekt a pridružených entít.

3.4 Architektúra

V tejto sekcii sú popísané jednotlivé časti, z ktorých bude aplikácia zložená, ich závislosti a účel. Diagram je zobrazený na obrázku 3.4. Aplikácia bude bežať v Docker prostredí a jednotlivé časti budú implementované v separátnych kontajneroch. Taktiež sú k jednotlivým častiam navrhnuté technológie, ktoré budú použité na ich implementáciu.



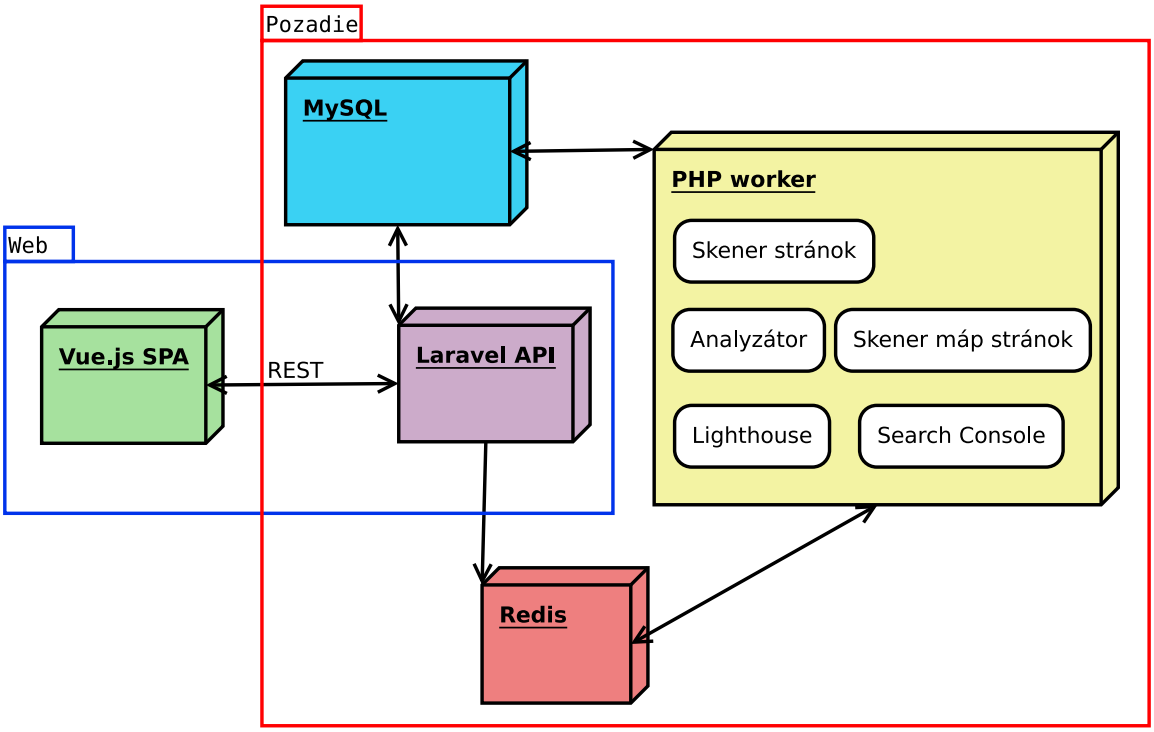
Obr. 3.3: ERD entity stránka a pridružených entít.

Frontend

Užívateľské rozhranie bude implementované ako SPA[3], pomocou JavaScriptového rámca `Vue.js` [13], ktorý uľahčuje vývoj webových užívateľských rozhraní. `Vuex` je knižnica a vzor stavového managementu a funguje ako centralizovaný sklad pre všetky komponenty aplikácie [25]. `Vue Router` je oficiálny smerovač (*router*) pre `Vue.js` aplikácie a zjednodušuje vytváranie SPA. Zabezpečuje vykresľovanie konkrétnej komponenty na základe URL a umožňuje definíciu akcii, vykonaných pred presmerovaním (napríklad kontrola prihláseného užívateľa a jeho role). Na uľahčenie vývoja webového rozhrania bude použitá knižnica *Vuetify* [27].

So serverom bude klient komunikovať pomocou REST API.

[3]Single-page aplikácia je webová aplikácia, ktorá dynamicky prepisuje obsah jednej stránky, namiesto načítavania celých nových stránok prehliadačom [37]



Obr. 3.4: Architektúra aplikácie.

Backend

Backend bude implementovaný PHP frameworkom `Laravel` [26] s architektonickým vzorom `Porto` [34], ktorý pomáha lepšie organizovať a vytvárať udržateľný a znovu-použiteľný kód. Pozostáva z 2 vrstiev. Vrstva kontajnerov, ktorá obsahuje *business* logiku a vrstva lode, ktorá obsahuje spoločnú logiku pre celú aplikáciu. Jeden kontajner môže byť jedna funkcionalita, logika koncových bodov API pre nejakú entitu (napríklad projekt) a podobne. Kontajnery môžu byť logicky zoskupené do sekcii.

Základnou funkcionalitou bude autorizácia užívateľov pomocou *JWT* (JSON web token), zabezpečenie prihlásenia pomocou *OAuth 2.0* na získanie prístupu do *Google API* a zabezpečenie potrebných *CRUD* operácii nad užívateľmi, projektmi a stránkami pomocou *REST API*.

PHP worker

`PHP worker` bude služba zabezpečujúca beh rôznych skriptov a úloh na pozadí pomocou systému Supervisor. Pavúk (skener) bude nepretržite prehľadávať web a do databázy ukladať objavené stránky. Skener máp stránok bude v pravidelných intervaloch skenovať obsah máp na jednotlivých weboch (projektoch). Analyzátor zas bude nepretržite vykonávať analýzu jednotlivých stránok v databáze. Ďalšími časťami budú skript na spustenie a ukladanie výsledkov nástroja *Lighthouse* a pravidelné získavanie dát z *Google Search Console*.

Databáza

`MySQL` je voľne šíriteľný systém pre správu relačných databáz [28]. Práve na ňom bude postavená hlavná databáza prístupná cez webové rozhranie pomocou programu *PhpMyAdmin*.

Redis

`Redis` je sklad dátových štruktúr, ktorý sa dá použiť ako databáza, dočasná pamäť alebo sprostredkovateľ správ [19]. Bude použitý na zabezpečenie behu fronty na skenovanie stránok. Pre každý projekt bude bežať samostatná fronta. Ďalšia fronta bude použitá na ostatné procesy aplikácie, ktoré nemusia bežať synchrónne, ako napríklad posielanie emailov alebo vynútenie skenovania mapy stránok ihneď po vytvorení projektu.

Kapitola 4

Implementácia

V tejto kapitole je popísaná implementácia Docker prostredia, v ktorom beží aplikácia, jednotlivých kontajnerov a ich účel, ale aj implementácia jednotlivých logických celkov a častí.

Zdrojový kód je verzovaný pomocou verzovacieho nástroja Git a webovej aplikácie a služby GitHub. Aplikácia je rozdelená do dvoch repozitárov. V jednom sú zdrojové súbory webového klienta. V druhom sa nachádza okrem zdrojového kódu servera (REST API a skripty bežiace na pozadí) aj definícia Docker kontajnerov, služieb, ktoré v nich bežia a ich konfigurácia pomocou nástroja *Docker Compose*[1]. Štruktúra je naznačená aj v prílohe A.

4.1 Docker prostredie

Docker bol zvolený kvôli tomu, aby sa vyhlo problémom s nekompatibilitou pri vývoji aplikácie na rôznych systémoch alebo pri budúcom nasadzovaní aplikácie na produkčné prostredie. V tejto sekcii sú popísané jednotlivé služby, ktoré bežia v samostatných kontajneroch. Všetky sú definované v súbore `docker-compose.yml`.

V zložke `scripts` sa nachádzajú dva skripty:

- `setup-services.sh` - slúži na základné nastavenie služieb *Nginx* a *PHP worker*,
- `build.sh` - slúži na nainštalovanie potrebných závislostí a aktualizáciu po vykonaní zmien v implementácií klienta aj servera.

Zložka `.data` obsahuje perzistentné dáta, ktoré slúžia na uchovanie stavu aplikácie pri vypnutí prostredia (napríklad príkazom `docker-compose down`).

V zložke `conf.d` sa zas nachádzajú konfiguračné súbory stránok v rámci služby *Nginx* a procesov spravovaných službou *Supervisor* (systém na kontrolu procesov) v kontajneri *php-worker*.

Pracovné prostredie servera

Pracovné prostredie (*workspace*) je kontajner, ktorý slúži ako vývojové prostredie servera. Okrem samotného vývoja sa používa na testovanie, napríklad skriptov, ktoré inak spúšťa

[1] Nástroj na správu viac-kontajnerových Docker aplikácií. *YAML* konfiguračný súbor umožňuje vytvárať a spúšťať jednotlivé služby pomocou jediného príkazu. Dokumentácia je dostupná na adrese `https://docs.docker.com/compose/`.

PHP worker na pozadí. Kontext kontajneru sa nachádza v zložke *workspace* a definuje napríklad inštaláciu PHP vo verzii *8.1* alebo prehliadača Chrome a nástroja Lighthouse.

FPM

V zložke *php-fpm* sa nachádza kontext kontajneru so službou *PHP-FPM* (PHP FastCGI Process Manager). Tá poskytuje rôznu funkcionalitu užitočnú hlavne pre silno zaťažované stránky. Obyčajne sa používa spolu s webovými servermi, na ktorých bežia webové aplikácie, ktoré vracajú HTML, JavaScript a iný obsah bez PHP.

PHP worker

Pokiaľ by sme chceli všetky úlohy na pozadí spúšťať v pracovnom prostredí, museli by sme otvoriť niekoľko relácii a každá by vykonávala niečo iné. Pre uľahčenie je zadefinovaný kontajner *php-worker*. Kontext v rovnomennej zložke definuje podobné služby ako pracovné prostredie. Navyše je tu ale nainštalovaná služba Supervisor. Príklady konfiguračných súborov jednotlivých procesov aplikácie sú v pod-zložke *supervisor.d*. Tieto procesy sú viac popísané v sekcii 3.4.

Node - pracovné prostredie klienta

Podobne ako je definované pracovné prostredie pre server, tak je definované aj pre klient. Kontext tohto kontajneru definuje inštaláciu *Node.js* verzie 16 a rôznych *npm* (Node Package Manager) balíkov potrebných na spustenie klienta. V tomto kontajneri sa dá spustiť auto-obnovovacia verzia pre uľahčenie a zrýchlenie vývoja pomocou sekvencie príkazov:

```
$ docker-compose exec node bash
$ yarn serve-docker
```

Mariadb

Tento kontajner obsahuje relačný databázový server MariaDB, ktorý je nástupcom MySQL. Konfigurácia a kontext sa nachádzajú v zložke *mariadb*. Pomocou premenných prostredia v súbore *.env* sa dá nastaviť názov hlavnej databázy, užívateľ, heslo a heslo pre *root* užívateľa.

PhpMyAdmin

Pre jednoduchšiu správu databázy je definovaný kontajner s webovou aplikáciou phpMyAdmin. Ide o nástroj napísaný v jazyku PHP, ktorý umožňuje okrem iného *mazať/vytvárať* databázy, *mazať/vytvárať/editovať* tabuľky alebo spúšťať SQL príkazy.

Nginx

Hlavný webový server beží v kontajneri *nginx*. Nginx server pozostáva z modulov kontrolovaných direktívami definovanými v konfiguračných súborov. Príklady týchto súborov, potrebných pre aplikáciu sú v zložke *nginx/sites*:

- *api.conf.example* - definícia domény, cez ktorú bude dostupné REST API,
- *app.conf.example* - definícia domény, cez ktorú bude dostupný webový klient,

- *pma.conf.example* - definícia domény, cez ktorú bude dostupný phpMyAdmin.

Redis

Posledný kontajner slúži ako prostredie pre službu Redis. Tá je definovaná iba v konfiguračnom súbore *docker-compose.yml*. Použitý je oficiálny Docker obraz dostupný na `https://hub.docker.com/_/redis`.

4.2 Server

Zdrojový kód (zložk `backend` na disku) má upravenú klasickú architektúru Laravel aplikácie upraveného vzoru architektúry *Porto*.

Z klasickej architektúry sa v zložke *routes* nachádzajú definície koncových bodov API (endpoint) a v `migrations` zas migrácie, ktoré vytvoria tabuľky databázy, a skript na naplnenie databázy testovacími dátami.

Jadro implementácie sa nachádza v zložke `App`. `Core` obsahuje kód spoločný pre všetky moduly aplikácie. Sú tu definície rodičovských tried, rozhraní, rôznych pomocných tried (napríklad na prácu s reťazcami alebo spracovanie HTML) alebo implementácia API služby, ktorá umožňuje uniformné spracovanie požiadaviek a vytváranie odpovedí v rámci definície jednotlivých koncových bodov API v ovládačoch (*Controller*).

Konkrétne moduly servera, popísané v tejto sekcii, sú implementované v zložke `Containers`. Každá pod-zložka predstavuje jeden kontajner alebo sekciu (skupinu kontajnerov) podľa použitého vzoru Porto.

Užívatelia, autentizácia a autorizácia

Kontajner s logikou pre prácu s užívateľmi sa nachádza v zložke `Users`, ktorá má nasledovnú štruktúru (podobnú štruktúru s malými obmenami majú aj ostatné kontajnery. Môžu sa v nich nachádzať rozličné, inak pomenované zložky, podľa toho aká logika alebo aký modul je implementovaný v danom kontajnery):

- **Actions** - Triedy (akcie) majú definovanú metódu `run`, v ktorej je implementovaná určitá logika týkajúca sa užívateľov a ich rolí. Napríklad vytvorenie, aktualizácia a mazanie užívateľa, priradenie role užívateľovi alebo jej odstránenie.
- **Contracts** - Definície rozličných rozhraní.
- **Controllers** - Každá metóda v ovládači prislúcha určitému koncovému bodu API. Metóda podľa potreby autorizuje užívateľa, zavolá filter požiadavku a predá vrátený objekt so vstupnými dátami určitej akcii. Pomocou služby na filtrovanie definovanej v jadre aplikácie a transformátoru potom vráti dáta a správny HTTP kód podľa daného koncového bodu.
- **Events** - Udalosti vyvolané z rôznych miest aplikácie. Napríklad po prihlásení užívateľa.
- **Listeners** - Triedy, ktoré spracovávajú udalosti.
- **Models** - Tu je definovaný model `User`, ktorý reprezentuje jeden záznam užívateľa v databáze.

- **Policies** - `UsersPolicy` definuje metódy na autorizovanie toho, že prihlásený alebo neprihlásený užívateľ môže vykonávať určitú akciu nad daným modelom (v tomto prípade modelom užívateľa). Napríklad či môže aktualizovať, upraviť alebo zmazať daný model (záznam databázy).
- **Queries** - Triedy, ktoré poskytujú určitú úroveň abstrakcie pri vytváraní dotazov do databázy. Umožňujú volanie metód ako `whereName('foo')` alebo `whereIsActive(true)` namiesto písania čistej SQL požiadavky.
- **Repositories** - Tieto triedy zas poskytujú určitú úroveň abstrakcie nad prácou s dátami databázy.
- **Requests** - V tejto zložke sú definované triedy filtrujúce a validujúce vstupné dáta koncových bodov API. `UserRequestFilter` validuje požiadavky na vytvorenie a aktualizáciu užívateľa, `UserRoleRequestFilter` na pridanie a odstránenie role a `UserPasswordRequestFilter` na aktualizáciu hesla užívateľa.
- **Tasks** - Triedy na rozdiel od akcii implementujú znovu-použiteľnú logiku, ktorá môže byť použitá kdekoľvek v aplikácií.
- **Transformers** - Transformátory obsahujú metódu `transform`, ktorá ako vstupný parameter príjme model a vráti pole položiek, ktoré sú vrátené nejakým koncovým bodom API. Okrem toho definujú položky, nad ktorými je možné vykonávať operácie filtrovania, radenia a vyhľadávania v záznamoch daného modelu v databáze pomocou *query* parametrov v URL požiadavky. Taktiež sa tu nachádzajú definície *embedov*, dát, ktoré sa v tele odpovede posielajú na vyžiadanie (môžu to byť napríklad rôzne relácie). Tieto potom klient definuje v *query* parametri `embeds` a v rovnakom parametri sa nachádzajú aj v tele odpovedi.
- **Values**
  - **InputData** - Vstupné dáta sú triedy, ktoré slúžia na prenos dát medzi rôznymi časťami aplikácie. Napríklad na predanie dát, validovaných triedou v zložke `Requests`, metóde `run` nejakej akcie.

V kontajneri pre užívateľov je implementácia týchto koncových bodov API, ktoré boli využité v SPA klienta:

- `GET /api/users/[userId]` - Vráti konkrétneho užívateľa s daným identifikátorom.
- `PUT|PATCH /api/users/[userId]` - Úprava užívateľa s daným identifikátorom podľa položiek predaných v tele požiadavky.
- `GET /api/users/[userId]/roles` - Vráti zoznam rolí konkrétneho užívateľa s daným identifikátorom.
- `PUT api/users/[userId]/password` - V tele požiadavku očakáva nové heslo užívateľa, jeho potvrdenie a staré heslo kvôli zabezpečeniu. Skontroluje či staré heslo zodpovedá záznamu v databáze a prípadne aktualizuje heslo prihláseného užívateľa.

Prihlasovanie užívateľov je implementované pomocou sedení (*sessions*) a žetónov (*tokens*). Informácie o každom sedení sa ukladajú do tabuľky `user_session`. Jeden užívateľ ich

môže mať aktívnych niekoľko naraz. Ku každému sedeniu sa po autorizovaní vytvorí dvojica unikátnych JWT[2]. Jeden sa používa na priami prístup do aplikácie pomocou REST API (*access token*) a druhý na obnovenie toho prvého (*refresh token*). Užitočné informácie (*payload*) o každom žetóne, ako napríklad dátum skončenia platnosti, sú v tabuľke `user_token`. Žetóny sú hašované algoritmom *sha256*. Pre autentizáciu každej požiadavky na REST API sa musí odoslať jeden z dvojice žetónov v hlavičke `Authorization` ako tzv. *bearer* žetón.

V kontajneri `Authentication` sú implementované koncové body API pre vytvorenie, obnovenie a zmazanie sedenia spolu so žetónmi:

- `POST /api/auth/user-sessions` - V tele požiadavky sa očakáva e-mail a heslo. Potom sa skontroluje či existuje užívateľ s daným e-mailom. Pokiaľ áno tak sa ešte porovná zadané heslo s heslom zahašovaným pomocou *bcrypt* funkcie uloženým v databáze. Ak je aj heslo správne, potom sa vytvorí nové sedenie spolu so žetónmi. V tele odpovedi sa odošlú informácie o sedení spolu s prístupovým a obnovovacím žetónom a dátumami skončenia ich platnosti. V prípade, že sa e-mail a heslo nezhodujú so žiadnym záznamom v databáze, vráti sa HTTP kód 403. Pri odoslaní mnoho pokusov (10 za minútu) o prihlásenie sa vráti kód 423.
- `GET /api/auth/user-sessions` - Vráti informácie o všetkých sedeniach prihláseného užívateľa.
- `DELETE /api/auth/user-sessions/[user_session_id]` - Zmaže sedenie s daným identifikátorom a deaktivuje žetóny k nemu priradené.
- `POST /api/auth/user-tokens` - V tele požiadavky očakáva obnovovací žetón. Pokiaľ je platný, vytvorí nový pár žetónov k danému sedeniu a invaliduje staré žetóny. Každý obnovovací žetón môže byť takto použitý iba raz. Pokiaľ je žetón v tele požiadavky neplatný, odpovie sa HTTP kódom 401.
- `GET /api/auth/me` - Vráti informácie o prihlásenom užívateľovi. V tele úspešnej odpovede sa nachádza položka `user`, ktorá obsahuje informácie o užívateľovi a položka `permissions` s informáciami o právach užívateľa na základe rolí, ktoré ma priradené (toto je ešte popísané ďalej v tejto pod-sekcii).

Kontajner `Authorization` obsahuje zdrojový kód pre prácu s rolami a k nim priradenými oprávneniami v rámci modulov. Každá rola v tabuľke `role` má priradené oprávnenia v tabuľke `role_permission`, kde každý záznam obsahuje identifikátor role, typu oprávnenia a identifikátor modulu z tabuľky `module`. Všetky tieto tabuľky sú potrebnými dátami naplnené pomocou typického Laravel príkazu `php artisan db:seed`. Aplikácia pracuje s modulmi `users` pre prístup k zdrojom užívateľov, `projects` pre projekty a `sites` pre stránky. Existuje šesť typov oprávnení:

- **access** - Znamená, že užívateľ (resp. rola) má prístup k danému modulu. V aplikácií sa používa napríklad na autorizovanie výpisu všetkých projektov.
- **read** - Znamená, že užívateľ (resp. rola) má možnosť čítať dáta daného modulu. V aplikácií sa používa napríklad na autorizovanie výpisu detailu konkrétneho projektu.

[2]JSON Web Token (JWT) je otvoreným štandardom, ktorý definuje kompaktný a bezpečný spôsob prenosu informácií medzi dvomi stranami ako objekt JSON (`https://jwt.io/introduction`).

- **create** - Znamená, že užívateľ (resp. rola) má možnosť vytvárať záznamy daného modulu. Napríklad nové projekty.
- **update** - Znamená, že užívateľ (resp. rola) má možnosť upravovať záznamy daného modulu.
- **delete** - Znamená, že užívateľ (resp. rola) má možnosť mazať záznamy daného modulu.
- **full** - Znamená, že užívateľ (resp. rola) má plný prístup k danému modulu.

Po naplnení databázy základnými dátami existuje rola administrátor a rola užívateľ. Administrátor má plný prístup ku všetkým modulom a užívateľ k ním má iba oprávnenia `access` a `read`.

V rámci tohto kontajneru je dôležitá ešte implementácia služby `PermissionsService`, ktorá slúži na overenie, že prihlásený užívateľ má konkrétne oprávnenie v rámci daného modulu. Táto služba sa využíva hlavne v tzv. *policies* spomínaných na začiatku tejto podsekcie.

Potom sa tu ešte nachádza akcia `AuthorizeAction`, ktorá ma variabilný počet vstupných argumentov. Prvý je ale povinný a je to vždy typ oprávnenia. Ďalšie argumenty slúžia na zistenie, ktorá *policy* sa má použiť. Na základe predaných argumentov a danej *policy* skontroluje či má prihlásený alebo neprihlásený užívateľ dostatočné oprávnenie. Pokiaľ nemá tak vyhodí autorizačnú alebo autentifikačnú výnimku. Táto akcia sa volá v každej metóde každého ovládača, kde je potrebné autorizovať prístup k danému zdroju.

REST API projektov

V sekcii `Projects` sa nachádzajú kontajnery s implementáciou logiky pre samotné projekty, výkony projektov a výkony projektov vo vyhľadávaní.

Koncové body API pre projekty, dostupné iba prihláseným užívateľom, majú nasledovnú funkcionalitu:

- `GET /api/projects` - Vráti prípadne vyfiltrovaný, zoradený alebo stránkovaný zoznam projektov.
- `POST /api/projects` - Vloží nový projekt do databázy a v tele odpovede vráti jeho dáta. V tele požiadavky sa očakáva názov, adresa domovskej adresy a prípadne *Google token*. Pred uložením sa validuje unikátnosť názvu, pomocou HTTP klienta implementovaného v jadre Laravelu aj existencia URL domovskej stránky. *Google token* môže byť nulová hodnota (`Null`) alebo to môže byť pole obsahujúce položky *access_token*, *refresh_token*, *expires_in*, *created*, *id_token*, *token_type*, *scope*, *email*. Proces získania týchto dát je popísaný v samostatnej sekcii 4.4. V prípade chybných dát sa odpovie HTTP kódom 422 s chybovými správami v tele.
- `GET /api/projects/[projectId]` - Vráti detail projektu s daným identifikátorom alebo HTTP kód 404 v prípade, že takýto projekt neexistuje.
- `PUT|PATCH /api/projects/[projectId]` - Aktualizuje dáta projektu. V tele požiadavky očakáva názov a *Google token*. Adresa domovskej stránky sa aktualizovať nedá, pretože v prípade jej zmeny by išlo vlastne o vytvorenie nového projektu. V prípade chybných dát sa odpovie HTTP kódom 422 s chybovými správami v tele. Pokiaľ projekt s identifikátorom neexistuje, vráti HTTP kód 404.

- `DELETE /api/projects/[projectId]` - Zmaže projekt s daným identifikátorom z databázy. V prípade, že takýto projekt neexistuje, odpovie kódom 404.

Pri vytváraní projektu sa automaticky nastaví aj parameter `queue_id`. Ten sa vypočíta ako modulo celkového počtu projektov a počtu front, definovaných v súbore *.env*. To slúži na priradenie projektu do určitej fronty na analýzu alebo na skenovanie a aby sa tak znížila záťaž na procesor, tým, že sa obmedzení počet naraz vykonávaných analýz. Po úprave premennej v súbore *.env* je potrebné spustiť skript `php artisan projects:queue-ids`, ktorý prepočíta a uloží identifikátor fronty pre už existujúce projekty.

Pre entity výkonu projektu a výkonu vo vyhľadávaní existujú tieto koncové body:

- `GET /api/project-performances` a
- `GET /api/project-seo-performances`.

Oba jednoducho vracajú zoznam dát, ktoré je možne filtrovať podľa identifikátoru projektu alebo dátumu vytvorenia záznamu. Tým je možné získať dáta o výkone projektu v priebehu určitého časového rozhrania.

REST API stránok

V sekcii `Sites` sa podobne ako v prípade projektov nachádzajú kontajnery s implementáciou logiky pre samotné stránky, výkony stránok a výkony stránok vo vyhľadávaní.

Stránky není ale možné vytvárať, editovať ani mazať. Existujú tak iba tieto dva koncové body API:

- `GET /api/sites` - Vráti prípadne vyfiltrovaný, zoradený alebo stránkovaný zoznam stránok.
- `GET /api/sites/[siteId]` - Vráti detail stránky s daným identifikátorom alebo HTTP kód 404 v prípade, že takáto stránka neexistuje.

K obom koncovým bodom je možné definovať filter `performance_date`, ktorým je možné vymedziť časové rozhranie pre načítané dát o výkone stránky. Pomocou *embedu* `performance` je potom možné získať polia dát pre jednotlivé metriky, kde kľúčom každej hodnoty je dátum vo formáte `Y-m-d`. Taktiež sa pri použití tohto *embedu* pošle aj posledná hodnota každej metriky. Tieto metriky už boli opísané v predchádzajúcich kapitolách.

Podobne funguje filter `seo_performance_date`, ktorým je možné vymedziť časové rozhranie pre načítané dát o výkone stránky vo vyhľadávaní. Takisto existuje *embed* `seo_performance`. Pri použití tohto *embedu* sa ale priemerná hodnota každej metriky za dané časové obdobie. Tieto metriky už boli takisto opísané v predchádzajúcich kapitolách.

Tak ako tomu bolo pri projektoch, aj tu je existujú koncové body pre entity výkonu stránky a výkonu vo vyhľadávaní.

- `GET /api/site-performances` a
- `GET /api/site-seo-performances`.

Oba jednoducho vracajú zoznam dát, ktoré je možne filtrovať podľa identifikátoru stránky alebo dátumu vytvorenia záznamu. Tým je možné získať dáta o výkone stránky v priebehu určitého časového rozhrania.

Procesy na pozadí

V tejto časti je popísaná implementácia procesov, ktoré bežia na pozadí servera. Ako už bolo popísané, beh týchto procesov ma sa starosti služba Supervisor, ktorá beží v kontajneri *php-worker*. Konfiguračné súbory sa nachádzajú v zložke *docker/conf.d/php-worker/supervisord.d*. Táto zložka je mapovaná na zložku */etc/supervisord.d* vo vnútri kontajneru. Každý konfiguračný súbor obsahuje definíciu jedného programu, ktorý obsahuje nastavenia jedného alebo viacerých procesov spustených pomocou rovnakého príkazu.

```
$ php artisan queue:work --queue=default --max-time=3600 --tries=3
```

Tento proces má na starosti beh základnej fronty aplikácie. Spúšťa sa pomocou Laravel príkazu. Argument `--queue=default` označuje frontu, ktorá sa má spracovávať, `--max-time=3600` je čas v sekundách, po ktorom sa spracovávanie fronty ukončí (Supervisor potom zabezpečí reštart procesu) a `--tries=3` je maximálny počet pokusov na vykonanie jednej úlohy (procesu) vo fronte v prípade neúspechu. Do tejto fronty sa po vytvorení projektu vloží proces (*job*), ktorý po spustení začne prechádzať súbory *sitemap.xml* a *sitemap_index.xml* na danej doméne projektu. V daných súboroch sa hľadajú odkazy na webové stránky na danej doméne.

Pre každú takúto stránku, ktorá ešte nie je uložená v databáze sa do fronty, ktorej názov je vo formáte `crawler-[project_id]`, kde *project_id* je identifikátor projektu, do ktorého patrí daná stránka, vloží *job* na skenovanie danej stránky. Spracovanie týchto front je zabezpečené pomocou plánovača úloh (*scheduler*), ktoré je implementovaný v jadre Laravelu. Jeho beh je zabezpečený ďalším procesom služby Supervisor.

```
$ php artisan schedule:work
```

Tento proces je vhodný na správu iných procesov, ktoré by sa inak spúšťali pomocou služby *cron*. Konfigurácia jednotlivých príkazov, ktoré tento proces spúšťa sa nachádza v jadre implementácie v súbore *Kernels/ConsoleKernel.php*.

Každú minútu spúšťa na pozadí frontu pre každý projekt pomocou podobného príkazu ako v prípade základnej fronty. Rozdiel je v argumente `--queue`, ktorý má v tomto prípade hodnotu konkrétnej fronty pre projekt vo vyššie spomínanom formáte.

Každých tridsať minút spúšťa na pozadí príkaz `php artisan crawler:sitemaps`, ktorý prejde každým projektom v databáze a znovu vykoná prehľadávanie súborov *sitemap.xml* a *sitemap_index.xml*.

Každú minútu sa taktiež spúšťa skript pomocou príkazu `php artisan crawler:sites`. Ten zoberie prvých 60 stránok zoradených podľa dátumu posledného skenovanie od najstaršieho po najnovší. Tieto stránky vloží do fronty na skenovanie, ktorá bola spomínaná vyššie.

Pri skenovaní stránky sa do databázy uložia niektoré dáta spomínané v sekcii 3.3 a pre prvých 10 nájdených odkazov na stránku v rámci rovnakej domény, ktoré ešte nie sú uložené v databáze, sa vloží *job* do fronty na skenovanie.

Plánovač ma na starosti správu ešte jedného procesu, ktorý sa spúšťa dvakrát denne pomocou príkazu `php artisan analyzer:search-console`. Tento príkaz získava dáta s *Google Search Console* pomocou Google žetónu uloženého v databáze v zázname projektu. Najskôr sa odošle požiadavka na koncový bod *https://googleapis.com/webmasters/v3/sites*. V úspešnej odpovedi príde zoznam polí, kde každá položka obsahuje doménu alebo celú URL (väčšinou je to adresa domovskej stránky) a taktiež úroveň oprávnení daného užívateľa

(ktorému patrí Google žetón). V tomto zozname sa hľadá záznam zodpovedajúci určitému projektu a s plným prístupom. V prípade, že takýto existuje, odošle sa ďalšia požiadavka na koncový bod *https://googleapis.com/webmasters/v3/sites/[siteUrl]/searchAnalytics/query*, kde `siteUrl` je adresa z predošlej odpovede. Pomocou parametrov v tele požiadavky sa získajú štatistiky vyhľadávania zoskupené podľa celej URL stránky a dátumu vo formáte *Y-m-d*. Tieto dáta sa potom pre adresy, ktoré súčasne existujú aj v databáze projektu uložia do tabuľky `site_seo_performance`.

Rovnakým spôsobom sú potom získané aj dáta pre celý projekt tým, že je vyžiadané ich zoskupenie iba podľa dátumu. Štatistiky v odpovedi sú uložené do tabuľky `project_seo_performance`.

Ďalšie procesy spravované službou Supervisor sa spúšťajú nasledovným príkazom:

```
$ php artisan lighthouse:run-queue %(process_num)02d
```

Jediný argument tohto príkazu je dvojciferné číslo procesu. V konfigurácii každého procesu Supervisora je možné definovať parameter `numprocs`. Ten predstavuje koľko inštancií daného programu sa má spustiť naraz. V tomto prípade by toto číslo malo byť rovnaké ako počet front projektov (viz. sekcia 4.2).

Tento príkaz vykonáva analýzu výkonu stránky pomocou nástroja Lighthouse. Aby toto prebiehalo lokálne a nemuselo sa odosielať požiadavky na Google API, bolo nutné v kontajneri *php-worker* nainštalovať tento nástroj globálne pomocou *npm*. Taktiež sa musel nainštalovať prehliadač Chromium.

```
RUN apk add --update nodejs npm
RUN apk add --update chromium chromium-chromedriver
RUN npm i -g lighthouse
```

Po spustení skriptu sa na základe argumentu zoberú projekty s daným identifikátorom fronty (aj keď v tomto prípade nejde úplne o frontu, ale ide o podobný princíp). Pre každý takýto projekt sa potom zoberie stránka s najstarším parametrom `last_performance_check_at`, ktorý značí, kedy prebehla posledná kontrola výkonu. Pomocou PHP funkcie `shell_exec` sa spustí príkaz `lighthouse` s parametrami `--output=json` a `--output-path=[path]`. Spolu zabezpečia uloženie výsledku v objekte JSON do dočasného súboru. Z toho sú potom načítané dáta jednotlivých metrík výkonu, ktoré sú uložené do tabuľky `site_performance`, kde každý záznam ešte obsahuje dátum dňa (pre každý deň existuje jeden záznam pre danú stránku).

Po uložení dát do databázy sa do základnej fronty vloží *job*, ktorý prepočíta výkon projektu. To prebehne tak, že sa vypočíta priemerná hodnota každej metriky ukladanej do databázy v priebehu určitého dňa. Tieto priemerné hodnoty sa uložia do tabuľky `project_performance`.

Supervisor zabezpečí opätovné spustenie tohto skriptu po jeho ukončení. Takto by mala analýza prebiehať nepretržite a dokola pre všetky stránky.

Posledný definovaný program v službe Supervisor spúšťa nasledujúci príkaz:

```
$ php artisan audits:project-queue %(process_num)02d
```

Princíp fungovania tohto programu je rovnaký ako v predošlom prípade, líšia sa iba samotnou analýzou stránky a výberom stránok na základe parametru `last_check_at`. V tomto prípade ide o analýzu zdrojového kódu, HTTP komunikácie, SEO a zabezpečenia stránky.

Zdrojový kód prvej časti tejto analýzy sa nachádza v kontajneri `SiteAudits` v sekcii `Sites`. Služba `SiteAuditor` najskôr pomocou HTTP klienta získa obsah stránky spolu s hlavičkami. Aktualizuje ukladané dáta v zázname stránky v databáze a spustí audity implementované v zložke `Services/Audits` tohto kontajneru:

- **CompressionAudit** - Z HTTP hlavičiek kontroluje, či bola na prenos dát použitá nejaká kompresia (gzip, deflate, br alebo compress).
- **Http2VersionAudit** - Kontroluje, či boli dáta prenesené HTTP protokolom verzie 2.
- **Http200Audit** - Jednoducho kontroluje, či stránka vracia HTTP kód 200 a nie nejaký chybový kód.
- **MetaDescriptionPresentAudit**, **MetaDescriptionLengthAudit**, **MetaDescriptionUniqueAudit** - Kontrolujú v tomto poradí prítomnosť, dĺžku a unikátnosť meta popisku uloženého v databáze.
- **OgDescriptionPresentAudit**, **OgDescriptionLengthAudit**, **OgDescriptionUniqueAudit** - Kontrolujú v tomto poradí prítomnosť, dĺžku a unikátnosť OpenGraph popisku uloženého v databáze.
- **OgTitlePresentAudit**, **OgTitleLengthAudit**, **OgTitleUniqueAudit** - Kontrolujú v tomto poradí prítomnosť, dĺžku a unikátnosť OpenGraph titulku, uloženého v databáze.
- **TitlePresentAudit**, **TitleLengthAudit**, **TitleUniqueAudit** - Kontrolujú v tomto poradí prítomnosť, dĺžku a unikátnosť titulku stránky uloženého v databáze.
- **SingleUrlFormAudit** - Na základe odosielania požiadaviek na rôzne formy URL (s/bez www na začiatku domény alebo s/bez lomky na konci) kontroluje, či je stránka dostupná iba pod jednou formou URL a ostatné sú na ňu presmerované.
- **SqlInjectionAudit** - Nájde na stránke všetky dotazy s nejakými GET parametrami alebo formuláre s metódou GET. S formulárov a ich vstupných polí zostaví adresy. Zo všetkých adries sa vybere maximálne 8, tak, aby pokryli čo najviac odkazov a formulárov a pomocou metódy Blind SQL Injection sa otestuje prítomnosť tohto rizika.

Ďalšia časť analýzy prebieha pomocou prehliadača Chromium spusteného v „headless“ prostredí. Pomocou príkazu `chromium-browser` sa spustí nový proces na pozadí. Komunikácia s prehliadačom potom prebieha prostredníctvom websocketov a PHP knižnice Wrench[3].

Audity, ktoré sú takto implementované sa nachádzajú v zložke `Services` kontajneru `Audits`:

- **CertificateBrowserAudit** - Pri každej odpovedi na požiadavku, ktorá smeruje na rovnakú domény na akej je auditovaná stránka, kontroluje prítomnosť SSL/TLS certifikátu, jeho platnosť a skutočnosť, že bol vydaný pre správnu doménu.
- **ConsoleLogAudit** - Kontroluje, či sa v konzole prehliadača nevyskytujú žiadne chybové hlášky (najčastejšie ide chyby v JavaScript kóde).

[3]Wrench je PHP knižnica umožňujúca implementáciu websocket servera alebo klienta. Dokumentácia je dostupná na adrese `https://wrench.readthedocs.io/en/latest/index.html`.

- **HttpResponsesAudit** - Analyzuje odpoveď na každý dotaz a kontroluje, či nejaký nevracia chybový kód (väčší ako 400).
- **HttpToHttpsRedirectAudit** - Po otvorení adresy cez protokol HTTP kontroluje, či bola presmerovaná na zabezpečený protokol.

Každý audit má určitú úroveň aplikovanú v prípade zlyhania. Buď je to chyba, varovanie alebo iba informovanie. Po skončení auditov sa ich výsledky uložia do databázy v podobe objektu JSON zakódovaného do formátu *base64*. Audity *CertificateBrowserAudit*, *CompressionAudit* a *Http2VersionAudit* sa ukladajú ako parameter záznamu projektu. Ostatné sa ukladajú do záznamu analyzovanej stránky. Entita stránky taktiež obsahuje informáciu o počte nájdených chýb a varovaní na danej stránke. Rovnaké parametre sa ukladajú aj k entite projektu, tam ale sa ale k počtu chýb a varovaní v rámci troch spomínaných auditov pripočítava aj počet chýb a varovaní na všetkých stránkach k nemu priradených.

Po skončení analýzy a celého skriptu, Supervisor zabezpečí opätovné spustenie. Takto by mala analýza prebiehať nepretržite a dokola pre všetky stránky.

4.3 Klient

SPA klienta je implementovaná v JavaScriptovom frameworku Vue.js 2. Novšia verzia 3 nebola zvolená z toho dôvodu, že sa použila UI knižnica Vuetify, ktorá v čase riešenia projektu ešte nemala plnú podporu pre túto verziu. Projekt, ktorého zdrojový kód sa nachádza v zložke `frontend` v zdrojových súboroch v prílohe A, bol inicializovaný službou `@vue/cli`[4]. Po inicializácii bol vytvorený súbor `package.json` obsahujúci zoznam balíkov potrebných na inštaláciu aplikácie.

Vývoj prebieha v Docker kontajneri `node`, konkrétne v zložke `/var/www/frontend`. Pre inštaláciu závislosti je potrebné spustiť `yarn`. Pre preloženie zdrojového kódu potom príkaz `yarn build`.

Spustením `yarn serve-docker` začne bežať vývojový server vo vnútri kontajneru na porte 3000. V konfiguračnom súbore Docker prostredia je parameter, pomocou ktorého sa tento port mapuje na port hostiteľa, na ktorom je potom aplikácia dostupná cez prehliadač. Táto aplikácia sa, ako už bolo spomenuté v časti 4.1, automaticky obnovuje pri každej zmene zdrojového kódu.

Zdrojový kód ma nasledujúcu štruktúru:

- **src**
  - **assets** - Obrázky použité v aplikácií.
  - **components** - Znovu-použiteľné komponenty.
  - **helpers** - Pomocné funkcie, napríklad na generovanie náhodných reťazcov.
  - **layouts** - Komponenty s rozloženiami základných stránok aplikácie. `LoginPage.vue` obsahuje zdrojový kód prihlasovacej stránky a `App.vue` zas šablónu pre všetky stránky dostupné po prihlásení.
  - **misc** - V tejto zložke sa nachádza kód enumerátorov, rodičovských tried alebo tried, ktoré uľahčujú komunikáciu prostredníctvom REST API so serverom, ako napríklad `ApiResponse` alebo `ApiUrlBuilder`.

[4]Ide o systém, ktorý zrýchľuje vývoj Vue.js aplikácii. Dokumentácia je dostupná na adrese `https://cli.vuejs.org/guide/`.

- **models** - Modely sú triedy, do ktorých sa mapujú položky v odpovediach REST API. Napríklad `ProjectModel`, `UserModel` alebo `SiteModel`.
- **plugins** - Obsahuje konfigurácie knižníc *Vuetify* a *moment* (prácu s dátumami). Taktiež je tu implementovaná služba pre prihlásenie pomocou Google OAuth 2.0 (viz. sekcia 4.4).
- **router** - Implementácia smerovania pomocou knižnice *vue-router*.
- **services** - Obsahuje definície API služieb, ktoré obsahujú metódy implementujúce komunikáciu s určitým koncovým bodom REST API. Napríklad `ProjectsApiService` obsahuje metódy `get` (detail projektu), `getAll` (zoznam projektov), `post` (vytvorenie projektu), `patch` (aktualizácia projektu) a `delete` (zmazanie projektu).
- **store** - Implementácia návrhového vzoru na správu stavu aplikácie pomocou knižnice *vuex*.
- **views** - Vue.js komponenty reprezentujúce konkrétne stránky aplikácie.

Obsah niektorých zložiek bude viac predstavený ďalej v tejto sekcii.

Vuex sklad

Sklad slúži na správu stavu aplikácie, ktorý je spoločný pre všetky komponenty. Okrem stavu existujú ešte mutácie, funkcie, prostredníctvom ktorých je možné meniť stav. Akcie sú niečo podobné. Ich úlohou je však vykonávať rôzne asynchrónne operácie a vyvolávať mutácie. Sklad sa potom ešte skladá z modulov, ktoré majú svoj vlastný stav, mutácie, akcie a ešte v niektorých prípadoch aj tzv. *gettre*, ktoré slúžia na získanie odvodeného (vypočítaného) stavu na základe aktuálneho stavu modulu. Definované sú nasledujúce moduly:

- **alerts** - Obsahuje akcie, zavolaním ktorých je možné zobraziť rôzne notifikácie v pravom hornom rohu obrazovky. Napríklad po úspešnom vytvorení projektu.
- **auth** - Slúži na správu sedenia pomocou JWT žetónov, získanie dát o prihlásenom užívateľovi alebo obsahuje *getter* na získanie oprávnení užívateľa.
- **projects** - Implementácia volaní koncových bodov REST API projektov.
- **projectPerformances** - Implementácia volaní koncových bodov REST API výkonov projektov.
- **projectSeoPerformances** - Implementácia volaní koncových bodov REST API výkonov projektov vo vyhľadávaní.
- **sites** - Implementácia volaní koncových bodov REST API stránok.
- **sitePerformances** - Implementácia volaní koncových bodov REST API výkonov stránok.
- **siteSeoPerformances** - Implementácia volaní koncových bodov REST API výkonov stránok vo vyhľadávaní.
- **users** - Implementácia volaní koncových bodov REST API výkonov stránok vo užívateľov alebo aktualizácie hesla.

- **rightSidebar** - Obsahuje akcie, pomocou ktorých je možné otvoriť pravý bočný panel a nastaviť nech sa v ňom vykreslí nejaká komponenta. Definície takýchto komponent sú v zložke `components/rightSidebar`. Napríklad komponenta na vytvorenie nového projektu, čo je zobrazené na obrázku C.3.

Smerovanie

Pomocou balíka *vue-router* sú definované nasledujúce trasy (*routes*):

- `/` - Zobrazí sa komponenta `views/DashBoardView`. Táto stránka slúži iba ako uvítacia stránka.
- `/nastaveni-uctu` - Zobrazí sa komponenta `views/ProfileSettingsView`. Ako môžeme vidieť na obrázku C.14, na tejto stránke je možné upraviť meno, e-mail a heslo užívateľa.
- `/projekt/:projectId/:tab?` - Zobrazí sa komponenta `views/ProjectDetailView`. Parameter `projectId` je povinný identifikátor projektu a `tab` je nepovinný identifikátor záložky na detaile (viz. sekcia 4.3).

Každá táto trasa je dostupná iba po prihlásení. Na zabezpečenie tejto funkcionality slúžia ochrany navigácie (*navigation guards*). Ochrana `isAuthenticated` skontroluje, či v sklade existuje prihlásený užívateľ, pokiaľ nie, tak sa ho snaží získať pomocou obnovovacieho žetónu. Ak aj to zlyhá, tak presmeruje užívateľa na prihlasovaciu stránku.

Pre tú existuje trasa `/prihlaseni`, ktorá vykresľuje komponentu `layouts/LoginPage`. Táto cesta je chránená ochranou `isUnauthenticated`, ktorá presmeruje prihláseného užívateľa na *dashboard*.

Ešte existuje ochrana navigácie, ktorá obmedzuje prístup na základe oprávnení. Na prístup k detailu projektu sa používa ochrana `canRead`, ktorá zabezpečuje, že sa na danú stránku dostane iba užívateľ s oprávnením čítať projekty.

Rozloženie

Základné rozloženie tvoria ľavý bočný panel, horný panel a samotný obsah, ktorého vykresľovanie na základe konkrétnej trasy zabezpečuje *vue-router*.

Ľavý bočný panel obsahuje odkazy na detaily projektov. Tlačidlo na vytvorenie projektu slúži na otvorenie pravého bočného panelu, kde môže užívateľ vytvoriť nový projekt. Toto tlačidlo vidí iba administrátor. Jeho chovanie je naznačené na obrázku C.3.

V hornom paneli sa nachádza vstupné pole, ktoré slúži na vyhľadávanie projektov podľa domény alebo názvu. Na pravom okraji panelu je tlačidlo s menom prihláseného užívateľa. Po kliknutí naň sa rozbalí menu, kde je tlačidlo na odhlásenie a odkaz na stránku s nastaveniami profilu.

Detail projektu

Detail projektu je zobrazený na obrázku C.2 a implementovaný v komponente `views/ProjectDetailView`. Nachádzajú sa tu dve tlačítka, ktoré môže vidieť iba administrátor. Jedno je na úpravu projektu (otvorí pravý bočný panel, viz. obrázok C.4) a druhé na zmazanie projektu (túto akciu je nutné potvrdiť, viz. obrázok C.5).

Ďalej sú na tejto stránke záložky s rôznym obsahom, ktoré fungujú ako odkazy. Konkrétna záložka sa aktivuje na základe parametru súčasnej cesty (ako bolo popísané v časti 4.3). Obsah každej záložky je definovaný v komponentoch, ktoré sú umiestnené v zložke `components/projectTabs`.

`DashboardTab` obsahuje zdrojový kód základnej záložky, kde je zobrazený počet nájdených stránok, dátum posledného skenovania nejakej stránky, dátum poslednej analýzy nejakej stránky, počet chýb a počet varovaní nájdených na stránkach projektu. V spodnej časti sa nachádzajú základné audity projektu. Snímka tejto záložky je na obrázku C.2.

`PagesTab` je definícia záložky s analýzou jednotlivých stránok, ktorá obsahuje tabuľku stránok. V tejto tabuľke je zobrazený HTTP kód každej stránky a počet chýb a varovaní (obrázok C.6).

Po kliknutí na tlačidlo „zobraziť analýzy“ sa v záložke vykreslí komponenta `SiteDetail`. Ako je zobrazené na snímke obrazovky C.7, nachádza sa tu odkaz späť na zoznam stránok a nižšie výsledky jednotlivých auditov stránky. Po kliknutí na nejaký audit sa otvorí pravý bočný panel, ktorý obsahuje viac informácií (viz. obrázok C.8).

Záložka „výkon“ obsahuje posledné hodnoty metrík výkonu pre celý projekt. Nad každou hodnotou je graf, ktorý naznačuje vývoj za posledný mesiac. Rovnako sú tieto metriky zobrazené v tabuľke stránok, kde je ešte aj zobrazený dátum poslednej analýzy výkonu. Snímka záložky je na obrázku C.9. Toto je implementované v komponente `PerformanceTab`.

Po kliknutí na tlačidlo „zobraziť grafy“ sa vykreslí komponenta `SitePerformanceDetail`, ktorá obsahuje graf vývoja výkonu stránky za posledný mesiac. Ku každej metrike sú v spodnej časti vysvetlivky. Graf je vykreslený pomocou balíka *vue-chartjs*[5]. Snímka detailu stránky je na obrázku C.10.

Podobne ako záložka „výkon“ je implementovaná aj záložka „výkon vo vyhľadávaní“ v komponentách `SeoPerformanceTab` a `SiteSeoDetail`. Snímky sú na obrázkoch C.11 a C.12.

Posledná záložka obsahuje bezpečnostné audity projektu. Implementovaná je v komponente `SecurityTab` a snímka sa nachádza na obrázku C.13.

Autentizácia

Prihlásenie sa vykoná odoslaním e-mailu a heslo na koncový bod REST API, `POST /api/auth/user-sessions`. Pokiaľ sa vráti úspešná odpoveď obsahujúca JWT žetóny, prebehne presmerovanie na *dashboard* pomocou volania `window.location.replace`. Do URL sa vloží GET parameter `token` obsahujúci obnovovací žetón z odpovede. Po presmerovaní sa tento žetón z URL odošle na koncový bod `POST /api/auth/user-tokens`, ktorý vráti ďalší pár prístupového a obnovovacieho žetónu. Ktoré sú v tomto poradí uložené do lokálneho úložiska (`window.localStorage`) pod kľúčmi `bearer-token` a `refresh-token`. Prístupový žetón je vložený do hlavičky `Authorization` každej požiadavky na REST API. Pri odhlásení sú tieto žetóny z lokálneho úložiska odstránené. V prípade že prístupový žetón prestane platiť, znovu sa odošle požiadavka na `POST /api/auth/user-tokens`. Pokiaľ prestane platiť aj obnovovací žetón, oba žetóny sú rovnako odstránené z lokálneho úložiska.

[5]Odkaz na dokumentáciu: `https://vue-chartjs.org/`.

4.4 Získanie Google žetónu

Implementácia získania Google žetónu je v aplikácií klienta v súbore `plugins/gapi.js`. Po kliknutí na tlačidlo „prihlásiť pomocou OAuth“, pri vytváraní alebo upravovaní projektu, sa zavolá funkcia `getGoogleTokenData`, ktorá vytvorí *Promise*, v ktorom sa v novom *pop-up* okne (zavolaním funkcie `window.open`) otvorí adresa na doméne servera otvorí cesta `/google-api/oauth`.

Na serveri sa pomocou PHP knižnice *google/apiclient*[6] vytvorí adresa, na ktorú prebehne presmerovanie a užívateľ sa na nej prihlási pomocou svojho Google účtu. Pomocou knižnice je potrebné nastaviť tzv. rozsah (*scopes*). Pre potreby je aplikácie sa nastavuje nasledovný rozsah:

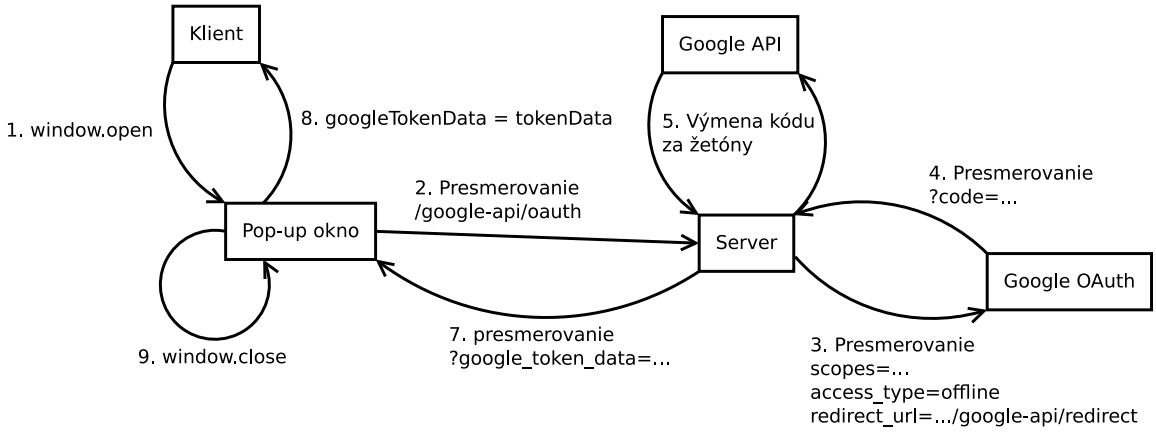
- `https://www.googleapis.com/auth/webmasters.readonly` pre prístup k dátam Google Search Console a
- `https://www.googleapis.com/auth/userinfo.email` pre získanie e-mailu užívateľa.

Musí sa nastaviť parameter `access_type` na hodnotu `offline`, aby bolo možné získať obnovovací žetón a užívateľovi by tým stačilo sa prihlásiť iba raz.

Posledným dôležitým nastavením je adresa presmerovania, ktorá sa nastavuje na adresu servera s cestou `/google-api/redirect`. Na túto adresu prebehne presmerovanie po úspešnom prihlásení a v GET parametri `code` bude autorizačný kód, pomocou ktorého sa získa prístupový a obnovovací žetón znovu pomocou knižnice od Google. Po získaní žetónu sa jeho dáta zakódujú do objektu JSON, ktorý sa ešte zakóduje do *base64*. Potom prebehne presmerovanie na domovskú adresu klienta s GET parametrom `google_token_data`, v ktorom je zakódovaný žetón.

Toto všetko sa deje v *pop-up* okne. Pri inicializácii JavaScript kódu klienta, sa zavolá funkcia, ktorá je tiež implementovaná v súbore `plugins/gapi.js`, `getGoogleTokenData`. V tejto funkcii sa skontroluje, či existuje GET parameter `google_token_data` a objekt `window.opener`. Pokiaľ áno, žetón sa dekóduje najskôr z *base64* do JSON reťazca a z toho do normálneho JavaScript objektu. Tento objekt je potom nastavený ako parameter `window.opener.googleTokenData`, čím sa stane prístupný v pôvodnom okne. Nakoniec sa zavolá funkcia `window.close`. Po zavretí okna sa vyrieši (*resolve*) *Promise* vytvorený vo funkcii `getGoogleTokenData`. Užívateľovi sa pod tlačidlom zobrazí, ktorým e-mailom sa prihlásil a po odoslaní dát na server sa odošlú aj dáta žetónu. Tento proces je znázornený v diagrame 4.1.

[6] Knižnica oficiálne podporovaná spoločnosťou Google, ktorá umožňuje prácu s Google API. Repozitár je dostupný na adrese (`https://github.com/googleapis/google-api-php-client`). Táto knižnica je používaná na vytvorenie všetkých požiadaviek na Google API.



Obr. 4.1: Proces získania Google žetónu pomocou Google OAuth 2.0.

Kapitola 5

Testovanie

V tejto kapitole je popísaný spôsob testovania aplikácie, ktoré prebiehalo zároveň s vývojom. Cieľom bolo vyladenie chýb v užívateľskom rozhraní a overenie správnosti výsledkov analýz testovaných webov, a taktiež overenie funkčnosti skeneru stránok. Aplikácia počas vývoja ešte nebola nasadená na žiadny produkčný ani vývojový server, takže všetky testy prebiehali na lokálnom stroji.

5.1 Testovanie UI

Pri testovaní užívateľského rozhrania nastal práve problém s tým, že aplikácia nebola nasadená na žiadny produkčný server. Avšak samotný klient slúži viac na prezentáciu dát a výsledkov získaných na pozadí servera, a preto bolo potrebné otestovať iba pár úkonov, ako prihlásenie, upravenie profilu, CRUD operácie nad projektom, vyhľadávanie v projektoch a orientáciu v prezentovaných dátach analýzy nejakého webu. Testovanie s používateľmi na lokálnom stroji, na ktorom aj prebiehal vývoj nebolo preto ťažké zorganizovať a nebolo ani časovo náročné.

Na testovanie boli použité nasledujúce osoby:

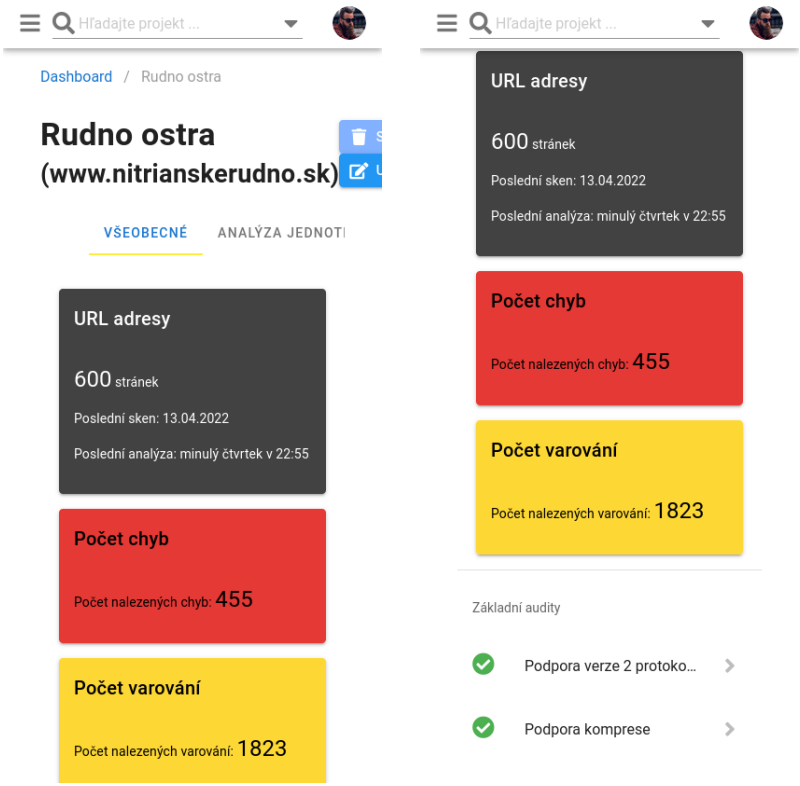
- webmaster vo veku 33 rokov,
- projektový manažér vo veku 30 rokov,
- vývojárka klientskych aplikácií vo veku 27 rokov a
- PHP programátor vo veku 25 rokov.

Testované boli nasledujúce prípady užitia:

- vytvorenie, úprava a zmazanie projektu,
- vyhľadávanie v projektoch,
- získanie výsledku nejakého auditu spusteného na konkrétnej stránke v rámci projektu,
- získanie výsledku nejakého auditu spusteného na konkrétnej stránke v rámci projektu,
- orientácia v grafoch a zistenie hodnoty určitej metriky na konkrétnej stránke,
- úprava profilu a hesla.

Odozva na prácu s užívateľským rozhraním bola pozitívna. Medzi nedostatky, ktoré sa zároveň podarilo vyriešiť patrili prezentácia chybových hlášok pri práci z formulármi, informovanie o zmene stavu na serveri alebo obsah kartičiek na úvodnej karte detailu projektu a dôležité informácie, ktoré sa v nich zobrazujú. Najväčším nedostatkom však bola absencia vyhľadávania v tabuľkách stránok. Z časových dôvodov sa toto nestihlo implementovať.

Testovala sa aj práca s aplikáciou na mobilnom zariadení. Pri tomto sa zistilo, že aplikácia je v tomto prostredí takmer plne použiteľná, a to bez toho, že by na to bol pri vývoji kladený dôraz. Za túto skutočnosť môže hlavne použitie knižnice Vuetify. Niektoré tlačidlá a texty však neboli úplne viditeľné alebo iné prvky neboli správne zarovnané. Mobilná verzia tak bude v budúcnosti potrebovať menšie úpravy na to, aby sa stala použiteľnou v rovnakej miere ako desktopová verzia. Na obrázkoch 5.1 je vidieť napríklad neúplné tlačidlá alebo nekompletný text v kartičke jedného auditu. Nesprávne zarovnanie je zas viditeľné na obrázku 5.3.



Obr. 5.1: Snímky obrazovky z testovania mobilnej verzie aplikácie.

5.2 Testovanie servera

Prvá časť testovania servera bola zameraná na odhalenie chýb pri spracovaní požiadaviek na REST API. K tomuto účelu sa použil voľne dostupný nástroj Postman[1]. Overovala sa správnosť HTTP kódov v odpovediach, funkčnosť koncových bodov API prostredníctvom správnych HTTP metód, či chybové kódy a chybové hlášky v telách odpovedí.

[1]Dostupný na stiahnutie na adrese `https://www.postman.com/downloads/`

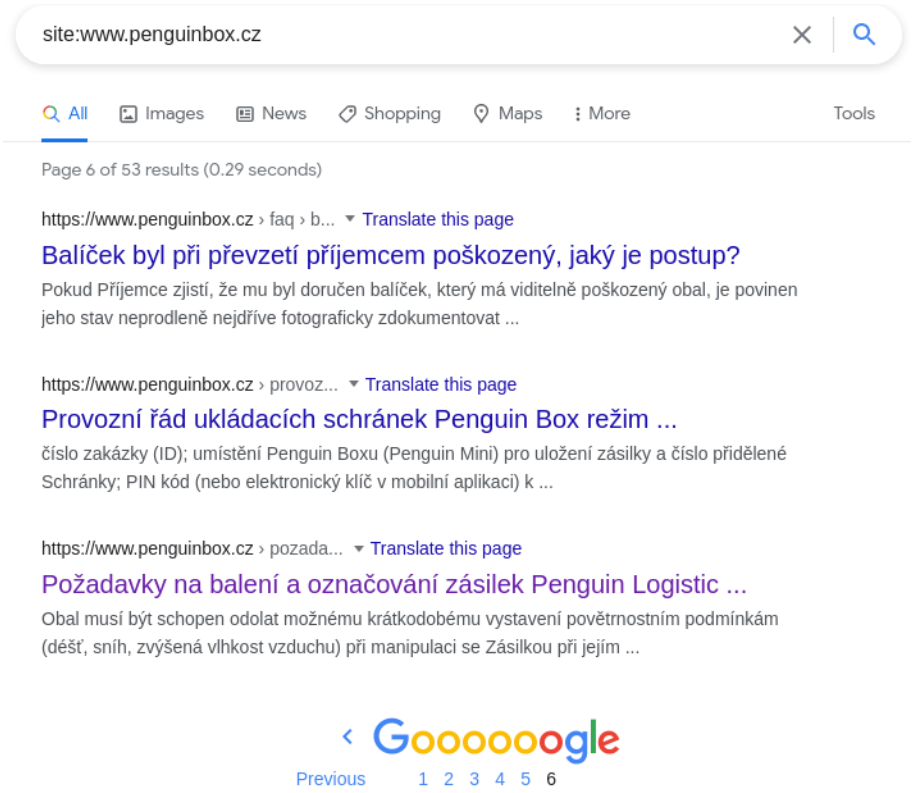
Na testovanie ďalších častí bolo zvolených 5 informačných webov a jeden internetový obchod. Jeden z webov bežal vo vývojovom prostredí, takže bolo možné otestovať odhaľovanie viacerých chýb.

V druhej časti sa zameralo na overenie funkčnosti jednotlivých procesov bežiacich na pozadí servera. Skener webu sa testoval porovnávaním dát v Google Search Console alebo počtom stránok na danej doméne v indexe Google, ktorý sa testoval zadaním podobného reťazca, ako v nasledujúcom príklade, do vyhľadávania Google.

```
site:www.penguinbox.cz
```

Napríklad pre tento konkrétny výraz bolo nájdených 53 výsledkov. V aplikácií bolo v databáze uložených 55 výsledkov, ktoré boli všetky unikátne. Navyše mohli byť stránky, ktoré mali zakázané indexovanie alebo sa ich Google rozhodol z nejakého dôvodu neindexovať.

Na obrázku 5.2 je možné v hornej časti vidieť počet stránok na danej doméne v indexe Google.

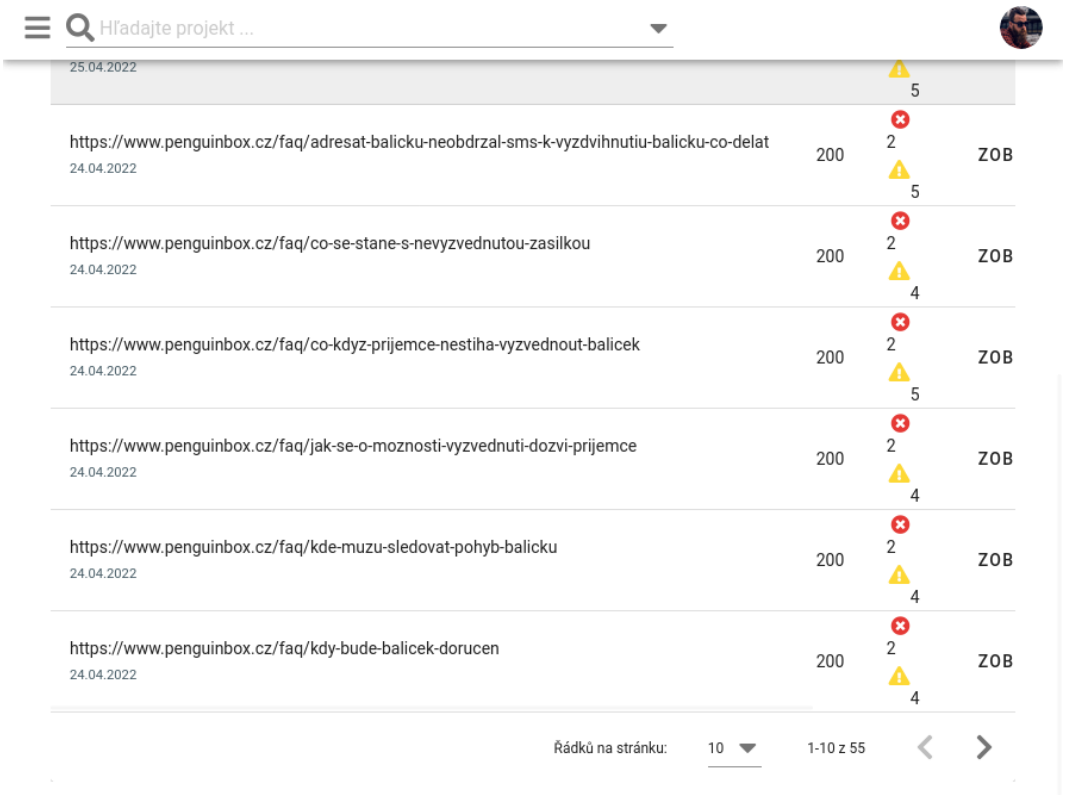


Obr. 5.2: Vyhľadávanie všetkých stránok na určitej doméne.

Na obrázku 5.3 je zas v spodnej časti vidieť počet stránok v databáze aplikácie.

Nástroj na získavanie metrík výkonu stránok, Lighthouse, poskytuje, v prípade, že je spustený v lokálnom prostredí, výsledky závislé napríklad od internetového pripojenia alebo hardvéru, a taktiež od nastavenia behu v mobilnom alebo desktopovom prostredí. Pri testovaní sa zvolilo mobilné prostredie, v ktorom sa aplikácie spravidla načítajú pomalšie. Získavané dáta boli porovnávané s dátami, získanými po spustení nástroja na adrese `https://pagespeed.web.dev/`, kde je tento nástroj dostupný na použitie online.

Validita výsledkov jednotlivých analýz stránky sa testovala porovnávaním ich výsledkov s reálnym stavom stránky. Nešlo o žiadny zložitý proces testovania, skôr iba o overenie,



Obr. 5.3: Zoznam stránok s ich počtom v mobilnej verzii aplikácii.

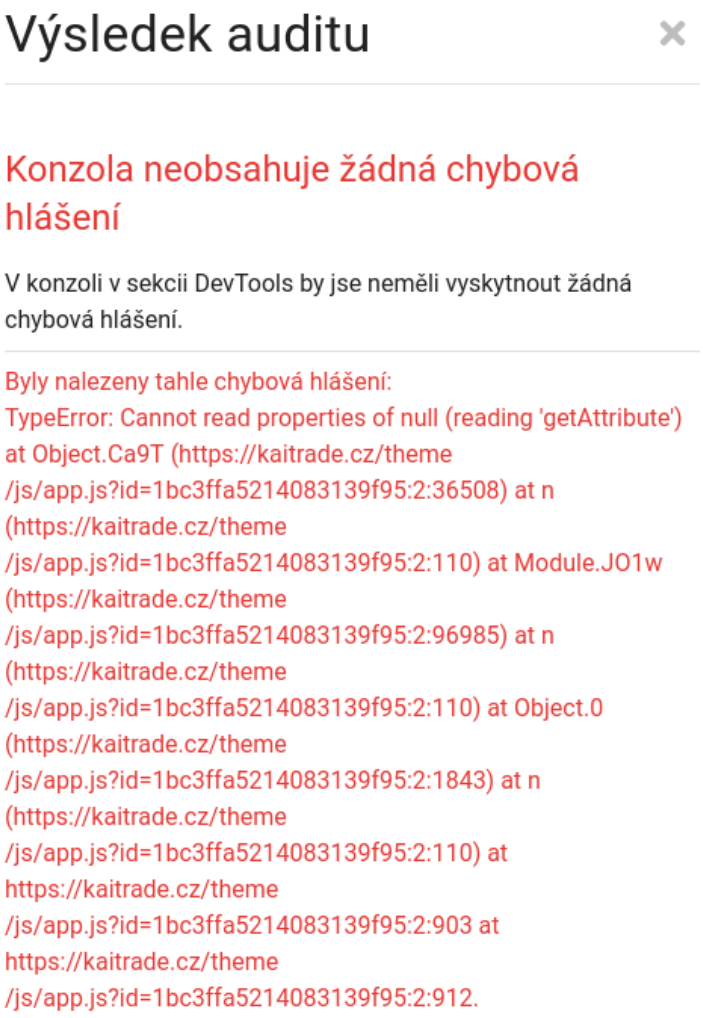
že výsledky zodpovedajú tomu, v akom stave sa z ich pohľadu analyzovaný web naozaj nachádza.

Pri testovaní aplikácie sa narazilo na problém s odosielaním veľkého počtu požiadaviek na server, kde bol hosťovaný analyzovaný web. Toto sa vyriešilo ukladaním obsahu HTTP hlavičky `Retry-After` do dočasnej pamäte pod kľúčom, ktorý identifikuje daný projekt. Do vypršania doby v uvedenej hlavičke sa neodosielajú žiadne požiadavky na analyzovaný web, a teda neprebieha ani skenovanie a ani samotná analýza. Po vypršaní tejto doby začnú zase znovu fungovať.

5.3 Zhodnotenie testovania

Počas testovania sa podarilo odhaliť viaceré nedostatky na testovaných weboch. Patrili sem napríklad nedostatočná implementácia niektorých SEO prvkov, chybové hlásenia v konzole prehliadača (možné vidieť na obrázku 5.4), neúspešné požiadavky (napríklad na neexistujúce obrázky) alebo chýbajúce presmerovanie HTTP na HTTPS. Taktiež sa podarilo odhaliť nefungujúce obnovovanie SSL/TLS certifikátu po tom, čo v aplikácií vyskočilo upozornenie. V rámci jedného webu sa našlo 14 existujúcich nefunkčných stránok (HTTP kód 500).

Analyzovaním výkonu sa zas podarilo odhaliť pomalé stránky.



Obr. 5.4: Výsledok neúspešného auditu chybových hlásení v konzole prehliadača.

Kapitola 6

Záver

Cieľom tejto práce bolo navrhnúť a implementovať webovú aplikáciu umožňujúcu skenovanie a pravidelné analyzovania optimalizácie a zabezpečenia informačných webov alebo internetových obchodov.

V prvej časti boli popísané viaceré prvky stránok ovplyvňujúce rýchlosť načítavania, hodnotenie vyhľadávačmi alebo zraniteľnosť z pohľadu bezpečnosti. Okrem toho prebehol prieskum už existujúcich nástrojov a služieb, z ktorých niektoré boli zakomponované do výsledného riešenia.

Potom bola aplikácia vyvíjaná pomocou moderných nástrojov v kontajnerizovanom prostredím, vďaka ktorému je budúce nasadenie na produkčný server jednoduchšie. Na implementáciu serverovej časti sa použil PHP rámec Laravel s architektúrou Porto, ktorá značne zjednodušuje rozšírenia o nové logické celky, ale aj úpravu tých starých, hlavne vďaka samotnej organizácii zdrojového kódu.

Klientská časť vyvinutá vo Vue.js umožňuje spravovať projekty (reprezentujú analyzovaný web) a slúži ako nástroj na prezentáciu stavu webu z pohľadu vykonaných analýz a napojených nástrojov a služieb. Obrovskú výhodu a urýchlenie vývoja prinieslo použitie knižnice Vuetify.

Na analýzu výkonu bol použitý nástroj Lighthouse a bolo zabezpečené napojenie a zobrazovanie dát z Google Search Console.

Taktiež boli implementované ďalšie audity. Niektoré prebiehajú analýzou HTTP komunikácie, konkrétne hlavičiek a tiel odpovedí. Ďalšia séria auditov je spúšťaná v „headless“ prostredí prehliadača Chromium. Na testovaných stránkach sa podarilo v prípade jedného webu zachytiť vypršanie SSL/TLS certifikátu, v inom zase 14 nefunkčných stránok, či prítomnosť javascriptových chybových hlášok v konzole prehliadača. Analyzovanie, zber dát a skenovanie stránok prebiehajú na pozadí servera.

Cieľ práce sa podarilo naplniť ale v budúcnosti budú potrebné ďalšie rozšírenia. Plánujem ďalej implementovať *WebSocket* server na zabezpečenie okamžitého informovania o zmenách stavu v procese analýzy či skenovania na strane klienta. Ďalej chcem aplikáciu rozšíriť o systému notifikácii, hlavne pomocou e-mailu, prípadne sms.

Aplikácia by sa tiež dala rozšíriť o ďalšie analýzy alebo externé služby. Užitočné by bolo aj inteligentnejšie skenovanie stránok, kedy by sa napríklad ignorovali niektoré stránky s podobnou URL.

Literatúra

[1] *Cache-Control* [online]. Mozilla, 2005 - 2021 [cit. 2022-01-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control.

[2] *Content-Security-Policy* [online]. Mozilla, 2005 - 2021 [cit. 2022-01-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy.

[3] *Cross-Origin-Opener-Policy* [online]. Mozilla, 2005 - 2021 [cit. 2022-01-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cross-Origin-Opener-Policy.

[4] *An overview of HTTP* [online]. Mozilla, 2005 - 2021 [cit. 2021-12-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[5] *Referrer-Policy* [online]. Mozilla, 2005 - 2021 [cit. 2022-01-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy.

[6] *Set-Cookie* [online]. Mozilla, 2005 - 2021 [cit. 2022-01-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie.

[7] *Browser Rendering: JS + DOM + CSSOM* [online]. ProgrammingSoup, 2019-2021 [cit. 2022-01-17]. Dostupné z: https://programmingsoup.com/browser-rendering.

[8] *All Reports* [online]. http archive, 2021 [cit. 2021-12-28]. Dostupné z: https://httparchive.org/reports.

[9] *Canonicalization* [online]. Mozilla, Inc., 2021 [cit. 2021-12-29]. Dostupné z: https://moz.com/learn/seo/canonicalization.

[10] *Google images best practices* [online]. Google, 2021 [cit. 2021-12-30]. Dostupné z: https://developers.google.com/search/docs/advanced/guidelines/google-images.

[11] *HTTP - Overview* [online]. Tutorials Point, 2021 [cit. 2021-12-02]. Dostupné z: https://www.tutorialspoint.com/http/http_overview.htm.

[12] *Introduction to robots.txt* [online]. Google, 2021 [cit. 2021-12-29]. Dostupné z: https://developers.google.com/search/docs/advanced/robots/intro.

[13] *Introduction. What is Vue.js?* [online]. Vue.js, 2021 [cit. 2022-01-15]. Dostupné z: https://v3.vuejs.org/guide/introduction.html.

[14] *Learn about sitemaps* [online]. Google, 2021 [cit. 2021-12-29]. Dostupné z: https://developers.google.com/search/docs/advanced/sitemaps/overview.

[15] *Lighthouse* [online]. Google, 2021 [cit. 2022-01-02]. Dostupné z: https://developers.google.com/web/tools/lighthouse.

[16] *Meta tags that Google understands* [online]. Google, 2021 [cit. 2021-12-30]. Dostupné z: https://developers.google.com/search/docs/advanced/crawling/special-tags.

[17] *Method: urlTestingTools.mobileFriendlyTest.run* [online]. Google, 2021 [cit. 2022-01-02]. Dostupné z: https://developers.google.com/webmaster-tools/search-console-api/reference/rest/v1/urlTestingTools.mobileFriendlyTest/run.

[18] *Redirects* [online]. Mozilla, Inc., 2021 [cit. 2021-12-29]. Dostupné z: https://moz.com/learn/seo/redirection.

[19] *Redis* [online]. Redis Ltd., 2021 [cit. 2022-01-15]. Dostupné z: https://redis.io/.

[20] *Rich media file best practices* [online]. Google, 2021 [cit. 2021-12-30]. Dostupné z: https://developers.google.com/search/docs/advanced/guidelines/rich-media-files.

[21] *Rich results, structured data and Schema: a visual guide to help you understand* [online]. Yoast, 2021 [cit. 2021-12-31]. Dostupné z: https://yoast.com/rich-results-schema-structured-data-story.

[22] *Search Analytics: query* [online]. Google, 2021 [cit. 2022-01-02]. Dostupné z: https://developers.google.com/webmaster-tools/search-console-api-original/v3.

[23] *Search Engine Market Share Worldwide* [online]. StatCounter, november 2021 [cit. 2021-12-02]. Dostupné z: https://gs.statcounter.com/search-engine-market-share.

[24] *Video best practices* [online]. Google, 2021 [cit. 2021-12-30]. Dostupné z: https://developers.google.com/search/docs/advanced/guidelines/video.

[25] *What is Vuex?* [online]. Vue.js, 2021 [cit. 2022-01-15]. Dostupné z: https://next.vuex.vuejs.org/.

[26] *Installation. Why Laravel?* [online]. Laravel LLC., 2022 [cit. 2022-01-15]. Dostupné z: https://laravel.com/docs/8.x/installation#why-laravel.

[27] *Introduction. What is Vuetify?* [online]. Vuetify, 2022 [cit. 2022-01-16]. Dostupné z: https://vuetifyjs.com/en/introduction/why-vuetify/.

[28] *MySQL* [online]. Oracle Corporation, 2022 [cit. 2022-01-15]. Dostupné z: https://www.mysql.com/.

[29] *Open Web Application Security Project* [online]. OWASP Foundation, Inc., 2022 [cit. 2022-01-05]. Dostupné z: https://owasp.org/.

[30] Chapter, O. G. *Use of Web Application Firewalls* [online]. 2008 [cit. 2022-01-06]. Dostupné z: hhttps://owasp.org/www-pdf-archive/Best_Practices_WAF_v105.en.pdf.

[31] Danny Dover, E. D. *Search Engine Optimization (SEO) Secrets*. 1. vyd. Wiley, 2011. ISBN 0470554185; 9780470554180. Dostupné z: libgen.li/file.php?md5=03bd4067ad7556915946ab342c84a45f.

[32] Grigorik, I. *High Performance Browser Networking*. 1. vyd. O'Reilly Media, 2013. ISBN 978-14-493-4476-4.

[33] Jero. *Understanding DOM, CSSOM, Render Tree, Layout, and Painting.* [online]. Medium, 2020 [cit. 2021-12-28]. Dostupné z: https://medium.com/weekly-webtips/understand-dom-cssom-render-tree-layout-and-painting-9f002f43d1aa.

[34] Mahmoud Zalt. *Porto (Software Architectural Pattern)* [online]. GitHub, Inc., 2021 [cit. 2022-01-15]. Dostupné z: https://github.com/Mahmoudz/Porto.

[35] Mueller, J. P. *Security for Web Developers: Using JavaScript, HTML, and CSS.* O'Reilly Media, 2015. ISBN 9781491928646; 1491928646. Dostupné z: libgen.li/file.php?md5=fb8906ebc2e76c9ae24837fa73156726.

[36] Ulfar Erlingsson, Y. X. *End-to-end Web Application Security* [online]. [cit. 2022-01-02]. Dostupné z: https://www.usenix.org/legacy/events/hotos07/tech/full_papers/erlingsson/erlingsson.pdf.

[37] Wikipedia contributors. *Single-page application* [online]. 2022 [cit. 2022-01-15]. Dostupné z: https://en.wikipedia.org/wiki/Single-page_application.

Príloha A

Obsah pamäťového média

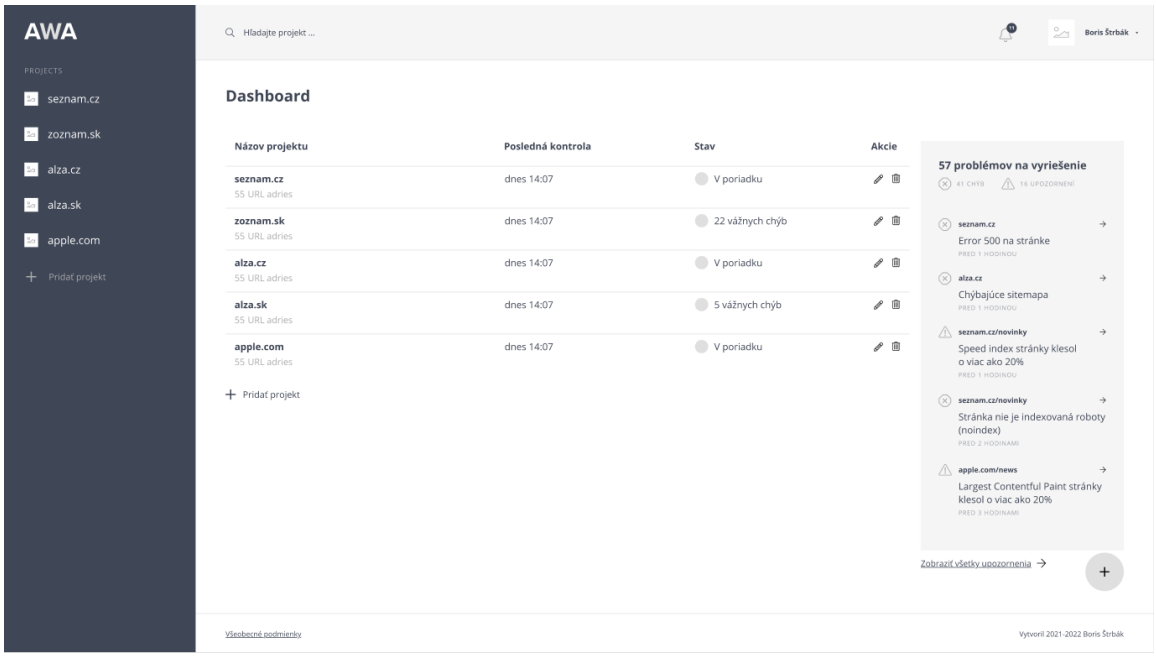
V tejto prílohe je popísaná základná štruktúra obsahu priloženého pamäťového média.

- **xstrba05.pdf** - táto práca
- **xstrba05.zip** - zdrojové súbory tejto práce
- **app.zip** - zdrojové súbory aplikácie v zložke `app` s nasledujúcou štruktúrou:
  - **frontend** - zdrojové súbory klienta
  - **backend** - zdrojové súbory servera
    - **docker** - zdrojové a konfiguračné súbory docker prostredia
    - **README.md** - návod ako lokálne preložiť a spustiť projekt

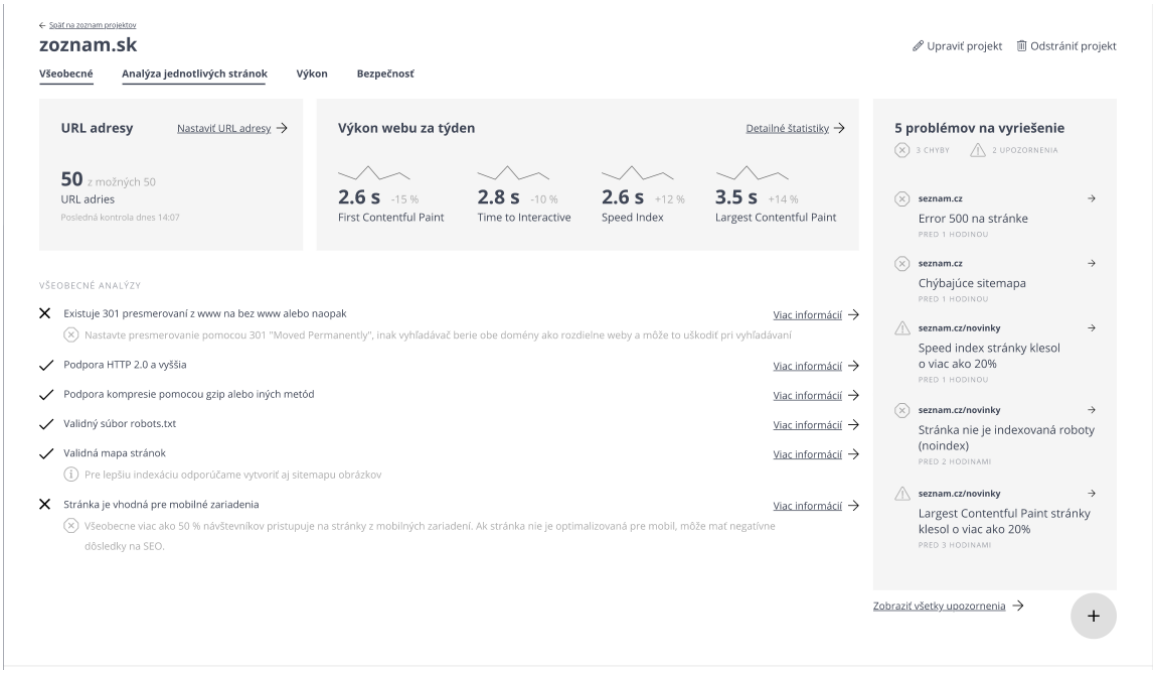
Príloha B

Návrhy užívateľského rozhrania

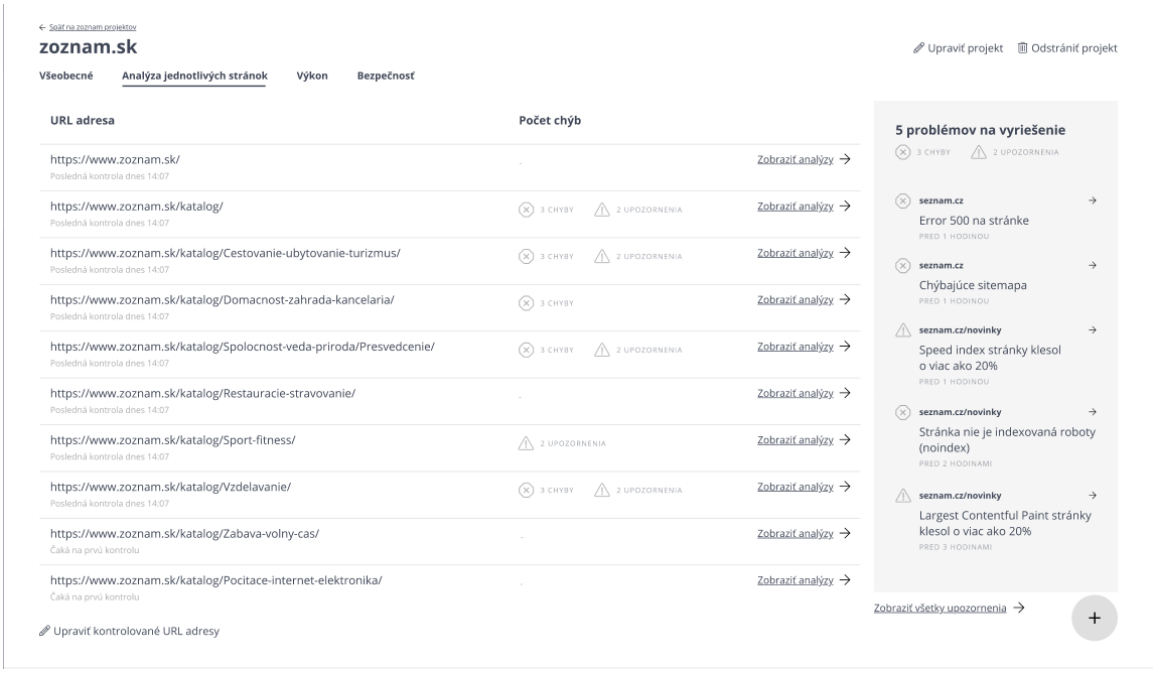
V tejto prílohe sa nachádzajú návrhy užívateľského rozhrania aplikácie.



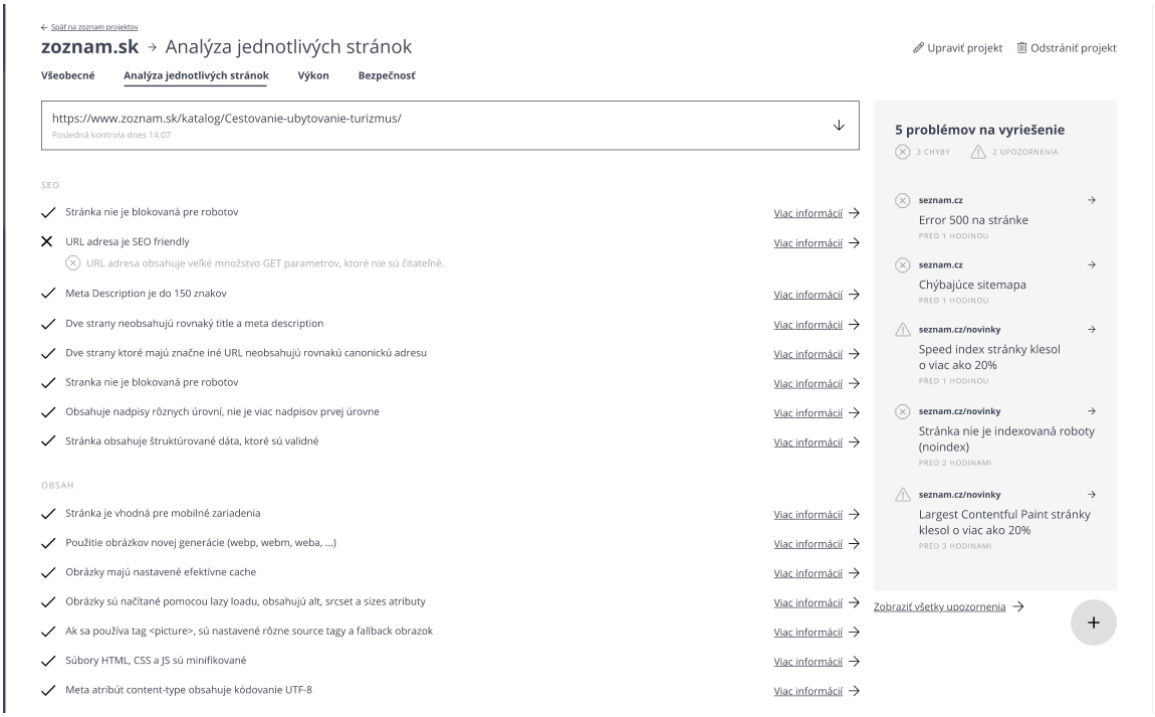
Obr. B.1: Návrh *dashboardu* s prehľadom všetkých projektov.



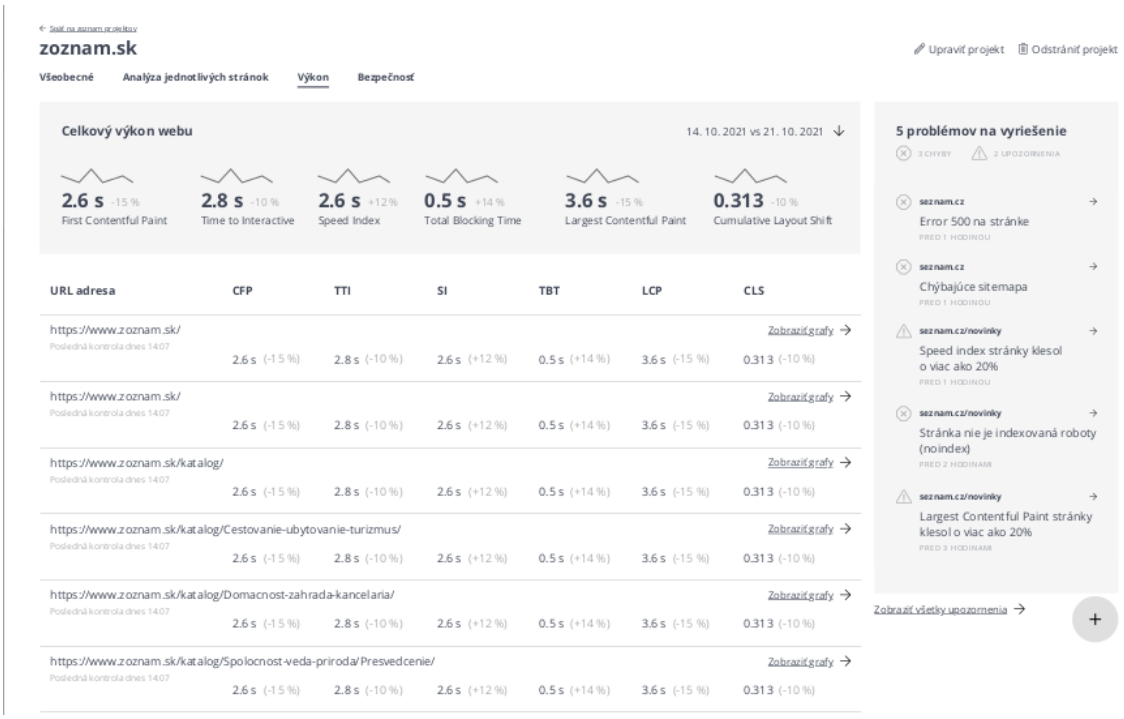
Obr. B.2: Návrh detailu projektu.



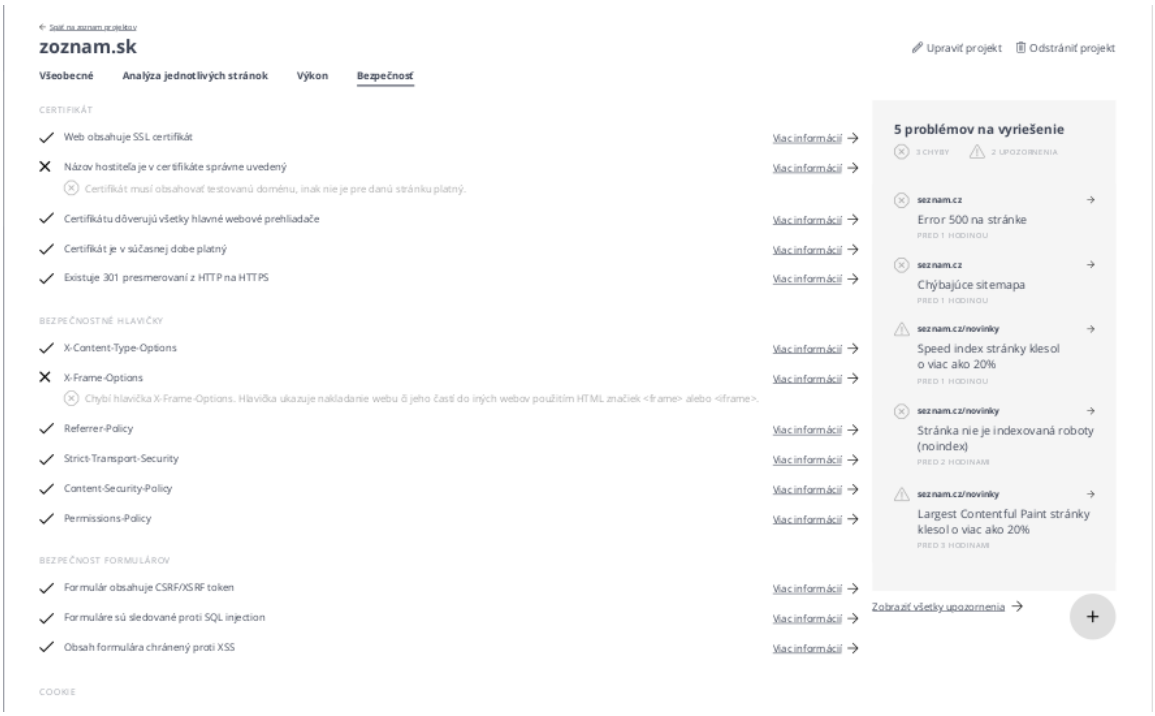
Obr. B.3: Návrh zoznamu stránok webu.



Obr. B.4: Návrh detailu stránky.



Obr. B.5: Detail výkonu webu.

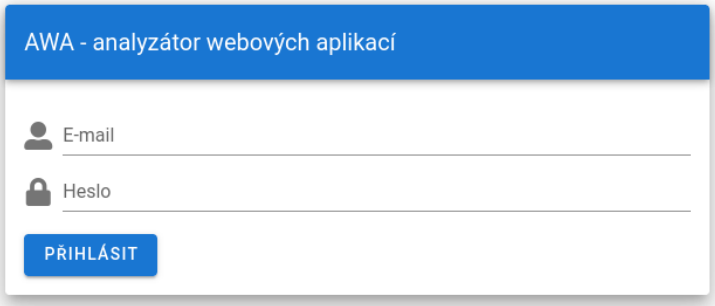


Obr. B.6: Detail analýzy zabezpečenia webu.

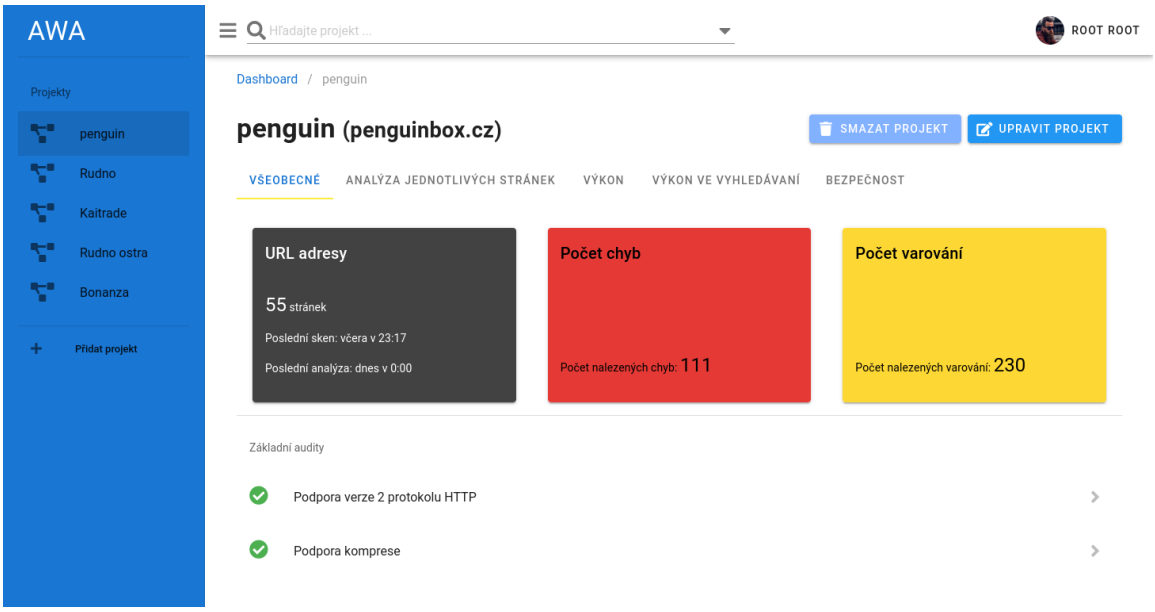
Príloha C

Snímky obrazovky desktopovej verzie aplikácie

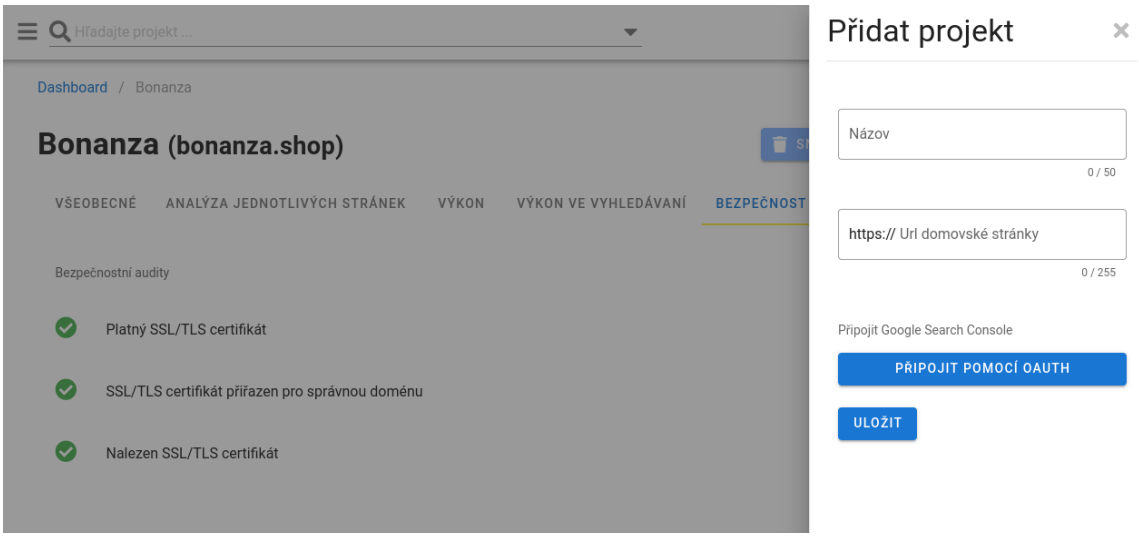
V tejto prílohe sa nachádzajú snímky obrazovky aplikácie v desktopovej verzii.



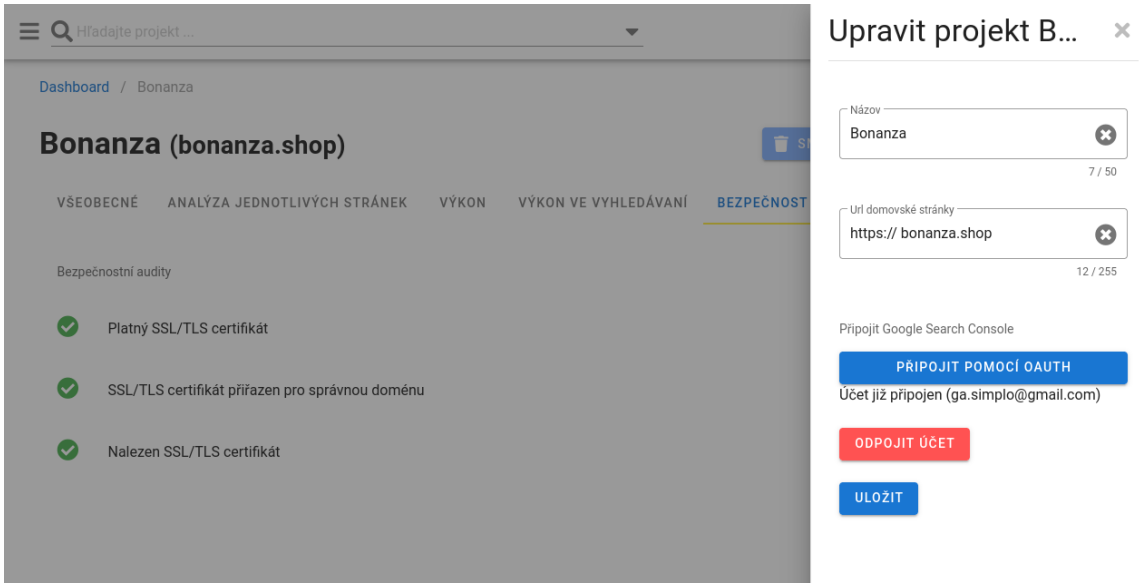
Obr. C.1: Snímka prihlasovacej obrazovky.



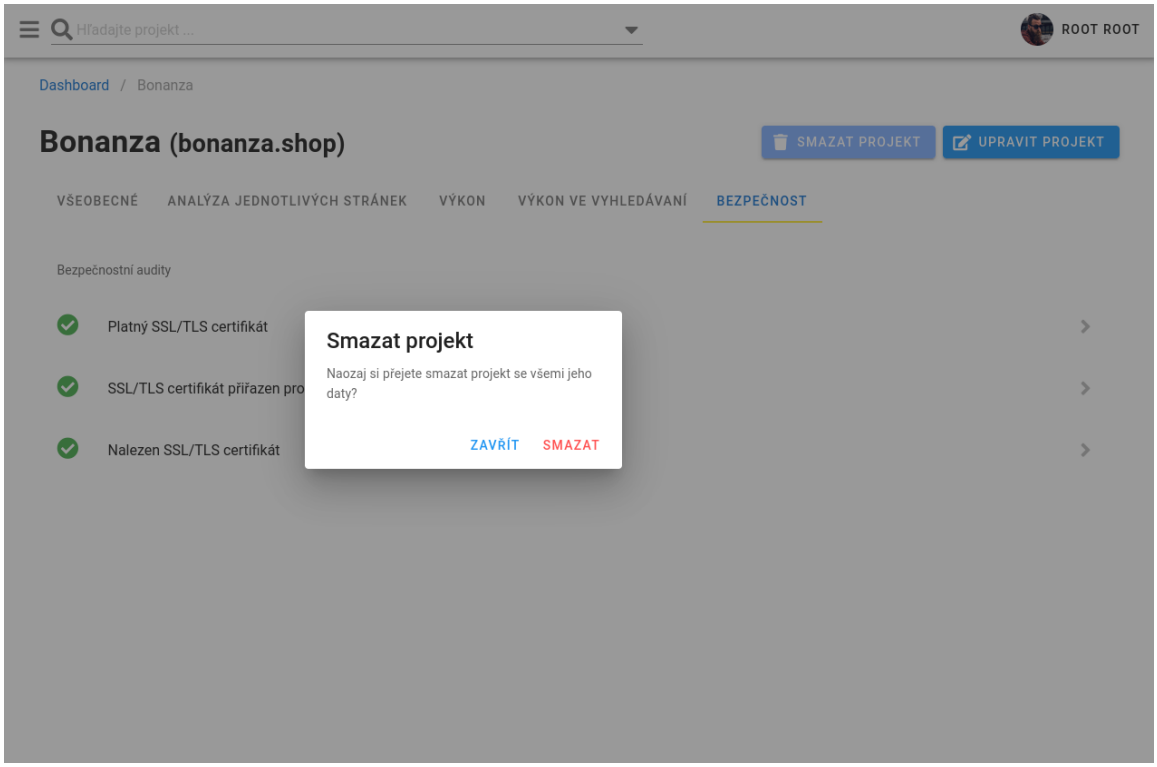
Obr. C.2: Snímka detailu projektu so základným prehľadom.



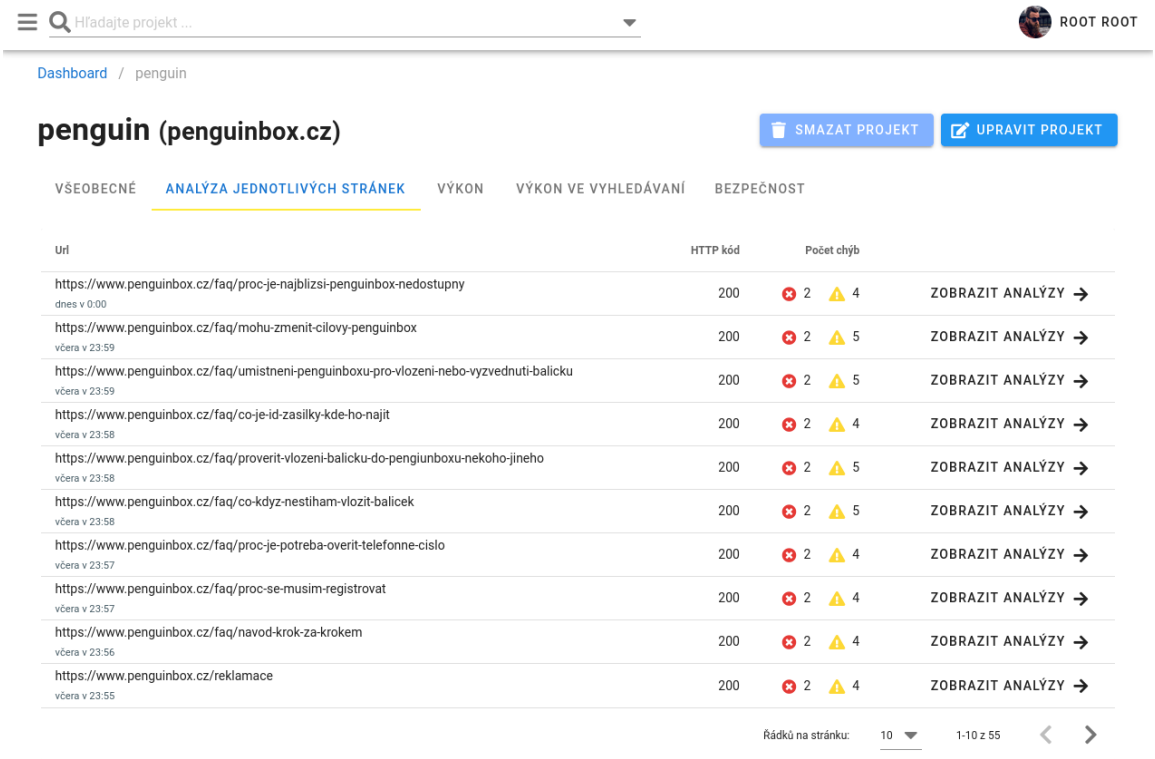
Obr. C.3: Snímka vytvárania projektu.



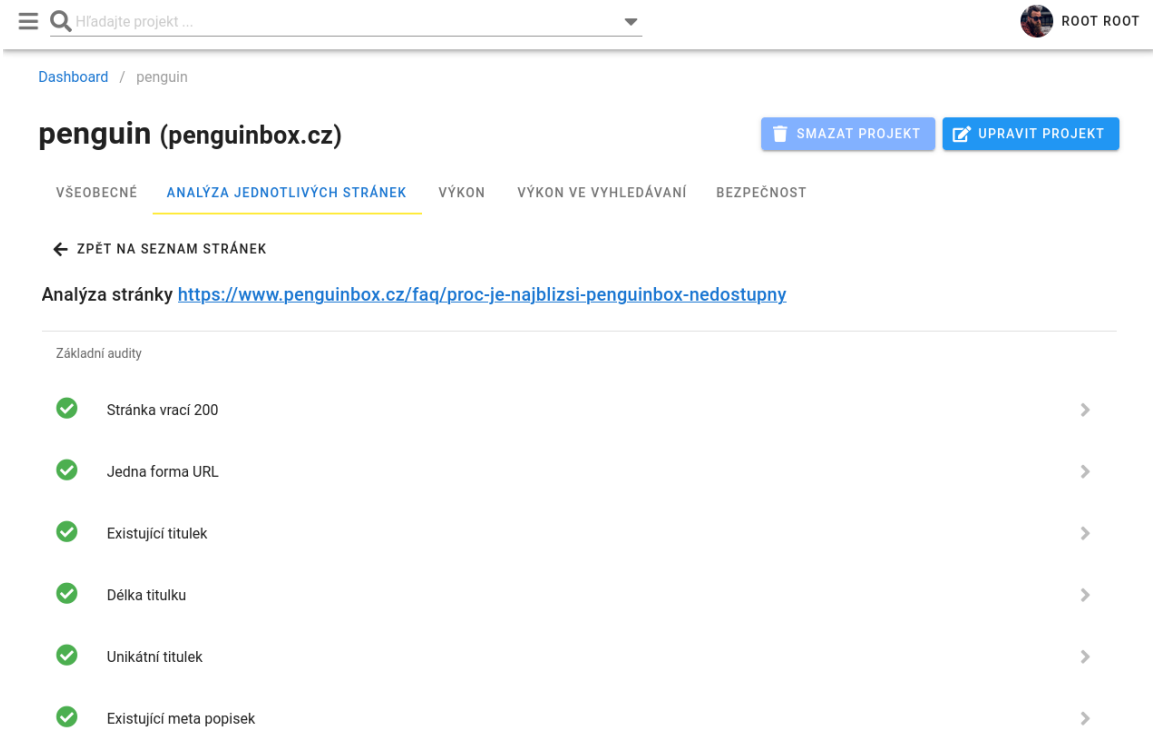
Obr. C.4: Snímka upravovania projektu.



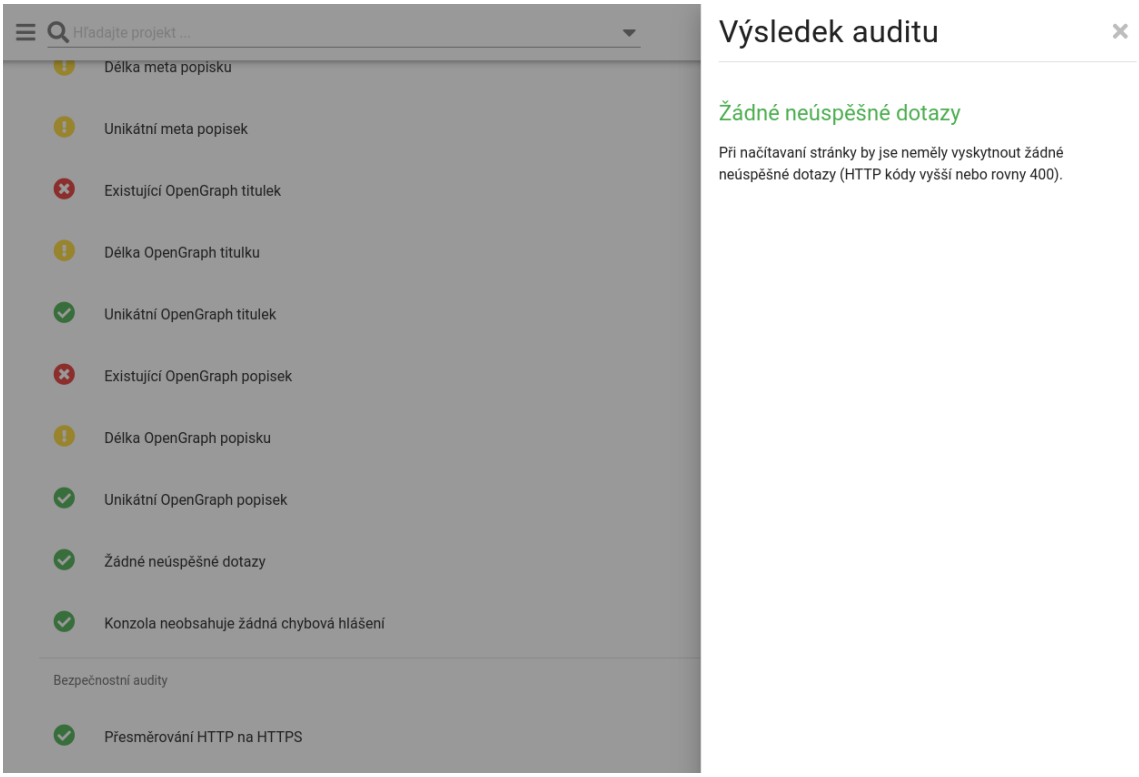
Obr. C.5: Snímka mazania projektu.



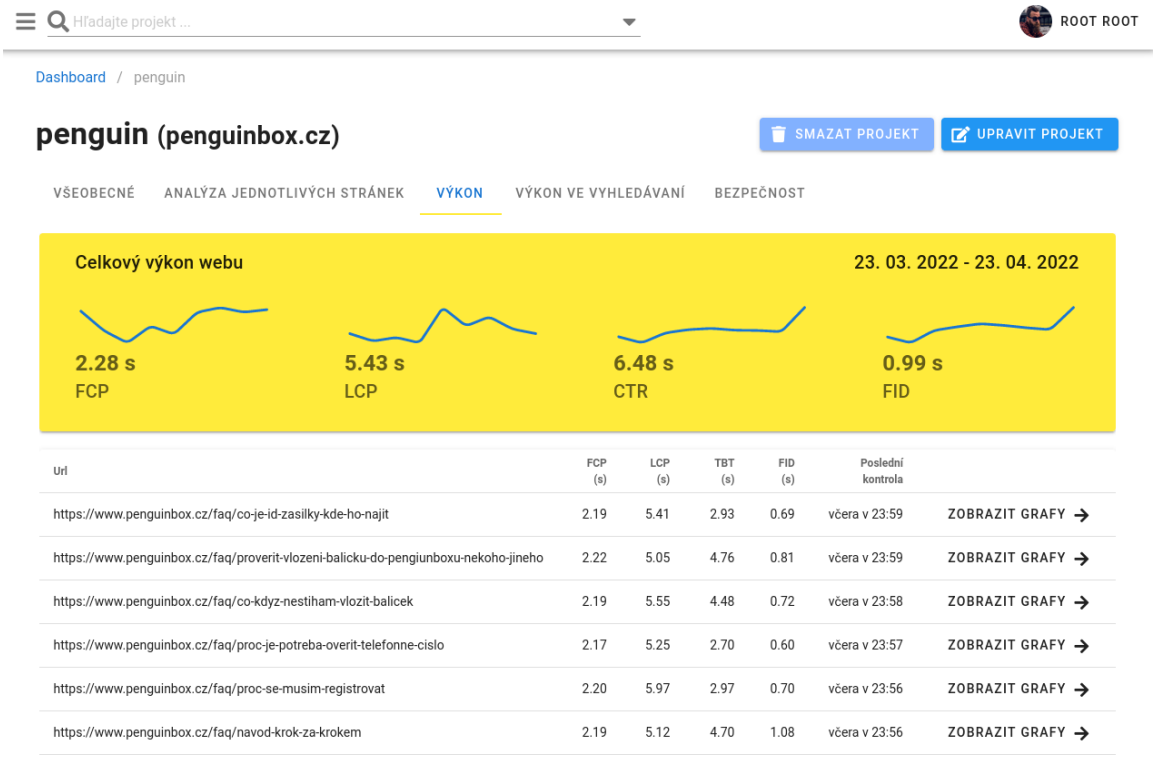
Obr. C.6: Snímka zoznamu stránok projektu.



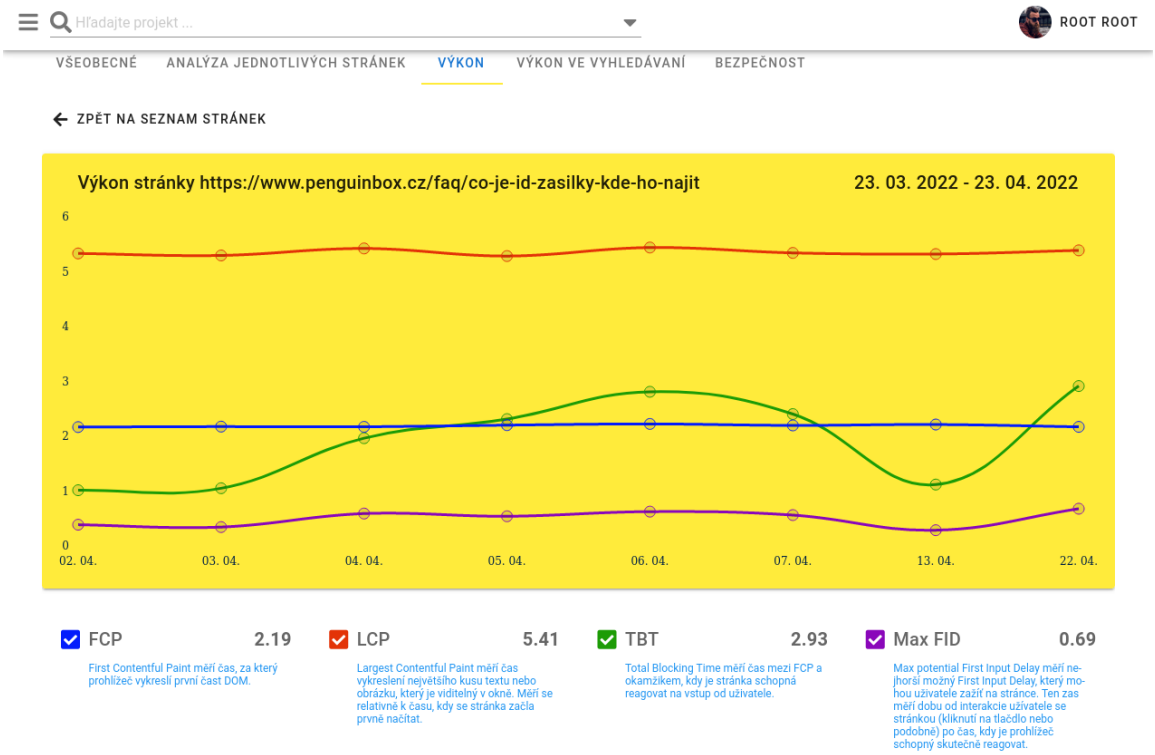
Obr. C.7: Snímka detailu stránky.



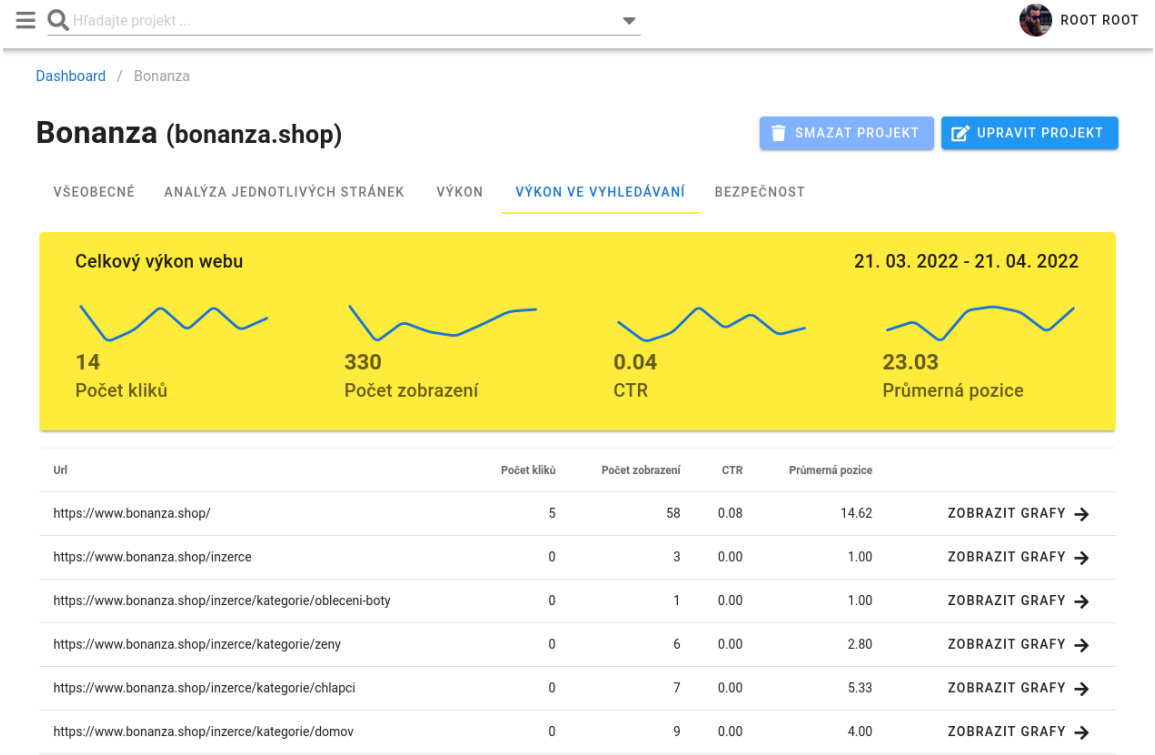
Obr. C.8: Snímka výsledku auditu.



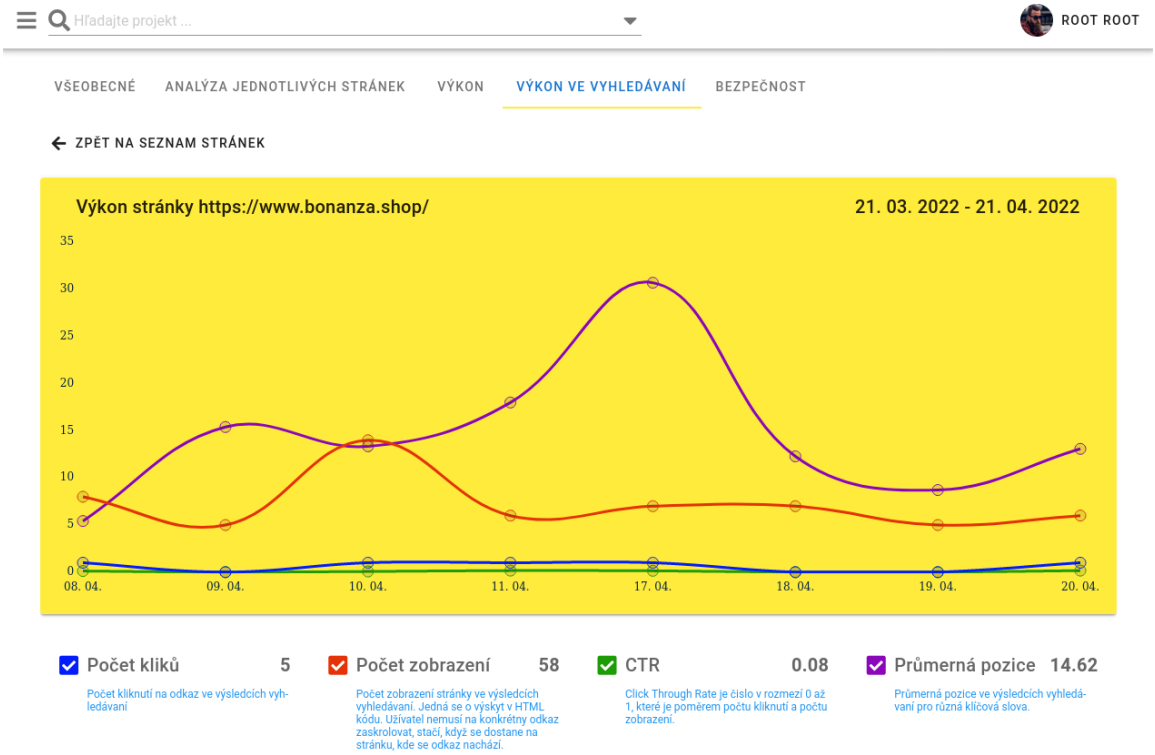
Obr. C.9: Snímka výkonu projektu.



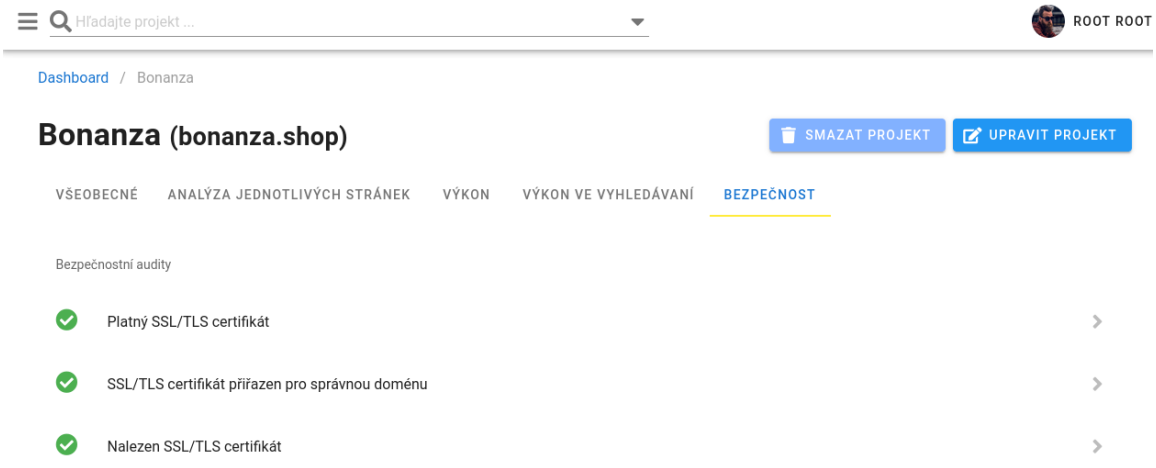
Obr. C.10: Snímka výkonu stránky.



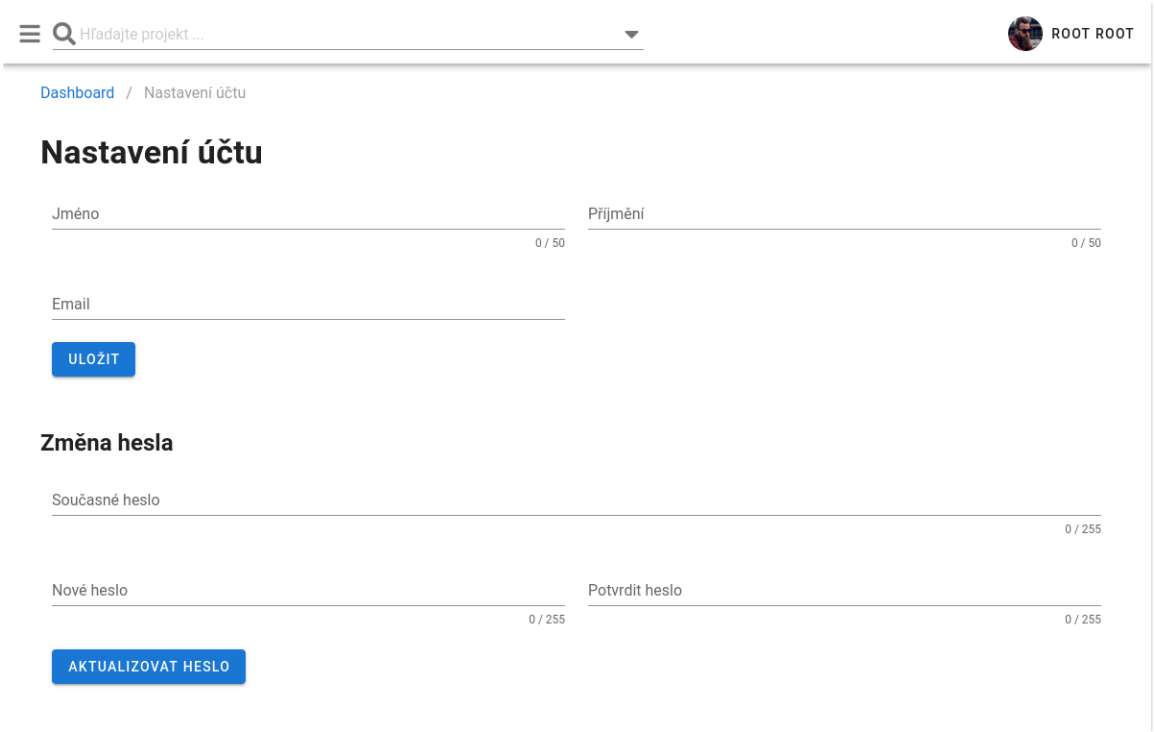
Obr. C.11: Snímka výkonu vo výsledkoch vyhľadávania projektu.



Obr. C.12: Snímka výkonu vo výsledkoch vyhľadávania stránky.



Obr. C.13: Snímka záložky s výsledkami bezpečnostných auditov.



Obr. C.14: Snímka nastavenia účtu.