Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Analýza front-end aplikací v závislosti na výkonu webu

Oleksandr Shaposhnykov

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Oleksandr Shaposhnykov

Informatika

Název práce

Analýza front-end aplikací v závislosti na výkonu webu

Název anglicky

Analysis of front-end applications depending on web performance

---

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku front-end aplikací. Hlavním cílem práce je najít efektivní řešení problému s výkonem webových aplikací a analyzovat metody, které pomohou dosáhnout maximálního výkonu při načítání a interakci s webovou aplikací.
Dílčí cíle práce jsou:
- vypracování přehledu a charakteristik front-end technologií,
- charakterizovat First meaningful paint(FMP), First input delay(FID), Time to interactive(TTI),
- navrh a provédení experimentálního měření výkonu načítání vytvořené testovací stránky.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní práce spočívá v nalezení efektivního řešení problému s výkonem front-end webových aplikací pomocí vytvoření webové aplikace pro porovnání a použití metod a technik, které zlepšují výkon. Mezi využité techniky bude patřit tvorba samotné webové aplikace a využití nástrojů pro měření výkonu webových stránek, jako např: Google PageSpeed Insights, Site24x7 a WebPage Test. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

40 – 50 stran textu.

Klíčová slova

Web, front-end, výkon, aplikace, javascript.

Doporučené zdroje informací

DAVIDMILLS, Chris a Andrew PFEIFFER, ed. Performance fundamentals: What is performance?. Performance fundamentals. MDN Web Docs, 2023, 2023, 1. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/Performance/Fundamentals

JANOVSKÝ, Dušan. Jak psát web: O tvorbě internetových stránek. Jak psát web. Jakpsatweb, 2021, 2021, 1. ISSN 1801-0458. Dostupné také z: https://www.jakpsatweb.cz

MICHÁLEK, Martin. CSS: Optimalizace datové velikosti: První zobrazení stránky a jeho ideální hodnota. CSS: Optimalizace datové velikosti. Vzhurudolu, 2019, 2019, 1. Dostupné také z: https://www.vzhurudolu.cz/prirucka/css-optimalizace

MONTTI, Roger. First Input Delay – A Simple Explanation. First Input Delay – A Simple Explanation. 2021, 2021, 1. Dostupné také z: https://www.searchenginejournal.com/core-web-vitals/first-input-delay/#close

RAI, Bikash. First Contentful Paint (FCP) and First Meaningful Paint (FMP) Explained. First Contentful Paint (FCP) and First Meaningful Paint (FMP) Explained. 2020, 2020, 1. Dostupné také z: https://www.acmethemes.com/blog/first-contentful-paint-and-first-meaningful-paint/

RUIKAR, Onkar, ed. JavaScript performance: Performance issues caused by scripts. JavaScript performance. MDN Web Docs, 2023, 2023, 1. Dostupné také z: https://developer.mozilla.org/en-US/docs/Learn/Performance/JavaScript

SHIOTSU, Yoshitaka. JavaScript Optimization Tips: JavaScript is a truly amazing tool for front-end programming. JavaScript Optimization Tips. 2021, 2021, 1. Dostupné také z: https://www.upwork.com/resources/javascript-optimization-tips

SMIČKA, Radim. Optimalizace pro vyhledávače – SEO: Jak zvýšit návštěvnost webu. Optimalizace pro vyhledávače – SEO. Knihkupectví Jasmínka, Dubany, 1. Dostupné také z: https://i.iinfo.cz/files/root/k/SEO-Jak_zvysit_navstevnost_webu.pdf

VANČURA, Tadeáš. Jak fungují webové stránky: Principiální pohled na to, jak weby fungují. Jak fungují webové stránky. ExpressInfo, 2020, 2020, 1. Dostupné také z: https://www.expressinfo.cz/technika/uspesny-web/jak-funguji-webove-stranky-html-css-javascript-i-php/4171/

WALTON, Philip. Time to Interactive (TTI). Time to Interactive (TTI). 2022, 2022, 1. Dostupné také z: https://web.dev/tti/

Předběžný termín obhajoby
2023/24 LS – PEF

Vedoucí práce
doc. Ing. Pavel Šimek, Ph.D.

Garantující pracoviště
Katedra informačních technologií

Elektronicky schváleno dne 4. 7. 2023

doc. Ing. Jiří Vaněk, Ph.D.
Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.
Děkan

V Praze dne 01. 03. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Analýza front-end aplikací v závislosti na výkonu webu" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2024 _____

Poděkování

Rád bych touto cestou poděkoval panu doc. Ing. Pavlu Šimkovi, Ph.D. za vedení moje práce. Ještě bych chtěl poděkovat také své rodině za podporu během napsání této práci.

Analýza front-end aplikací v závislosti na výkonu webu

Abstrakt

Tato bakalářská práce se zabývá hlavními aspekty ovlivňujícími tvorbu webových stránek a jejich výkonu. S využitím znalostí programování webových stránek je vytvořena webová aplikace s cílem optimalizace pomocí různých technik a metod. Jedním z rysů přístupu k práci je vyhýbání se používání dalších programů a frameworků. Provedeným výzkumem bylo zjištěno, jak optimalizace kódu a jeho zápis ovlivňují výkon stránky. Hlavním zjištěním bylo najít nejlepší způsob zápisu kódu stránky, aby se co nejvíce urychlilo načítání webové aplikace a bylo dosaženo vysoké rychlosti zobrazování prvků na stránce.

Klíčová slova: web, programovaní, html, css, javascript, výkon, aplikace, analýza, SEO, optimalizace, FCP, FID, TTI.

Analysis of front-end applications depending on web performance

Abstract

This bachelor's thesis deals with the main aspects influencing website design and performance. Using the knowledge of web programming, a web application is created with the aim of optimization using various techniques and methods. One of the features of the approach is avoiding the use of other programs and frameworks. Through the research conducted, it was found how code optimization and writing affect the performance of the website. The main finding was to find the best way of writing the page code to speed up the loading of the web application as much as possible and to achieve high speed display of the elements on the page.
Keywords: web, programming, html, css, javascript, performance, application, analyse, SEO, optimization, FCP, FID, TTI.

Obsah

1 Úvod

Práce se zabývá hlavními aspekty, které ovlivňují design a výkon webových stránek, a identifikuje charakteristiky, jako je první významná barva (FMP), první vstupní zpoždění (FID) a čas do interaktivity (TTI), které hrají důležitou roli při vnímání načítané stránky uživatelem. Použití front-end technologií HTML, CSS a JavaScript jako základních nástrojů pro tvorbu webových stránek. Proces optimalizace webových stránek zahrnuje také důkladné testování a analýzu výsledků. Pro účely testování je použity různé webové programy, které změří rychlost načítání webové stránky.

Zohlednění dopadů různých technik a metod optimalizace webových stránek, jako je minimalizace požadavků HTTP, komprese souborů, asynchronní načítání zdrojů, ukládání do mezipaměti, výběr optimálních barevných schémat, optimalizace obrázků, volba vhodných písem a další přístupy, které přispívají k rychlejšímu načítání a lepšímu výkonu webové aplikace. Důležitým aspektem tohoto výzkumu je studium vlivu různých front-end technologií na výkon webových stránek. Srovnávací analýza různých přístupů umožňuje určit nejúčinnější nástroje pro vytváření vysoce výkonných webových aplikací.

Je třeba také zdůraznit důležitost přizpůsobení mobilním zařízením a responzivního designu, protože mobilní zařízení se pro mnoho uživatelů stala hlavním prostředkem přístupu k internetu. Vytvoření webové stránky s ohledem na mobilní platformy zlepší uživatelskou zkušenost na různých zařízeních a sníží míru neúspěchu při načítání webové aplikace.

Jedním z klíčových rysů tuto práci je nepoužití žádných dalších softwarů a frameworků, což umožní přesnější výzkum vlivu optimalizace a technik psaní "čistého" kódu na výkonnost stránky. Je důležité vzít v úvahu, že moderní webové technologie jsou neustále vyvíjí, což umožňuje využít nové výhody k dosažení vytyčených cílů.

2 Cíl práce a metodika

Bakalářská práce je tematicky zaměřena na problematiku front-end aplikací. Hlavním cílem práce je najít efektivní řešení problému s výkonem webových aplikací a analyzovat metody, které pomohou dosáhnout maximálního výkonu při načítání a interakci s webovou aplikací.

Dílčí cíle práce jsou:

- vypracování přehledu a charakteristik front-end technologií,
- charakterizovat First meaningful paint(FMP), First input delay(FID), Time to interactive(TTI),
- navrh a provédení experimentálního měření výkonu načítání vytvořené testovací stránky.

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní práce spočívá v nalezení efektivního řešení problému s výkonem front-end webových aplikací pomocí vytvoření webové aplikace pro porovnání a použití metod a technik, které zlepšují výkon. Analýza literatury a přehled současného vývoje v oblasti optimalizace výkonu webových stránek. Zpracování aktuálních přístupů a technik optimalizace a jejich použití v různých webových projektech. Budou použity techniky, jako je optimalizace struktury kódu pro zlepšení výkonu prohlížeče, přidání metaznaček, media dotazů, Gulp a Webpack pro optimalizaci souborů webové aplikace, komprese souborů css a přidání funkcí Javascriptu. Pro získání reprezentativnějších výsledků budou provedeny experimenty na různých zařízeních. Definovano a popsano výkonnostní kritéria, jako jsou FMP, FID a TTI, a jejich dopad na uživatelskou zkušenost. To umožní lépe pochopit, které aspekty výkonnosti by měly být v rámci výzkumu zlepšeny. Mezi využité techniky bude patřit tvorba samotné webové aplikace a využití nástrojů pro měření výkonu webových stránek, jako např: Google PageSpeed Insights, Site24x7 a WebPage Test. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

3 TEORETICKÁ VÝCHODISKA

3.1 Přehled a charakteristiky front-end technologií

3.1.1 Fungování webových prohlížečů

Uživatelé chtějí aplikace, které rychle načítají obsah a hladce fungují. Vývojář by se měl porozumět tomu, jak fungují prohlížeče a snažit optimalizovat svou aplikaci alespoň pro tyto dva ukazatele.

- zlepšit výkon aplikace
- zlepšit uživatelsky vnímaný výkon (UPP)

Rychlé aplikace poskytují lepší zážitek. Uživatelé očekávají, že se aplikace bude rychle načítat a že s ní budou plynule pracovat. Dva hlavní problémy s výkonem jsou problém rychlosti sítě a problém jednovláknových prohlížečů.

Problém rychlého načítání. Aby vývojář urychlil načítání, musí požadovaná data odeslat co nejrychleji nebo alespoň předstírat, že jsou odesílána velmi rychle. Většina prohlížečů je považována za jednovláknové aplikace. Aby bylo dosaženo plynulé interakce, musí vývojář zajistit výkon ve všech oblastech, od plynulého posouvání až po rychlou odezvu na klepnutí na obrazovku.

Klíčovým pojmem je doba vykreslování. Vývojář musí zajistit, aby aplikace běžela tak, aby všechny její úkoly mohly být dokončeny dostatečně rychle. Procesor pak bude mít volný prostor pro zpracování uživatelských vstupů. K vyřešení problému jednovláknovosti je třeba pochopit podstatu prohlížečů a naučit se odlehčit hlavní tok procesu tam, kde je to možné a přijatelné. [1][2][3]

3.1.2 Navigace

Navigace je prvním krokem při načítání aplikace. Probíhá pokaždé, když uživatel požádá o stránku zadáním adresy URL do adresního řádku prohlížeče, kliknutím na odkaz, odesláním vyplněných polí formuláře a provedením některých dalších akcí.

Jedním z cílů vývojáře je zkrátit dobu, kterou aplikace potřebuje k dokončení fáze navigace. Za ideálních podmínek to obvykle netrvá dlouho, ale latence sítě a šířka kanálu jsou překážky, které způsobují zpoždění při načítání aplikace. [1][3]

3.1.3 DNS Dotazování

Prohlížeč si vyžádá záznam DNS. Dotaz obvykle obsahuje název serveru, který má být přeložen na IP adresu. Odpověď na tento dotaz se na chvíli uloží do mezipaměti zařízení, aby ji bylo možné rychle načíst při příštím dotazu na stejný server.

Dotaz DNS je obvykle třeba provést pouze jednou při načítání stránky. Dotazy DNS (obr.1) však musí být provedeny pro každý jedinečný název hostitele, který je stránkou požadován. Pokud se například písma, obrázky, skripty, reklamy nebo analytická počítadla nacházejí na různých doménách, bude dotaz DNS proveden pro každou z nich. [1]

Obrázek 1. Prohlížeč požaduje záznam DNS. Zdroj: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

To může představovat výkonnostní problém, zejména pro mobilní sítě. Když je uživatel v mobilní síti, musí každý dotaz DNS putovat z mobilního zařízení do mobilní věže a odtud na autoritativní server DNS. Vzdálenost a rušení mezi telefonem, věží a jmenným serverem mohou výrazně zvýšit latenci. [1][2][10]

3.1.4 Reakce na žádost

Po navázání spojení s webovým serverem odešle prohlížeč jménem uživatele iniciační požadavek HTTP GET. Nejčastěji se jedná o požadavek na soubor HTML (obr.2). V okamžiku, kdy server obdrží požadavek, zahájí odpověď odesláním hlaviček odpovědi a obsahu souboru HTML. [1][2][4][10]

HTML

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>My simple page</title>
    <link rel="stylesheet" src="styles.css" />
    <script src="myscript.js"></script>
  </head>
  <body>
    <h1 class="heading">My Page</h1>
    <p>A paragraph with a <a href="https://example.com/about">link</a></p>
    <div>
      <img src="myimage.jpg" alt="image description" />
    </div>
    <script src="anotherscript.js"></script>
  </body>
</html>
```

Obrázek 2. Příklad obsahu soubora HTML. Zdroj: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

Tato odpověď obsahuje první bajt přijatých dat. Čas mezi okamžikem, kdy uživatel odeslal požadavek kliknutím na odkaz, a okamžikem přijetí prvního datového paketu HTML. První paket obvykle obsahuje 14 KB dat. V uvedeném příkladu (obr.2) je odpověď mnohem menší než 14KB; skripty a styly uvedené v odpovědi budou vyžádány až po zpracování odpovědi prohlížečem.[3][4][11]

3.1.5 TCP Handshake, TLS Negotiation, TCP slow start / pravidlo 14kb

V okamžiku, kdy je známa IP adresa, začne prohlížeč navazovat spojení se serverem pomocí třícestného handshake TCP. Tento mechanismus je navržen tak, aby si dvě zařízení, která se pokoušejí navázat spojení, mohla vyměnit parametry spojení předtím, než přistoupí k přenosu dat. Nejčastěji prostřednictvím zabezpečeného připojení HTTPS.

Tříkrokový handshake TCP je technika velmi často označovaná jako **"SYN-SYN-ACK"** (přesněji **SYN, SYN-ACK, ACK**), protože při navazování spojení jsou přenášeny 3 zprávy. To znamená, že prohlížeč si musí se serverem vyměnit další tři zprávy, než dojde k navázání spojení.[1][4]

Navázání zabezpečených připojení pomocí protokolu HTTPS vyžaduje další handshake. Tentokrát se jedná o TLS negotiation (vyjednávání TLS). Tento krok určuje, jaká šifra bude použita k šifrování spojení, ověřuje důvěryhodnost serveru a navazuje bezpečné spojení. Tento krok také vyžaduje výměnu několika dalších zpráv mezi serverem a prohlížečem před odesláním dat. Zabezpečení spojení sice snižuje rychlost načítání aplikace, ale bezpečné spojení se vyplatí, protože data pak nemůže dešifrovat třetí strana. Po výměně osmi zpráv prohlížeč konečně dosáhne všech podmínek pro provedení požadavku.

Velikost prvního datového paketu je vždy 14 KB. Jedná se o součást specifikace TCP slow start, algoritmu, který vyrovnává rychlost připojení. Toto pravidlo umožňuje postupně podle potřeby zvětšovat velikost přenášených dat, dokud není určena maximální šířka kanálu. V algoritmu pomalého startu TCP se velikost každého dalšího paketu odeslaného serverem zdvojnásobí. Například velikost druhého paketu bude přibližně 28 KB. Velikost paketů se bude zvyšovat, dokud nedosáhne určité prahové hodnoty nebo problému přetečení.[1][4]

Optimalizace výkonu spouštění musí brát v úvahu omezení tohoto počátečního požadavku. Pomalý start TCP umožňuje plynulé zrychlení přenosu dat tak, aby nedocházelo k problému přetečení, kdy na odeslání čeká velké množství dat, která však nejsou odeslána kvůli omezení šířky kanálu. [1]

3.1.6 Parsing

Během parsování jsou přijatá data převedena na DOM a CSSOM, které se přímo podílejí na vykreslování.

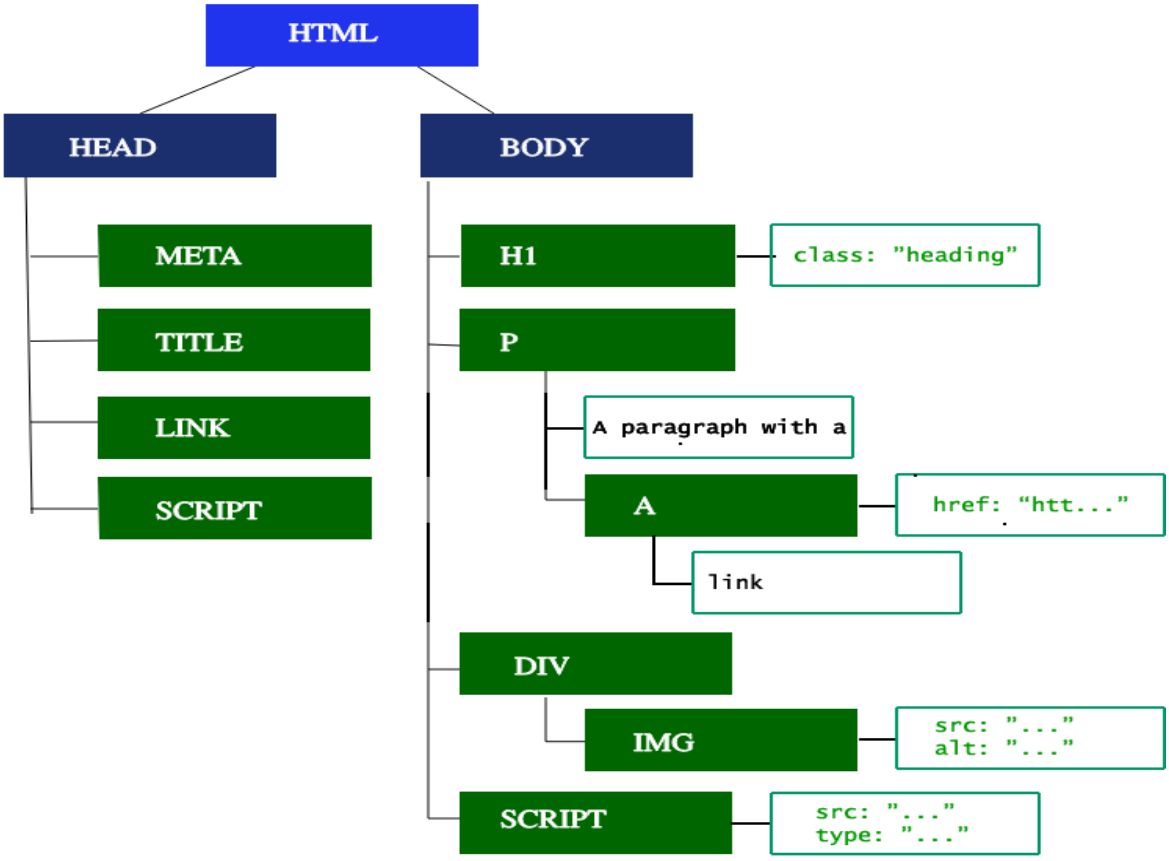
DOM (Document Object Model) je vnitřní reprezentace značek HTML. Prohlížeč poskytuje přístup k manipulaci s objekty tohoto modelu prostřednictvím různých rozhraní API jazyka JavaScript. [1][3][5]

I když je odpověď na požadavek větší než 14 kB, prohlížeč přesto začne analyzovat data a pokusí se vykreslit stránku s daty, která jsou již k dispozici. Proto je důležité zahrnout všechna data potřebná pro vykreslení do odpovědi o velikosti 14 KB, aby prohlížeč mohl začít vykreslovat stránku rychleji. Než se však cokoli zobrazí na obrazovce, musí se zpracovat HTML, CSS a JavaScript.[2][3]

3.1.7 Vytváření stromu objektového modelu dokumentu

Prvním krokem je zpracování značkovacího jazyka HTML a vytvoření stromu DOM. Zpracování HTML zahrnuje tokenizaci a sestavení stromu. Tokeny HTML se skládají z počátečních a koncových tagů a atributů. Pokud je dokument správně vytvořen, je jeho zpracování jednoduché a rychlé. Parser (zpracovatel) převede příchozí tokeny na dokument a sestaví strom dokumentu.[2]

Objektový model dokumentu (DOM) popisuje obsah dokumentu. Element <html> je první tag a kořenový element stromu dokumentu. Strom (obr.3) odráží vztahy a hierarchii mezi jednotlivými tagy. Tagy vnořené do jiných tagů jsou potomky. Čím více uzlů je ve stromu, tím obtížnější je strom sestavit. [1]



Obrázek 3. Struktura stromu objektového modelu dokumentu. Zdroj: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

Když parser najde neblokující zdroje (např. obrázky), prohlížeč odešle požadavek na načtení zdrojů, ale pokračuje ve zpracování. Zpracování může pokračovat, pokud je zjištěn odkaz na soubor CSS, ale pokud je zjištěn <script>, zejména pokud nemá parametry async nebo defer, je skript považován za blokující a pozastaví se zpracování HTML, dokud se skript nedokončí načítat.

Navzdory tomu, že skener předběžného načtení prohlížeče dokáže takové skripty vyhledat a vyžádat si je předem, mohou složité a objemné skripty stále způsobovat znatelné zpoždění načítání stránky. [1][4]

3.1.8 Předstartovní skener

Sestavení stromu DOM zabere celý tok procesu. Protože se jedná o zjevně výkonnostně slabé místo, byl vytvořen speciální skener pro předběžné načítání. Ten zpracovává dostupný obsah dokumentu a požaduje prostředky s vysokou prioritou (CSS, JavaScript a fonty). Díky tomuto skeneru se nemusí čekat, až se parser dostane na konkrétní místo, kde je prostředek volán. Vyžádá si a obdrží tato data předem, na pozadí, takže když hlavní vlákno parseru HTML dojde k požadavku na zdroj, je vysoce pravděpodobné, že zdroj již byl vyžádán nebo se právě načítá. Optimalizace dané tímto skenerem zkracují dobu blokování renderu stranky.[4]

```HTML
<link rel="stylesheet" src="styles.css" />
<script src="myscript.js" async></script>
<img src="myimage.jpg" alt="image description" />
<script src="anotherscript.js" async></script>
```

Obrázek 4. Zpracování dostupného obsahu dokumentů a vyžádání zdrojů s vysokou prioritou.. Zdroj: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

Ve výše uvedeném příkladu (obr.4) hlavní vlákno zpracovává HTML a CSS. Současně skener před načtením najde skripty a obrázek a začne je načítat.

Čekání na načtení CSS neblokuje parsování HTML, ale blokuje JavaScript, protože JavaScript se často používá k vzorkování uzlů dokumentu pomocí selektorů CSS. [2]

3.1.9 Vytvoření modelu stylů CSSOM

Druhým krokem při průchodu kritickou cestou renderování je zpracování CSS a vytvoření stromu CSSOM. CSSOM (objektový model CSS) je podobný DOM. DOM i CSSOM jsou stromy. Jsou to nezávislé datové struktury. Prohlížeč převádí soubory CSS na mapu stylů se kterou může pracovat. Prohlížeč načte každou sadu pravidel CSS a na základě selektorů CSS vytvoří strom uzlů s rodiči, dětmi a sousedy.

Stejně jako v případě jazyka HTML musí prohlížeč výsledná pravidla CSS převést na něco, s čím může pracovat. Celý tento proces je tedy opakováním tvarování DOM, pouze pro CSS. [3][4]

Strom CSSOM obsahuje styly uživatelského agenta - to jsou styly, které prohlížeč vkládá ve výchozím nastavení. Prohlížeč začíná sestavovat model s nejobecnějšími pravidly pro každý uzel a postupně aplikuje specifičtější pravidla. Jinými slovy, pravidla aplikuje kaskádovým způsobem. Odtud název CSS - kaskádové styly. [1][2]

3.2 Doplňkové procesy načítání webové aplikace

3.2.1 Kompilace JavaScriptu, Sestavení stromu přístupnosti

Při zpracování CSS a vytváření CSSOM pokračuje načítání dalších zdrojů, například souborů JavaScript (díky skeneru před načtením). Po dokončení načítání je třeba JavaScript interpretovat, zkompilovat, zpracovat a spustit. Skripty jsou převedeny do abstraktního syntaktického stromu (AST). Některé prohlížeče převezmou abstraktní syntaktický strom a předají jej interpretu, který strom převede na bajtový kód. Bytový kód se provádí v hlavním vlákně. Celý tento proces se nazývá kompilace.

Prohlížeč také vytváří strom přístupnosti, který používají asistenční zařízení k pochopení a interpretaci obsahu. Model objektů přístupnosti (AOM) je sémantická verze DOM. Prohlížeč aktualizuje AOM ve stejném okamžiku, kdy je aktualizován DOM. Strom přístupnosti zároveň nemůže být modifikován asistenčními technologií.[2][3]

3.2.2 Renderování, Styling, Rozvržení, Malování, Kompozice

Fáze renderování zahrnují stylizaci, rozvržení, malbu a v některých případech i kompozici. Stromy CSSOM a DOM vytvořené v předchozím kroku se spojí do renderovacího stromu, který se poté použije k výpočtu pozice každého viditelného prvku. Poté se prvky vykreslí na obrazovku. V některých případech může být obsah renderován v samostatných vrstvách a kombinován (kompozice) - tento přístup zvyšuje výkon tím, že umožňuje renderování obsahu obrazovky na grafickém procesoru místo CPU. Tím se uvolní hlavní tok.

Třetím krokem v kritické cestě renderování je spojení DOM a CSSOM do vykreslovacího stromu. Konstrukce tohoto stromu začíná procházením celého stromu DOM od kořene a identifikací každého viditelného uzlu.[4]

Prvky, které by neměly být vykresleny, jako je <head>, stejně jako jeho potomci nebo jakékoli prvky s display:none, jako je script { display: none; }, nebudou do renderovacího stromu zahrnuty, protože by neměly být renderovány. Uzly s pravidlem visibility: hidden jsou do renderovacího stromu zahrnuty, protože stejně zabírají jejich místo.

Protože nejsou zadána žádná zvláštní pravidla, která by přepisovala výchozí styly agenta, nebude uzel „script“ ve výše uvedeném příkladu rovněž zahrnut do vykreslovacího stromu.

Každý viditelný uzel má svá vlastní pravidla z CSSOM. Strom renderování obsahuje všechny viditelné uzly s jejich obsahem a vypočtenými styly. Styly jsou určeny použitím všech vhodných pravidel pomocí kaskády CSS.[3]

Čtvrtým krokem v kritické cestě vykreslování je zahájení rozvržení prvků vykreslovacího stromu. V tomto kroku se vypočítá geometrie každého uzlu, tj. šířka, výška a poloha prvků. Přepracování je jakýkoli následný proces určení velikosti a polohy pro některou z částí celého dokumentu.

Jakmile je strom vykreslování sestaven, začíná rozvržení. Strom nese informace o tom, které uzly mají být vykresleny (i když jsou neviditelné) a jaké styly mají být použity, ale strom nenese žádné informace o velikosti a pozici prvků. Pro zjištění těchto hodnot začne prohlížeč procházet strom. [1]

Na webové stránce jsou téměř všechny prvky obdélníkové (box). Různá zařízení a nastavení znamenají nespočet různých velikostí viditelné oblasti. V počáteční fázi prohlížeč vzhledem k velikosti viditelné oblasti určí, jaké velikosti mají mít jednotlivé prvky na obrazovce. Na základě velikosti viditelné oblasti jako výchozího bodu začne proces výpočtu s prvkem těla, poté přejde k jeho potomkům, vypočítá velikost každého prvku a vyhradí místo pro prvky, jejichž velikost ještě nezná (například obrázky).

Okamžik, kdy se vypočítávají pozice a velikosti uzlů, se nazývá rozvržení. Následné výpočty pozic a velikostí se nazývají reflow. V tomto příkladu dochází k předpokládanému počátečnímu rozložení před pořízením obrázku. Protože není zadána žádná velikost obrazu, dojde k reflow v okamžiku pořízení obrazu.

Posledním krokem kritické cesty renderování je renderování jednotlivých uzlů na obrazovce. Okamžik, kdy k tomu dojde poprvé, se nazývá First Meaningful Paint (smysluplné malování). Během fáze vykreslování neboli rastrování prohlížeč převede každý kontejner políčka na skutečné pixely na obrazovce. Rastrování zahrnuje vykreslování každé vizuální částečky prvku na obrazovce (text, barvy, okraje, stíny) a vykreslování vyměnitelných prvků (obrázky, tlačítka).[2][4][5]

Aby bylo zajištěno plynulé posouvání a animace, zabere vykreslování každého prvku celé hlavní vlákno. To zahrnuje výpočet stylů, přepočítávání stylů a vykreslování. Všechny tyto kroky by neměly trvat déle než 16,67 ms (1000 ms / 60 snímků za sekundu).

Aby se iniciace a překreslování zrychlily, je třeba celý proces rozdělit do více vrstev. Když se tak stane - je nutná kompozice. [5]

Vykreslování může rozdělit prvky ve stromu renderování do vrstev. Aby se renderování zrychlilo, může prohlížeč přenést renderování jednotlivých vrstev na grafický procesor (namísto hlavního CPU vlákna). K přenosu vykreslovacích výpočtů na GPU lze použít některé speciální tagy HTML, například <video> a <canvas>; a vlastnosti CSS: opacity, transform a will-change. Takto vytvořené uzly budou vykresleny ve své vlastní vrstvě spolu se svými potomky, pokud nejsou potomci sami vykresleni v samostatných vrstvách.

Pokud jsou části dokumentu renderovány v různých vrstvách a jedna vrstva se nachází nad jinou nebo ji překrývá, je nutné použít kompozici. Tento krok umožňuje prohlížeči zajistit, aby byly jednotlivé vrstvy na obrazovce zobrazovány ve správném pořadí a aby byl obsah zobrazen správně.

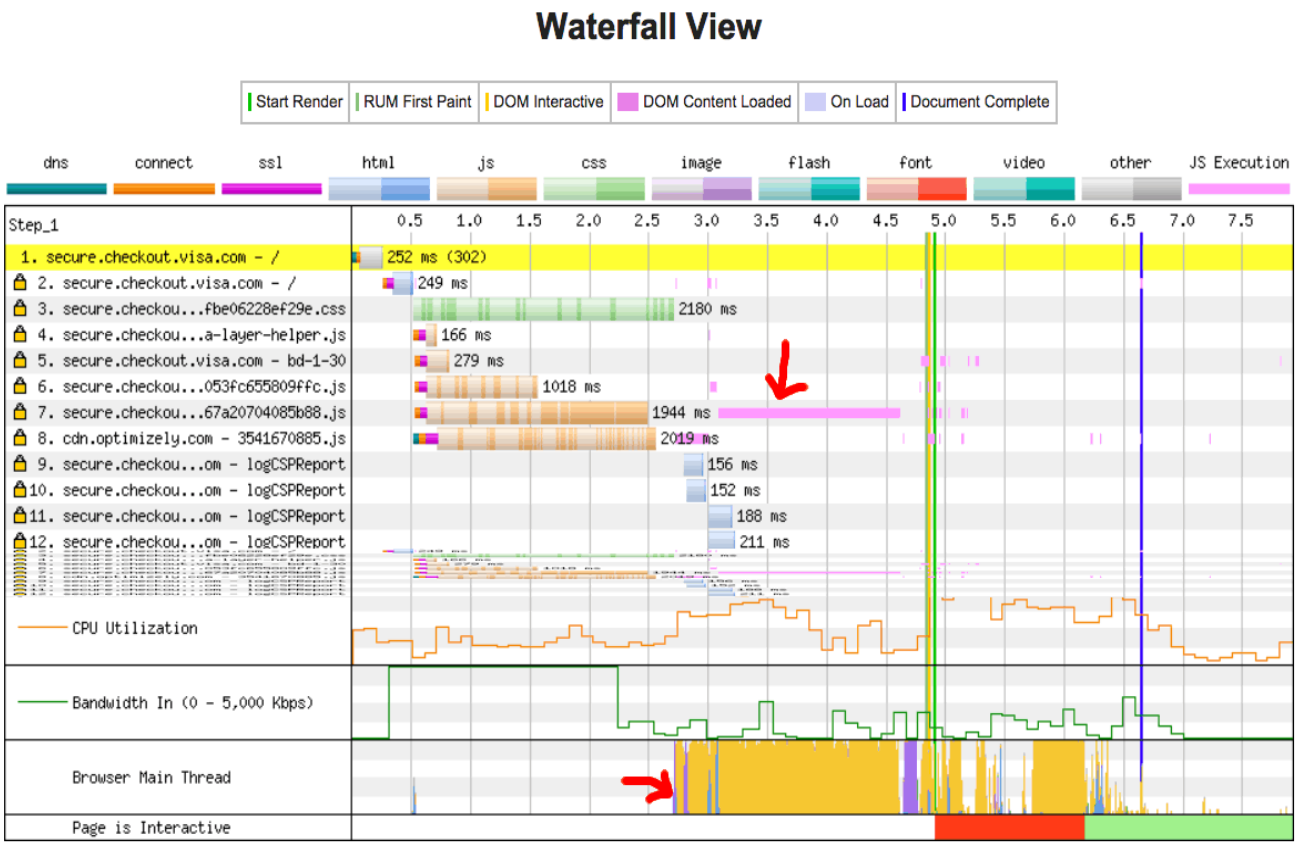
Při načítání dříve vyžádaných zdrojů (například obrázků) může být nutné přepočítat velikost a polohu prvků vůči sobě navzájem. Tento přepočet - reflow - vyvolá překreslení a opětovné složení. Pokud je předem definovana velikost obrázku, přepočet nebude nutný a v takovém případě se překreslí pouze vrstva, která má být překreslena. Pokud je však velikost obrázku předem nedefinovana, pak bude prohlížeč po obdržení odpovědi ze serveru nucen přetočit proces vykreslování zpět do kroku rozvržení a začít vykreslovat znovu. [1]

3.2.3 Interaktivita na webové stránce

Člověk by si myslel, že jakmile hlavní vlákno dokončí vykreslování stránky, "je hotovo". Ne vždy je tomu tak. Pokud se mezi načtenými zdroji nachází JavaScript, jehož načítání bylo správně odloženo a který se spustí až po události onload, hlavní vlákno zahájí zpracování skriptu. Během tohoto zpracování prohlížeč nemůže zpracovávat události rolování, kliknutí atd.[6]

Doba do interaktivity (TTI) je měřítkem toho, kolik času uplyne mezi prvním požadavkem sítě a okamžikem, kdy se stránka stane interaktivní. Na časové ose tato fáze následuje bezprostředně po First Meaningful Paint. Interaktivita je ukazatel toho, že stránka reagovala na akci uživatele v čase 50 ms. Pokud je procesor zatížen zpracováním, kompilací a prováděním JavaScriptu, prohlížeč nemůže reagovat dostatečně rychle, což znamená, že stránka je považována za neinteraktivní.[4][5][6]

V tomto příkladu se sice obrázek načetl rychle, ale načtení skriptu anotherscript.js, který má velikost až 2 MB, trvá dlouho. V tomto případě uživatel uvidí stránku velmi rychle, ale nebude s ní moci interagovat, dokud se skript nenačte, nezpracuje a neprovede. [1][11]



Obrázek 5. Načítání obsahu DOM.

Zdroj: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

Ve výše uvedeném příkladu (obr.5) trvalo načtení obsahu DOM přibližně 1,5 sekundy. Po celou tuto dobu byl hlavní tok procesu plně vytížen a nebyl schopen zpracovat vstup uživatele.

3.3 Základy výkonnosti

3.3.1 Co je to výkonnost?

Jediný výkon, na kterém záleží, je výkon vnímaný uživatelem. Uživatelé komunikují se systémem prostřednictvím nějakého druhu vstupu: dotykem, pohybem a řečí. Na základě vizuálního, hmatového nebo sluchového vstupu dostanou odpověď. Výkon je kvalita toho, jak systém uživateli odpovídá.[1]

Za stejných podmínek kód, který je optimalizován pro něco jiného než výkon vnímaný uživatelem (dále jen UPP, user-perceived performance), vždy prohraje kódu, který je optimalizován pro UPP. Jednoduše řečeno, uživatelé dávají přednost citlivé a plynulé aplikaci, která zpracovává 1 000 databázových transakcí za sekundu, před hrubě nereagující aplikací, která zpracovává 100 000 000 požadavků za sekundu. To samozřejmě neznamená, že ostatní metriky se stávají zbytečnými, ale prvním cílem by mělo být UPP.

HTML a CSS výrazně zvyšují výkon, někdy snižují počet snímků za sekundu a neumožňují kontrolovat každý pixel při vykreslování. Text a obrázky se překreslují automaticky, prvky uživatelského rozhraní automaticky získávají systémové téma a systém poskytuje "vestavěnou" podporu pro některé případy, které vývojáře možná zpočátku nenapadly. Například zobrazování obsahu v různých rozměrech.

Většina webových stránek používá více souborů CSS. Ačkoli je použití modulárních souborů CSS ve většině případů považováno za nejlepší řešení, načtení každého jednotlivého souboru může nějakou dobu trvat. Ale právě z tohoto důvodu jsou k dispozici nástroje pro minifikaci a kompresi CSS.

V případě souborů CSS je priorita obvykle vysoká, protože soubory stylů jsou potřebné k vytvoření objektového modelu CSSOM (CSS Object Model). Aby mohl prohlížeč vykreslit webovou stránku, musí vytvořit DOM i CSSOM. Bez toho prohlížeč nezobrazí pixely na obrazovce. Důvodem je to, že styly definují vzhled stránky a vykreslení stránky bez nich by znamenalo plýtvání výpočetním výkonem a mělo by za následek špatné uživatelské rozhraní. Teprve když má prohlížeč k dispozici DOM i CSSOM, může jejich kombinací vytvořit vykreslovací strom a začít vykreslovat obrazovku. Stručně řečeno, není načteno CSS, není vykreslena stránka.

CSS má obrovský vliv na dobu načítání webové stránky. Velikost souboru CSS a celkové množství CSS na stránce (počet souborů). Příliš velké soubory CSS se budou načítat déle, a proto bude vykreslování celé stránky trvat mnohem déle.[2]

3.3.2 Responzivita

Rychlost odezvy znamená, jak rychle systém nabídne odpověď (nebo více odpovědí) na požadavek uživatele. Například když uživatel klepne na obrazovku, očekává, že se pixely pod jeho prstem nějakým způsobem změní. Pro tento případ interakce by byla dobrou metrikou doba, která uplynula mezi okamžikem stisknutí a změnou pixelů. Reakce někdy zahrnuje více kroků. Spuštění aplikace je jednou z nejdůležitějších fází. Odezva je důležitá jednoduše proto, že uživatelé se ztrácejí a zlobí se, když jsou ignorováni. Aplikace ignoruje uživatele každou vteřinu, kdy nereaguje na jeho vstupy. [2][3]

3.3.3 Snímková frekvence

Snímková frekvence je asi tak důležitá jako "kvalita služeb". Displeje zařízení jsou navrženy tak, aby klamaly oči uživatelů dodáváním fotonů světla, takže obraz vypadá jako skutečný. Například papír pokrytý vytištěnými písmeny odráží fotony určitým způsobem. Manipulací s vykreslováním se čtecí aplikace snaží posílat fotony podobným způsobem a klamat tak oči.

Mozek ví, že pohyb není přerušovaný nebo diskrétní proces, ale plynulý a sekvenční. Zobrazovací zařízení s vysokou snímkovou frekvencí jsou vyrobena jednoduše proto, aby tato iluze byla reálnější. (Zajímavé je, že stroboskopická světla tento koncept převracejí naruby a způsobují, že mozek vytváří iluzi diskrétní reality). [2][5]

3.3.4 Využití paměti

Využití paměti je samostatnou klíčovou metrikou. Na rozdíl od odezvy a snímkové frekvence nemohou uživatelé přímo vnímat využití paměti, ale využití paměti ovlivňuje "stav uživatele". Ideální systém by udržoval 100% stav všech aplikací po celou dobu: všechny aplikace by běžely současně a každá aplikace by se vrátila do stavu, v jakém byla při poslední interakci uživatele s ní (stav aplikace je uložen v paměti počítače - takže toto jeho srovnání se stavem uživatele je poměrně přesné). [2][6]

Z toho plyne jeden ne zcela zřejmý závěr: dobře navržený systém se nestará o zvětšování volné paměti. Paměť je zdroj a volná paměť je nevyužitý zdroj. Naopak, dobře navržený systém využívá dostatek paměti k tomu, aby zajistil stav, kdy uživatel nepocítí žádnou změnu výkonu.

To neznamená, že by systém měl pamětí plýtvat. Pokud systém používá více paměti, než je nutné k udržení stavu aplikace - plýtvá pamětí.

V takovém případě jiná aplikace (nebo dokonce jiný stav), která by mohla tuto paměť využít - ji využít nemůže. V praxi žádný systém nemůže udržovat všechny stavy v paměti současně.

Poslední metrikou, kterou je třeba zmínit, je spotřeba energie. Podobně jako u využití paměti uživatel vnímá spotřebu energie nepřímo, a to tak, že si všímá doby, po které zařízení začne měnit uživatelem vnímaný výkon (UPP). Aby se minimalizovaly negativní dopady spotřeby energie, musí být systém energeticky účinný. [1][2]

3.3.5 Optimalizace výkonu pomoci JavaScriptu

Je velmi důležité zvážit, jakým je způsobem používáno JavaScript na webových stránkách, a zamyslet se nad tím, jak zmírnit případné problémy s výkonem, které může způsobovat. Zatímco obrázky a video tvoří více než 70 % bajtů stažených na průměrné webové stránce, JavaScript má větší potenciál negativního dopadu na výkon - může výrazně ovlivnit dobu stahování, výkon vykreslování a využití procesoru a baterie.

Při načítání stránky se obvykle nejprve analyzuje kód HTML, a to v pořadí, v jakém se na stránce objevuje. Kdykoli se objeví CSS, je analyzováno, aby se pochopily styly, které je třeba na stránku aplikovat. Během této doby se začnou načítat propojené prostředky, jako jsou obrázky a webová písma. Kdykoli se objeví JavaScript, prohlížeč jej analyzuje, vyhodnotí a spustí proti stránce. O něco později prohlížeč zjistí, jak by měl být každý prvek HTML stylizován vzhledem k CSS, které na něj bylo použito. Výsledek stylizace se pak vykreslí na obrazovku.

Klíčovým krokem je ve výchozím nastavení jsou parsování a spouštění JavaScriptu blokovány při vykreslování. To znamená, že prohlížeč blokuje rozbor jakéhokoli HTML, které se objeví po setkání s JavaScriptem, dokud není skript zpracován. V důsledku toho je blokováno i stylování a malování. [3][4]

3.3.6 Pořadí načítání prvků

Nejprve je důležité, aby se všechny prvky v sekci <head> načetly předem, ještě předtím, než návštěvník uvidí cokoli v prohlížeči, a poté aby se všechny následující prvky načetly v logickém pořadí. Jakýkoli JavaScript uvnitř sekce <head> může zpomalit vykreslování stránky. Při načítání neoptimalizované stránky je pravděpodobné, že se uživateli zobrazí "bílá obrazovka", než se načte celá stránka.

Optimalizované načítání stránky (ve skutečnosti vykreslování) probíhá spíše postupně, takže uživatel postupně vidí část obsahu, dokud se stránka nenačte celá.[4]

3.3.7 Minifikace kódu JavaScriptu pro menší velikost souborů a optimalizace samotného jazyka

Minifikace kódu – jedná se o metody transformace jazyka JavaScript - aby byl obtížněji čitelný nebo aby byl menší. Minifikace může zmenšit velikost souborů a zkrátit dobu načítání stránek.

Zalamování řádků, další mezery, komentáře atd. - to vše zvyšuje velikost souboru JavaScriptu a ovlivňuje rychlost načítání stránek. Komprimace kódu tento problém dobře řeší. Stroje nejsou citlivé na vizuální styl kódu jako člověk. Počítače dokáží minifikovaný kód přečíst a spustit, i když se celý Javascript vejde do jediného řetězce. [4]

3.3.8 Asynchronní načítání

Asynchronní načítání JavaScriptu je typ synchronního načítání. Znamená to, že se webové stránky načítají víceproudovým způsobem. Když prohlížeč najde řetězec s <script src="some.js"></script>, zastaví vytváření modelů DOM a CSSOM, zatímco se provádí JavaScript. Proto se většina kódu JavaScriptu nachází až za hlavním kódem HTML. Obvykle se do JavaScriptu přidáva asynchronní značka, aby vytváření modelu DOM probíhalo paralelně a nebylo přerušeno během načítání a provádění JavaScriptu. JavaScript musí provádět nějaké manipulace s HTML nebo CSS nebo pokud jde načítání skriptu v silném pořadí (např. knihovny závislé na jQuery). [4]

Většina vývojářů používá knihovny jako jQuery UI nebo jQuery Mobile tak, jak jsou. To znamená, že kód obsahuje všechny možné komponenty každé knihovny, i když může potřebovat pouze dvě nebo tři. Podobná situace nastává i u jiných knihoven JavaScriptu. Pokud je možnost řídit, které komponenty budou součástí balíčku knihovny, to je pohodlnější. Webové stránky se budou načítat mnohem rychleji a návštěvníci získají lepší zážitek. [4]

3.3.9 Používáni protokolu HTTP/2

Tato druhá, šifrovaná verze hlavního internetového protokolu poskytuje spoustu skvělých funkcí, včetně asynchronního stahování externích souborů, především JavaScriptu. Zatímco protokol HTTP vyžaduje hluboké učení a pokročilé znalosti teorie jazyka JavaScript, protokol HTTP/2 dokáže načítání jazyka JavaScript zrychlit. [4][6]

3.3.10 Náhodné globální proměnné

Globální proměnné jsou přístupné všem skriptům a funkcím v dokumentu JavaScriptu. Z tohoto důvodu nejsou globální proměnné automaticky vyčištěny garbage collectorem v jazyce JavaScript. Je důležité používat globální proměnné střídmě a nezapomenout je po použití ručně vynulovat nebo znovu přiřadit. [4][5]

3.3.11 Zavěšené proměnné vnější funkce v uzávěrech

Uzávěry mají přístup k proměnným a oboru vnější funkce. Podobně jako v případě náhodných globálních proměnných je možné, že funkce deklarovaná ve vnějším oboru zůstane v paměti i po vykonání vnější funkce, protože k ní má nějaká vnitřní funkce stále přístup, ale nepoužívá ji. [2][4]

3.3.12 Odpojené odkazy na DOM/odkazy mimo DOM

Objektový model dokumentu (DOM) je dvojnásobně provázaný strom, ve kterém jakýkoli odkaz na libovolný uzel ve stromu zabrání vybírání odpadu z celého stromu. Odpojený DOM nastane, když je uzel odstraněn ze stromu, ale stále zůstává v paměti prostřednictvím odkazu v rámci jazyka JavaScript. Pokud odkaz v JavaScriptu se neošetří, garbage collector odkaz nesmete a kód bude nadále spotřebovávat paměť.

3.3.13 Volání proměnných

Deklarace proměnných a zpětné volání referencí na ně je základem kódování. Jak však již dříve bylo uvedeno, pokaždé, když JavaScript drží odkaz na proměnnou, dochází ke spotřebě paměti a zvyšuje se možnost úniku paměti. Jak již bylo uvedeno výše, náhodné globální proměnné, závěsné uzávěry a reference „Out of DOM“ mohou mít potenciál ovlivnit výkon prostřednictvím úniků paměti. Prosté omezení počtu volání proměnných může vést k psaní stručnějšího výkonného kódu. [4]

3.3.14 Zmenšení velikosti DOM a přístupu

Objektový model dokumentu neboli DOM je datová reprezentace objektů, které tvoří strukturu webové stránky. Všechny webové stránky jsou dokumenty (obvykle HTML) a každý objekt v dokumentu se nazývá uzel. JavaScript přímo manipuluje s DOM a jeho uzly a mění strukturu, styl a obsah v reakci na vstup uživatele.

Pokaždé, když kód JavaScriptu přistupuje k prvku DOM nebo provede změnu v DOM, v závislosti na tom, co se děje, vyvolává se opětovné vykreslení části nebo celého dokumentu. To spotřebovává paměť a může zpomalit výkon, pokud systém musí přepočítávat velké množství uzlů v rozsáhlém DOM. Ořezávání velkých stromů DOM je dobrým začátkem při optimalizaci kódu.

3.3.15 Rozdělení kódu

Rozdělení kódu je praxe rozdělení kódu na funkční komponenty v rámci menších souborů, které lze volat podle potřeby. Zatímco celkové množství kódu je víceméně stejné, jako kdyby se použilo jeden soubor JavaScriptu, nahrazuje dobu načítání jednoho velkého souboru JavaScriptu dílčími dobami načítání pro konkrétní funkce a vlastnosti aplikace. [4]

3.3.16 Rozdělení dlouhých úkolů

Když prohlížeč spustí JavaScript, rozdělí skript do úloh, které se spouštějí postupně, jako jsou požadavky na načtení, řízení interakcí a vstupů uživatele prostřednictvím obsluhy událostí, spouštění animací řízených JavaScriptem atd. [3][5]

Pokud běh jedné úlohy trvá déle než 50 ms, je klasifikována jako dlouhá úloha. Pokud se uživatel pokusí o interakci se stránkou nebo je vyžádána důležitá aktualizace uživatelského rozhraní v době, kdy běží dlouhá úloha, bude to mít vliv na jeho zážitek. Očekávaná odezva nebo vizuální aktualizace se zpozdí, což způsobí, že uživatelské rozhraní bude vypadat pomalé nebo nebude reagovat. [3][4]

Rozdělení dlouhých úloh na menší. To dává prohlížeči více příležitostí provést důležité zpracování interakce s uživatelem nebo aktualizace vykreslení uživatelského rozhraní - prohlížeč je může případně provést mezi jednotlivými menšími úlohami, nikoli pouze před nebo po dlouhé úloze. V JavaScriptu to lze provést tak, že kód se rozdělí na samostatné funkce. To má smysl i z několika dalších důvodů, například pro snazší údržbu, ladění a psaní testů. [3]

```JS
function main() {
  a();
  b();
  c();
  d();
  e();
}
```

Obrázek 6. Rozdělení dlouhých úloh na menší. Zdroj: https://developer.mozilla.org/en-US/docs/Learn/Performance/JavaScript

```JS
async function main() {
  // Create an array of functions to run
  const tasks = [a, b, c, d, e];

  while (tasks.length > 0) {
    // Yield to a pending user input
    if (navigator.scheduling.isInputPending()) {
      await yield();
    } else {
      // Shift the first task off the tasks array
      const task = tasks.shift();

      // Run the task
      task();
    }
  }
}
```

Obrázek 7. Funkce setTimeout() k odložení provádění do samostatné úlohy. Zdroj: https://developer.mozilla.org/en-US/docs/Learn/Performance/JavaScript

Abychom se dalo s tím vypořádat, je tendenci pravidelně spouštět funkci "yield", aby se kód vzdal hlavnímu vláknu. To znamená, že kód je rozdělen do více úloh, mezi jejichž prováděním má prohlížeč možnost zpracovávat úlohy s vysokou prioritou, jako je například aktualizace uživatelského rozhraní. Běžný vzor pro tuto funkci používá funkci setTimeout() k odložení provádění do samostatné úlohy (obr.7) [3][4]

3.3.17 Zpracování animací v jazyce JavaScript

Animace mohou zlepšit vnímání výkonu, díky nimž se rozhraní zdá být svižnější a uživatelé mají pocit, že se při čekání na načtení stránky děje nějaký pokrok (například načítání spinnerů). Větší animace a větší počet animací však přirozeně vyžadují větší výpočetní kapacitu, což může snížit výkon. [4][5][10]

Pro optimalizace je vhodné používat méně animací - vyškrtnout všechny nepodstatné animace a zvážit, zda uživatelům neposkytnout předvolbu, kterou mohou nastavit pro vypnutí animací, například pokud používají zařízení s nízkým výkonem nebo mobilní zařízení s omezenou kapacitou baterie.

Pro nezbytné animace DOM se používá animace CSS, pokud je to možné, spíše než animace v JavaScriptu (rozhraní Web Animations API poskytuje způsob, jak se přímo připojit k animacím CSS pomocí JavaScriptu). Použití prohlížeče k přímému provádění animací DOM namísto manipulace s inline styly pomocí JavaScriptu je mnohem rychlejší a efektivnější. [3][10]

3.3.18 Optimalizace výkonu události

Sledování a zpracování událostí může být pro prohlížeč náročné, zejména pokud událost probíhá nepřetržitě. Například lze sledovat polohu myši pomocí události mousemove a kontrolovat, zda se stále nachází v určité oblasti stránky.

Použití delegace událostí všude, kde je to možné. Pokud je nějaký kód, který má být spuštěn v reakci na interakci uživatele s některým z velkého počtu podřízených prvků, nastaví se posluchače událostí na jejich rodiče. Události vyvolané na kterémkoli podřízeném prvku se promítnou do jeho rodiče, takže se nemusí nastavovat posluchače událostí na každém podřízeném prvku zvlášť. Méně posluchačů událostí, které je třeba sledovat, znamená lepší výkon. [3]

3.4 Načítání obsahu webové aplikace

Rychlost, jakou se webová stránka načítá v prohlížeči, je faktor, který na návštěvníka působí po každém kliknutí, proto je důležité, aby ho pomalejší načítání stránky neobtěžovalo. Ideální stav je ten, kdy si návštěvník vůbec neuvědomí, že na něco musí čekat. Většina lidi už řešila otázku, jak načítání stránky urychlit.[5][6][7]

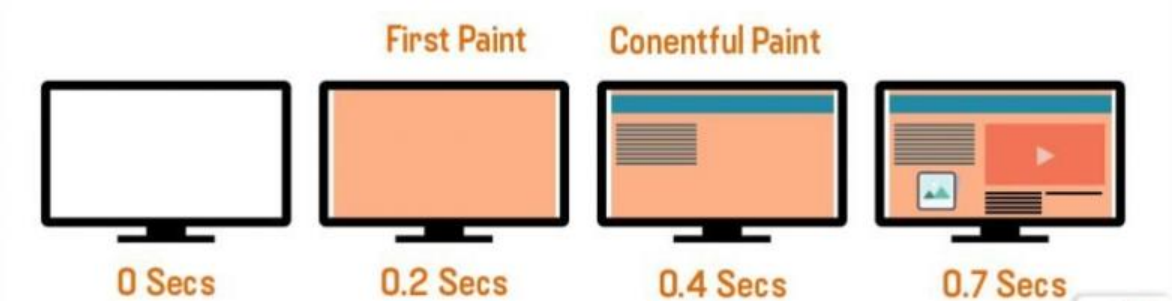
3.4.1 First Contentful Paint

První barva obsahu (FCP) je první obsah, který se zobrazí na obrazovce, když uživatelé procházejí webové stránky. Měří dobu od navigace do okamžiku, kdy prohlížeč vykreslí první část obsahu definovaného v objektovém modelu dokumentu (DOM). Může se jednat o text, obrázek nebo vykreslení plátna.

V případě webu WordPress jsou prvky záhlaví prvním obsahem, který se začne vykreslovat. Návštěvník vidí logo webu a navigační nabídku. V první fázi se zobrazuje první malování a malování obsahu.

První malování se spustí, když je v prohlížeči zjištěno vykreslování. Může to být něco tak jemného a neinformativního, jako je změna barvy pozadí. Problém s tímto načasováním spočívá v tom, že první malování může být spuštěno relativně brzy po načtení stránky, přičemž nemusí nutně poskytnout uživateli žádný obsah nebo informace ke konzumaci. Součásti webové stránky se mohou načítat postupně, a zatímco barva pozadí se může namalovat rychle, skutečný obsah/interaktivita se může načítat delší dobu. [7]

Funkce First Contentful Paint se spustí, když je namalován první kousek obsahu z objektového modelu dokumentu (DOM). Může to být text, obrázek nebo vykreslení plátna definované v DOM. Protože se zaměřuje na obsah, je myšlenka taková, že tato metrika poskytuje představu o tom, kdy uživatel obdrží konzumovatelné informace, jako je text, vizuál atd. [5][6]



Obrázek 8. Princip fungování First Contentful Paint. Zdroj: https://www.linkedin.com/pulse/first-contentful-paint-fcp-meaningful-fmpexplained-bikash-rai/

3.4.2 First Meaningful Paint

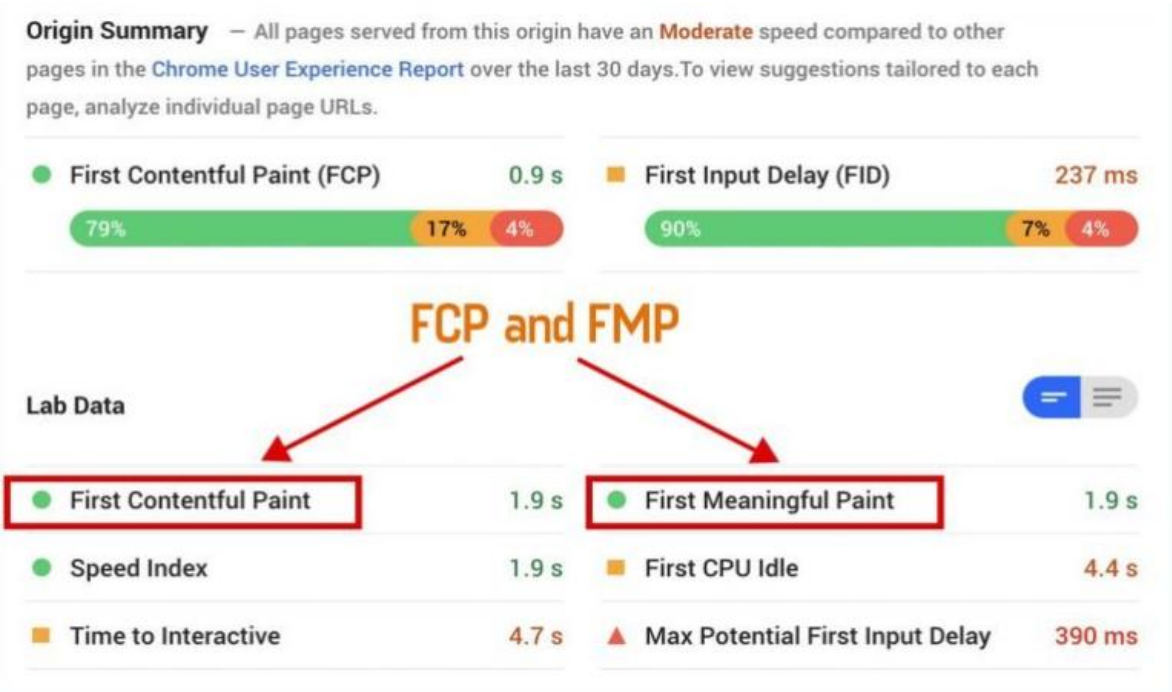
První významná barva neboli FMP měří dobu viditelnosti primárního obsahu webové stránky. Je to čas, kdy je větší obsah webové stránky viditelný pro koncového uživatele a vyvolává v něm dojem hlavních změn v rozložení. First Meaningful Paint zahrnuje také načítání aktuálních písem. [5][7]

Podle definice FCP poskytuje First Contentful Paint (První obsahový obraz) čas, kdy prohlížeč vykreslí první část obsahu, jako je text, obrázek, nebělavé plátno, indikátor načítání atd. ale tyto obsahy nejsou smysluplné. Okamžik First Meaningful Paint činí webovou stránku uživatelsky čitelnou a vizualizuje primární obsah smysluplně. Smysluplné informace se liší web od webu. [5][8]

V praxi bylo FMP příliš citlivé na malé rozdíly v načítání stránky, což vedlo k nekonzistentním (bimodálním) výsledkům. Definice metriky navíc závisí na implementačních detailech specifických pro prohlížeč, což znamená, že ji nelze standardizovat ani implementovat ve všech webových prohlížečích. [8]

Obrázek 9. Načítání obsahu webové stránky. Zdroj: https://www.perfmatrix.com/largest-contentful-paint/

Kdykoli se testuji webové stránky v nástrojích pro testování rychlosti, jako je PageSpeed Insights společnosti Google, GTMetrix.com, je vidět výsledek pomocí First Contentful Paint a First Meaningful Paint spolu s dalšími metrikami rychlosti. Jedná se o důležité metriky zaměřené na uživatele, které mohou hodně napovědět o tom, jak dlouho návštěvníci čekají na obsah. [5]



Obrázek 10. Zpráva o testování webové stránky od PageSpeed Insights. Zdroj: PageSpeed Insights

3.4.3 First Input Delay

Zpoždění prvního vstupu (FID) je metrika uživatelské zkušenosti, kterou Google používá jako malý faktor hodnocení. Zpoždění prvního vstupu je víc než jen snaha vyhovět Googlu.

FID je měření doby, za kterou prohlížeč zareaguje na první interakci návštěvníka s webem během načítání webu. Někdy se nazývá vstupní zpoždění. Interakcí může být klepnutí na tlačítko, odkaz nebo stisknutí klávesy a následná reakce. Dalšími druhy bodů interakce, které FID měří, jsou oblasti pro zadávání textu, rozevírací nabídky a zaškrtávací políčka. Cílem FID je měřit, jak je web během načítání citlivý.

První vstupní zpoždění je obvykle způsobeno obrázky a skripty, které se stahují neuspořádaně. Toto neuspořádané kódování způsobuje, že se stahování webové stránky nadměrně pozastavuje, pak se spustí a zase pozastaví. To způsobuje nereagující chování návštěvníků webu, kteří se pokoušejí s webovou stránkou komunikovat. Je to jako dopravní zácpa způsobená volným průjezdem, kde nejsou žádné semafory. Řešení spočívá ve zjednání pořádku v dopravě. [6][7]

Události, které se počítají jako uživatelský vstup měřený pomocí FID, musí být diskrétní (konečné). Spojité typy interakce uživatele, jako je zvětšování nebo posouvání stránky, nelze pomocí této metriky přesně měřit. Je to proto, že často neběží v hlavním vlákně prohlížeče a mají jiná omezení.[7][8]

Navíc většina blokování hlavního toku prohlížeče probíhá v prvních okamžicích životního cyklu webové stránky - tehdy se načítají kritické zdroje. FID je metrika, která pomůže tuto situaci řešit a zajistit, aby načítání těchto kritických zdrojů nezpůsobilo, že webové stránky budou působit těžkopádně a nereagovat.[6][7]

Skutečné zpracování nebo aktualizace webové stránky v důsledku této interakce se pomocí FID neměří. Je to proto, že pro vývojáře by bylo snadné hrát si s FID tím, že by oddělili obsluhu události od úlohy spojené s událostí.[5][7]

Ke zpoždění vstupu dochází, když se prvky stránky, jako jsou obrázky nebo skripty, načítají bez požadavku uživatele. Podle společnosti Google je jednou z příčin dlouhých vstupních prodlev spouštění JavaScriptu. Týká se to zejména velkých souborů JavaScriptu, které musí prohlížeč spustit před spuštěním jakýchkoli posluchačů událostí. Načítaný kód JavaScriptu může změnit následné akce prohlížeče.[7]

Může způsobit, že webové stránky nebudou reagovat, protože prohlížeč bude čekat na určení dalších kroků, což přispívá k dlouhému vstupnímu zpoždění. Prohlížeč má pocit, že uvízl v dopravní zácpě.[7]

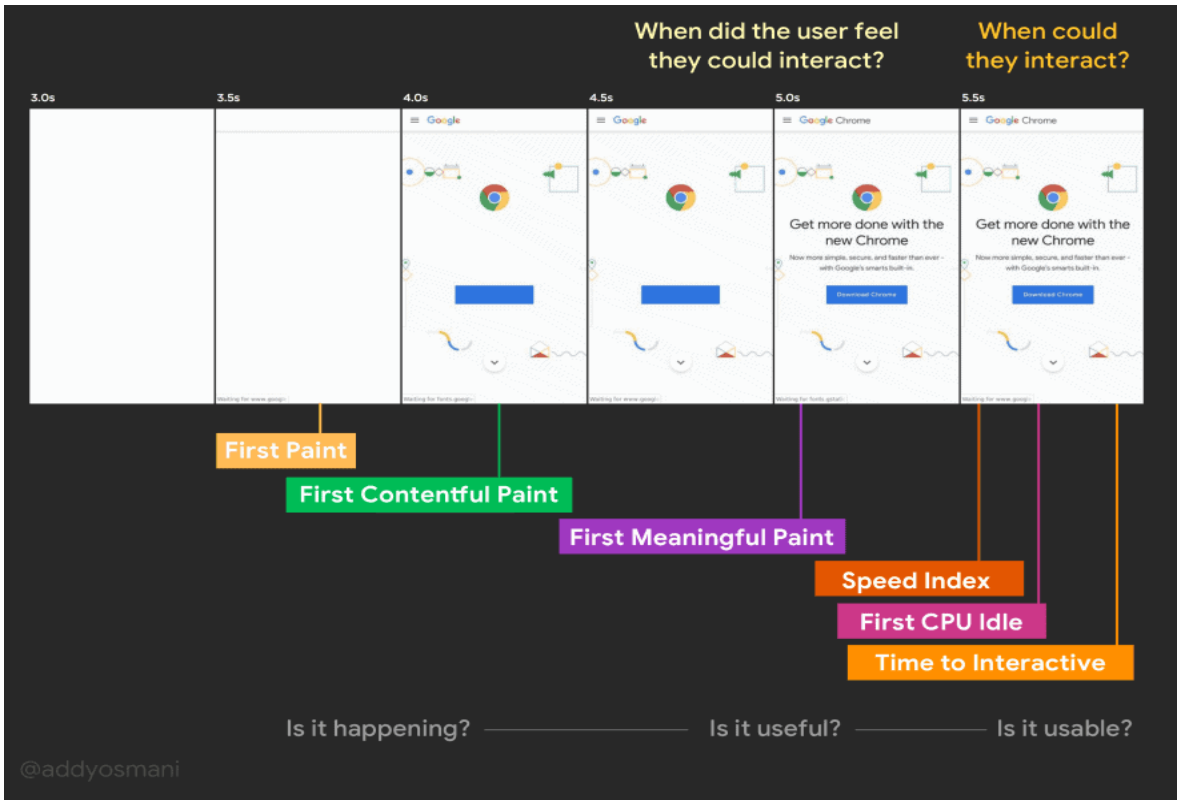
3.4.4 Time to Interactive (TTI)

Metrika TTI měří dobu od zahájení načítání stránky do načtení jejích hlavních dílčích zdrojů a je schopna spolehlivě a rychle reagovat na vstup uživatele.

Historicky vývojáři optimalizovali stránky pro rychlé vykreslování, někdy na úkor TTI. Techniky, jako je vykreslování na straně serveru (SSR), mohou vést ke scénářům, kdy stránka vypadá interaktivně (tj. odkazy a tlačítka jsou na obrazovce viditelné), ale ve skutečnosti není interaktivní, protože hlavní vlákno je zablokované nebo protože se nenačetl kód JavaScriptu ovládající tyto prvky.

Když se uživatelé pokusí interagovat se stránkou, která vypadá interaktivně, ale ve skutečnosti není, budou pravděpodobně reagovat jedním ze dvou způsobů. V lepším případě budou naštvaní, že stránka reaguje pomalu. V horším případě budou předpokládat, že je stránka nefunkční, a pravděpodobně ji opustí. Možná dokonce ztratí důvěru nebo důvěru v hodnotu stránky. Třeba-li se tomuto problému vyhnout, tak je třeba minimalizovat rozdíl mezi FCP a TTI. A v případech, kdy znatelný rozdíl přece jen existuje, lze dat prostřednictvím vizuálních ukazatelů jasně najevo, že komponenty na stránce ještě nejsou interaktivní. [8]

Skóre TTI souvisí se zpožděním prvního vstupu (FID), ale metrika výkonu Time to Interactive se počítá jinak. Uživatelské vnímání toho, jak rychle se web načítá a jak rychle s ním může interagovat, je obtížné zachytit pomocí jediné metriky výkonu. Je to proto, že načítání stránky probíhá v několika krocích, které mají vliv na uživatelský zážitek a interaktivitu. [9]



Obrázek 11. Metrika interaktivní stránky TTI. Zdroj: https://edgemesh.com/blog/what-is-time-to-interactive

Některé weby optimalizují rychlost načítání na úkor interaktivity. To může být pro uživatele frustrující. Webová stránka načte za krátkou dobu, ale když s ní da interagovat kliknutím na tlačítko, nic se nestane! To je problém pomalých webů s časem pro interaktivitu a hlavní důvod, proč je třeba zlepšit TTI, aby bylo návštěvníkům zajištěno co nejlepší zážitek. [5][9]

Metrika interaktivní stránky TTI je tedy doba, která uplynula mezi počátečním bodem (1) a koncem (2) definovaným dokončením poslední dlouhé úlohy před pětisekundovou nečinností (obr.11). Vzhledem k této definici nelze měřit čas do interaktivity před analýzou načtení celé stránky. [6][7]

Největším faktorem, který způsobuje zpoždění v Time To Interactive, je spouštění Javascriptu blokující vykreslování stránky. Čím více skriptů se načítá, tím delší je zpoždění TTI. Výkon skriptů a jejich vliv na Time to Interactive značně závisí na zařízení uživatele. Čím pomalejší je procesor, tím déle trvá prohlížeči analyzovat a kompilovat skripty. Proto se uživatelé mobilních zařízení často setkávají s problémy s výkonem webu. [7][9]

3.5 Shrnutí

Z výše uvedených výsledků a teoretických poznatků výzkumu lze vyvodit důkaz, že jednotlivé principy a techniky mají na výkonnost webové stránky nebo aplikace patřičný význam a vliv. Výzkum teoretických dat zdůraznil a analyzoval klíčové výkonnostní metriky, mezi něž patří First meaningful paint (FMP), First input delay (FID) a Time to interactive (TTI), které významně ovlivňují vnímání načtené stránky uživatelem. Byly přezkoumány současné přístupy a techniky optimalizace webových stránek a jejich použitelnost v praxi. Velmi důležitou roli hrají značkovací jazyky HTML a CSS pro rychlost vykreslování stránek a programovací jazyk Javascript pro rychlost optimalizace a načítání samotné webové stránky nebo webové aplikace.

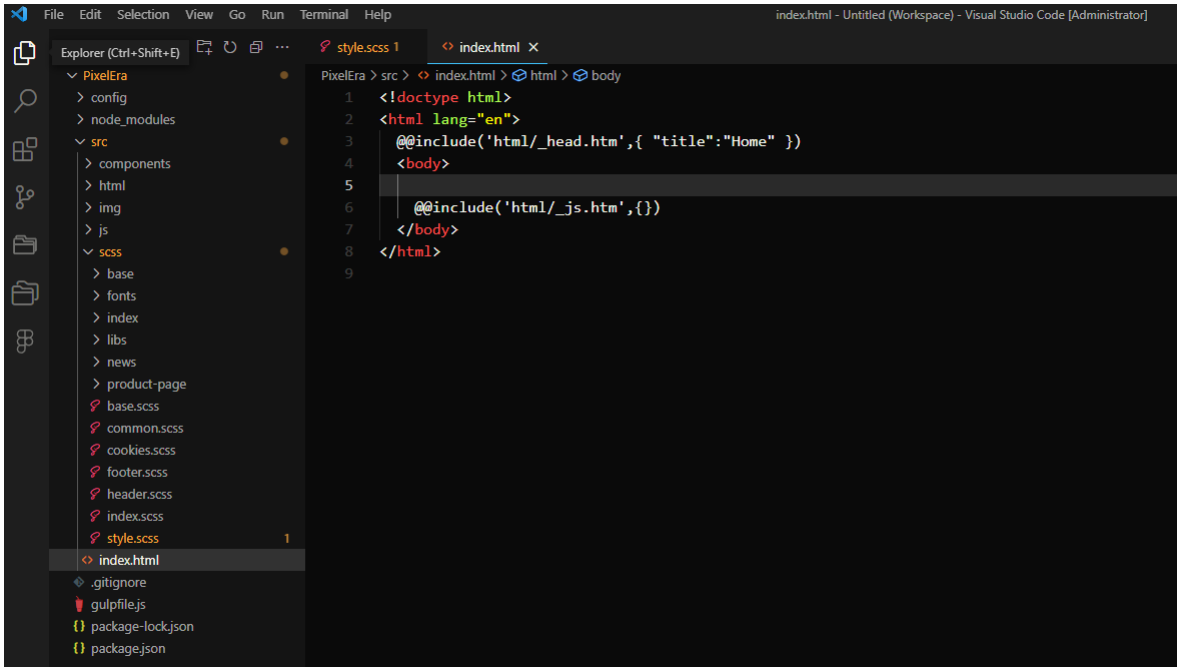
Na základě těchto výsledků bude provedena výzkumná práce s využitím webových technologií s cílem dosáhnout nejlepšího způsobu řešení v oblasti vývoje webových aplikací, na základě jejíchž výsledků bude učiněno vše možné pro zrychlení načítání a optimalizaci webové stránky.

4 Vlastní práce

4.1 Vytvoření webové stránky

4.1.1 Soubor HTML

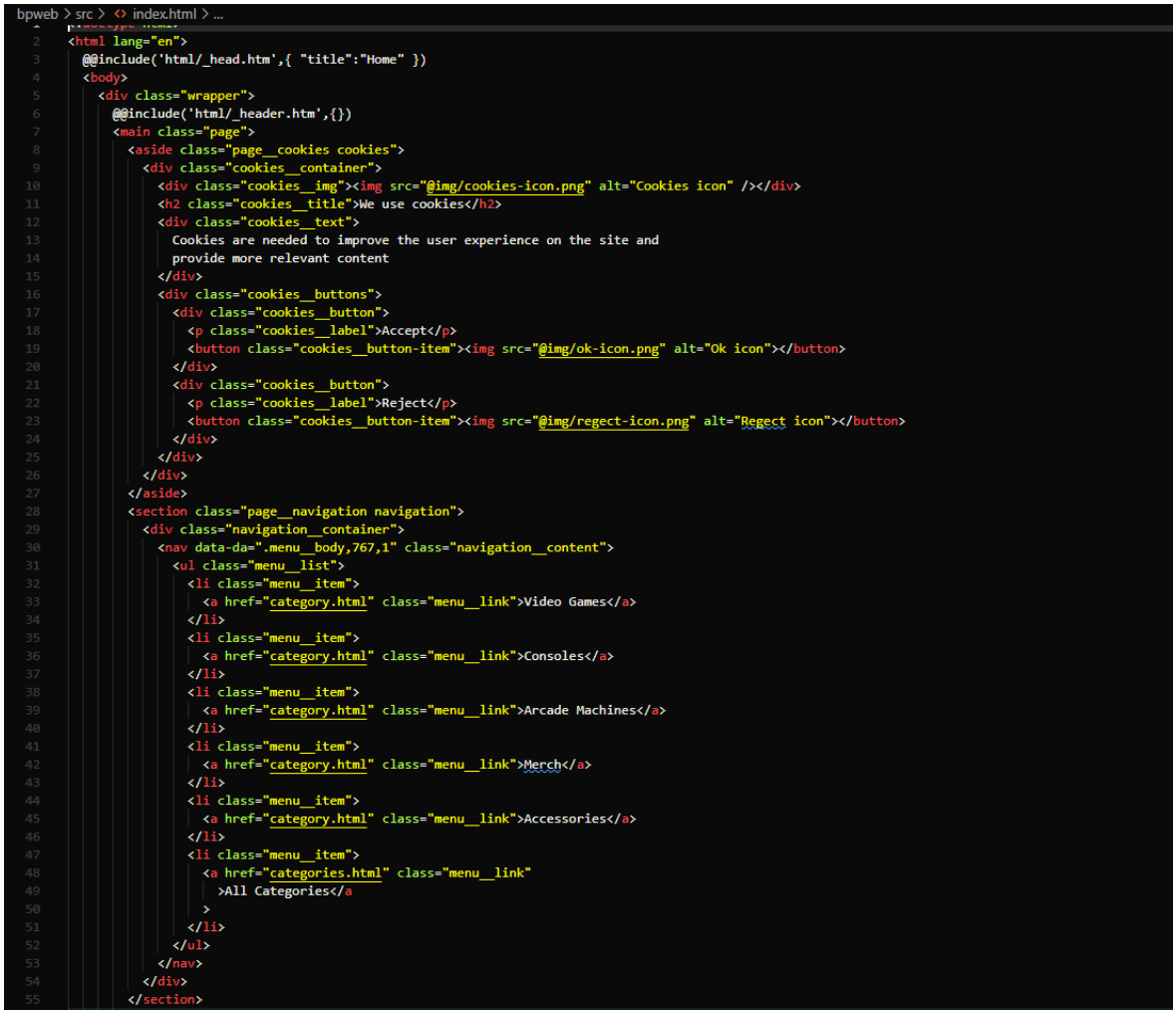
Pro testování a analýzu výše uvedených vlastností načítání webové aplikace je třeba vytvořit testovací webovou stránku. K zápisu kódu byl použit textový editor Visual Studio Code. Pro strukturu kódu a jeho rozdělení byly použity webové technologie HTML, CSS s preprocesorem SCSS a Javascript. A také nástroje pro optimalizaci kódu Gulp a Webpack.



Obrázek 12. Vytváření prostředí pro tvorbu aplikací. Zdroj: Vlastní zpracování

Jazyk HTML je základem pro tvorbu webových stránek, a pokud je použit správně, umožňuje rychlé načítání webových stránek. Definuje základní strukturu webové stránky. Prvky jako <html>, <head> a <body> pomáhají uspořádat a prezentovat obsah webové stránky. Značkovací jazyk poskytuje sémantické značky, pro význam obsahu. Značky <h1>-<h6> se používají pro nadpisy, značky <p> pro odstavce a značky <a> pro vytváření odkazů. Díky tomu je kód nejen čitelnější, ale má také pozitivní vliv na SEO a přístupnost webové stránky.

Pro lepší výsledky testování byla vytvořena webová stránka simulující internetový obchod s velkým počtem obrázků. Pomocí této ukázkové webové aplikace bylo možné použít všechny techniky pro zlepšení výkonu a změřit požadované parametry webu. Použití značek <article>, <figure>, <figcaption>, <header> a <footer> v jazyce HTML umožňuje vytvářet sémanticky správnější a strukturovanější popisek webové stránky. Právě díky této struktuře byly vytvořeny samostatné části testované stránky.



Obrázek 13. Struktura webové aplikace. Zdroj: Vlastní zpracování

Tento obrázek ukazuje (obr. 13), na počátek napsaní samotného webového dokumentu. Pro správnou práci webu je nutné dodržovat pravidla psaní kódu a sémantiku, aby nedocházelo k chybám a špatnému výkonu webové aplikace.

4.1.2 Soubor CSS

CSS je mocný nástroj pro vytváření krásných, přizpůsobivých a efektivních webových stránek. V kombinaci s jazyky HTML a JavaScript poskytuje CSS kompletní a moderní prostředí pro tvorbu webových stránek. Vývojáři mohou definovat vzhled webové stránky. Styly lze použít k ovládání barev, písem, odrážek, velikostí prvků a dalších aspektů designu. Jedním ze základních principů tvorby webových stránek je oddělení obsahu (HTML) a prezentace (CSS). To usnadňuje údržbu a úpravy webových stránek, protože změny stylů nevyžadují změny samotného kódu HTML.

Vytváření responzivních webových stránek, které se snadno přizpůsobí různým zařízením a obrazovkám. Media dotazování a další techniky umožňují přizpůsobení designu velikosti obrazovky a poskytují příjemné uživatelské prostředí na mobilních zařízeních, tabletech a počítačích. Zlepšení přístupnosti webových stránek pro uživatele se zdravotním postižením. Pro zlepšení čitelnosti lze snadno změnit velikost písma nebo barvy.

```scss
.section-title {
  color: $yellowColor;
  font-family: "Press Start 2P";
  font-size: rem(43);
  font-weight: 400;
  line-height: normal;
}

.main-page-section-sellers {
  @include adaptiveValue("margin-top", 45, 15);
  @include adaptiveValue("margin-bottom", 45, 15);
  padding-top: rem(30);
  padding-bottom: rem(30);
  background-color: rgba(17, 21, 34, 0.7);
  // .main-page-section-sellers__container

  &__container {
    display: flex;
    flex-wrap: wrap;
    flex-direction: column;
    gap: rem(25);
  }

  // .main-page-section-sellers__head

  &__head {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    gap: rem(15);
    @media (min-width: em(580)) {
      flex-direction: row;
      justify-content: space-between;
    }
  }

  // .main-page-section-sellers__content

  &__content {
  }

  &__adaptive-move {
    @media (min-width: $mobile) {
      display: none;
    }
  }
}
```

Obrázek 14. Zahájení práce s kaskádovými styly webové aplikace. Zdroj: Vlastní zpracování

Tento příklad (obr. 14) ukazuje vytvoření souboru css propojeného samostatným souborem v html, což již může urychlit práci s webovou stránkou jako celkem a poskytnout přístupnost pro snadnou správu souborů. Optimalizované styly mohou pomoci zlepšit rychlost načítání webové stránky, zejména pokud se používá ukládání do mezipaměti a miniaturizace souborů CSS. To je důležité pro zajištění lepšího uživatelského zážitku a zlepšení výkonu SEO.

4.1.3 Soubor Javascript

JavaScript ve spojení s HTML a CSS tvoří základ pro vývoj moderních, dynamických a interaktivních webových stránek a webových aplikací. Hraje klíčovou roli ve struktuře vývoje webových stránek a poskytuje interaktivitu, dynamiku a různé funkce. Umožňuje vytvářet interaktivní prvky a dynamicky měnit obsah webové stránky v reakci na akce uživatele. Tvoří formuláře s validací, posuvníky, rozevírací seznamy a další prvky, které reagují na interakci uživatele.

```
(() => {
    "use strict";
    function isWebp() {
        function testWebP(callback) {
            let webP = new Image;
            webP.onload = webP.onerror = function() {
                callback(webP.height == 2);
            };
            webP.src = "";
        }
        testWebP((function(support) {
            let className = support === true ? "webp" : "no-webp";
            document.documentElement.classList.add(className);
        }));
    }
    let isMobile = {
        Android: function() {
            return navigator.userAgent.match(/Android/i);
        },
        BlackBerry: function() {
            return navigator.userAgent.match(/BlackBerry/i);
        },
        iOS: function() {
            return navigator.userAgent.match(/iPhone|iPad|iPod/i);
        },
        Opera: function() {
            return navigator.userAgent.match(/Opera Mini/i);
        },
        Windows: function() {
            return navigator.userAgent.match(/IEMobile/i);
        },
        any: function() {
            return isMobile.Android() || isMobile.BlackBerry() || isMobile.iOS() || isMobile.Opera() || isMobile.Windows();
        }
    };
```

Obrázek 15. Vytvoření souboru Javascript pro ovládání dynamiky webu. Zdroj: Vlastní zpracování

Tento příklad (obr. 15) ukazuje výsledek zahájení práce na vnesení dynamiky do webu a optimalizaci webové aplikace pomocí programovacího jazyka Javascript. Přístup a možnost měnit strukturu a obsah webové stránky prostřednictvím DOM. To umožňuje dynamicky aktualizovat a upravovat prvky stránky, aniž by bylo nutné načítat celou stránku.

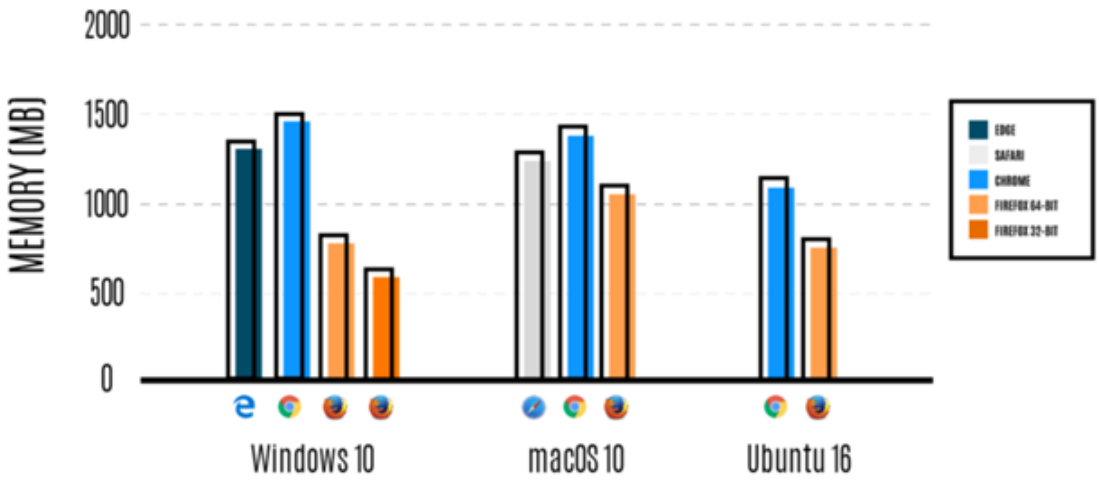
4.2 Využití metod pro zlepšení výkonu webové aplikace

4.2.1 Vliv optimalizace souborů na zpracování v prohlížeči

Velikost webové stránky je důležitá a má vliv na různé aspekty výkonu webu. Výzkumy dokonce ukázaly, že uživatelé očekávají načtení webových stránek během několika sekund a jakékoli zpoždění může vést k vyššímu počtu odmítnutých stránek a nižší angažovanosti.

Hmotnost webové stránky přímo ovlivňuje rychlost načítání, protože větší soubory se stahují déle. Optimalizací velikosti webových stránek lze výrazně zlepšit dobu načítání a udržet pozornost uživatelů. Důležitý vliv má velikost webové stránky na faktory hodnocení ve vyhledávačích.

Rychlejší webové stránky se ve výsledcích vyhledávání umisťují výše, protože vyhledávače jako Google upřednostňují poskytování pozitivního uživatelského zážitku. Proto optimalizace hmotnosti webových stránek zlepšuje rychlost načítání a zvyšuje možnosti dosažení lepší viditelnosti stránky.



Obrázek 16. Využití paměti prohlížeči v megabitech. Zdroj: https://supuniuthpalameegahapola.medium.com/how-web-browsers-use-process-threads-695a6f42e9a8

V této praxi byly provedeny různé metody optimalizace webové stránky založené na známých metodách úpravy struktury souborů webové aplikace s cílem dosáhnout co nejvyššího výkonu a načítání webové stránky v prohlížeči. Aby bylo možné správně aplikovat techniky správy paměti a zpráv prohlížeče na server

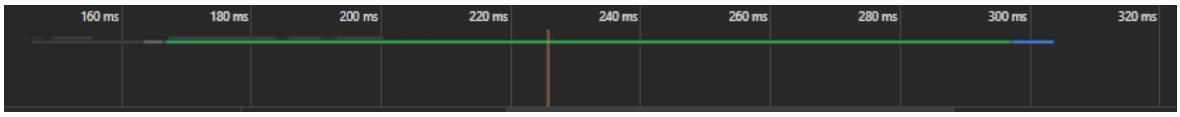
Optimalizace obrázků. Protože obrázky se významně podílejí na velikosti webové stránky, je jejich změna velikosti a komprese bez snížení kvality úplně prvním a důležitým krokem. Místo těžkých formátů obrázků, jako jsou PNG a JPEG, je vhodné používat moderní formáty obrázků, jako jsou WebP a AVIF, které poskytují lepší kompresi a rychlejší načítání.

Minifikace CSS a JavaScript. Dalším krokem je minimalizace nepotřebných znaků, bílých znaků a komentářů ze souborů CSS a JavaScript. To pomáhá snížit velikost souborů, což zlepšuje dobu načítání. Minifikační nástroje a zásuvné moduly mohou pomoci tento proces automatizovat a zajistit tak optimální snížení velikosti souborů.

Implementace líného načítání. Pomalé načítání funguje tak, že odkládá načítání určitých zdrojů, dokud nejsou nezbytně nutné, například když se dostanou do zobrazení uživatele. Tento přístup výrazně zlepšuje počáteční dobu načítání stránky tím, že upřednostňuje nezbytný obsah.

Optimalizovaná písma. Písma maji významný vliv na velikost webové stránky při lepší kompresi. K minimalizaci velikosti webové stránky může přispět i omezení stylů a vah písem používaných na webu.

Sledování a analýza výkonu. Nástroje pro měření výkonu webu, jako je Google PageSpeed Insights, poskytují informace o velikosti webové stránky, době načítání a dalších výkonnostních ukazatelích.



Obrázek 17. Měření rychlosti načítání prvků testovací stránky bez optimalizace. Zdroj: Vlastní zpracování

Jak je vidět(obr.17), rychlost načítání v milisekundách je poměrně vysoká i při malém počtu prvků na stránce. Tyto charakteristiky ukazují zprávu o zobrazování souborů v prohlížeči. Vysoký index znamená dlouhé načítání prvků, které zabírá značnou část výkonu procesoru a prohlížeče.

Name	Status	Type
index.html	304	document
logo.png	200	png
user-avatar.jpg	200	jpeg
consoles02.jpg	200	jpeg
consoles03.png	200	png
consoles04.png	200	png
news01.png	200	png
news02.png	200	png
news03.png	200	png
logo.png	200	png
instagram.png	200	png
facebook.png	200	png
app.min.js?_v=20240224000455	200	script
style.min.css?_v=20240224000455	200	stylesheet
money-img.png	404	text/html

Obrázek 18. Příklad zpracování jednotlivých prvků stránky. Zdroj: Vlastní zpracování

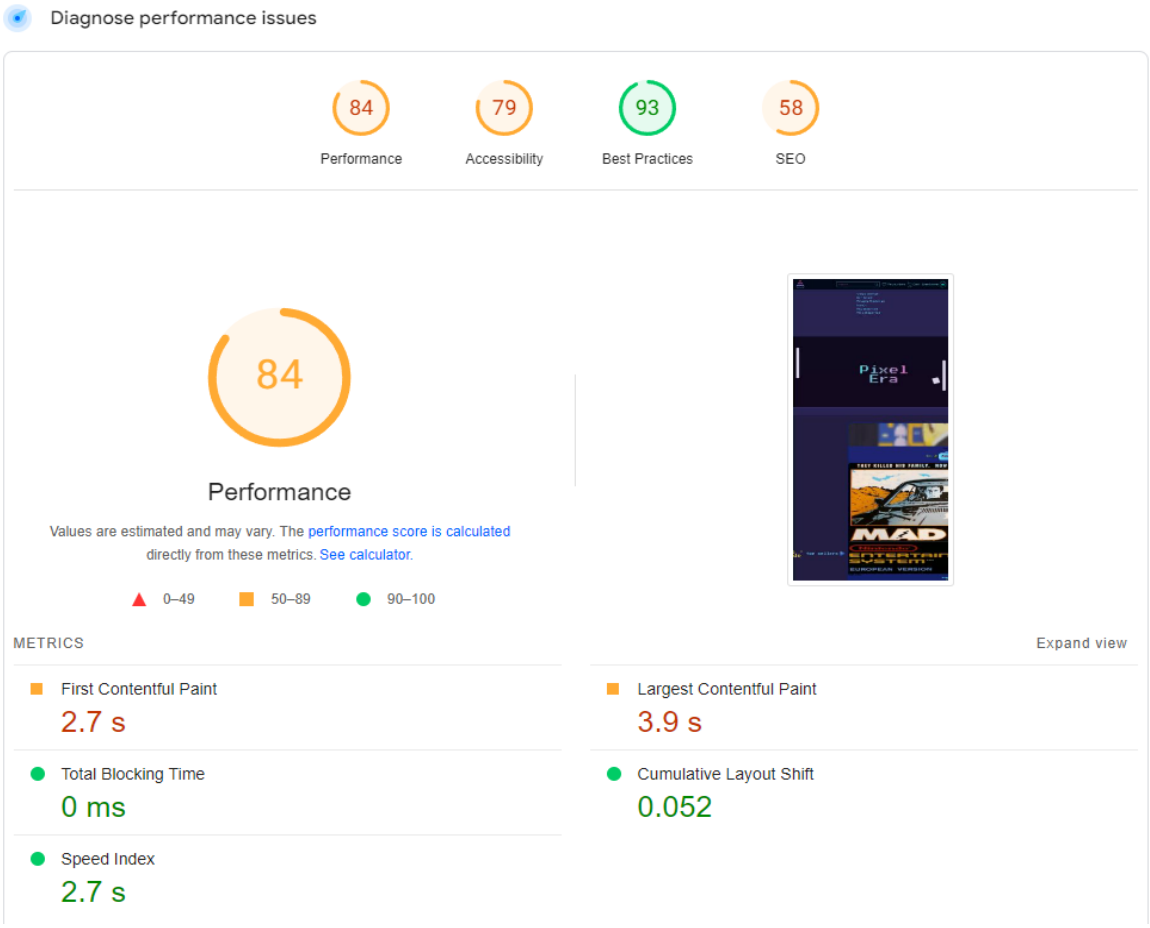
Zde jsou zobrazeny některé prvky stránky(obr.18). V tomto případě jsou všechny obrázky bez komprese a neoptimalizované. To výrazně snižuje výkon stránky. Analýza některých částí stránky pomohla odstranit problematické prvky ovlivňující výkon prohlížeče a rychlost spojení se serverem.

4.2.2 Sémantika a struktura kódu

Pro dobrý výkon webové stránky je důležitá správná struktura kódu a vyloučení faktorů, které mohou porušit pravidla sémantiky. Jednou z hlavních priorit pro rychlé načítání je rychlost načítání kódu prohlížeči. K tomu je potřeba dodržovat jasná pravidla interpretace kódu.

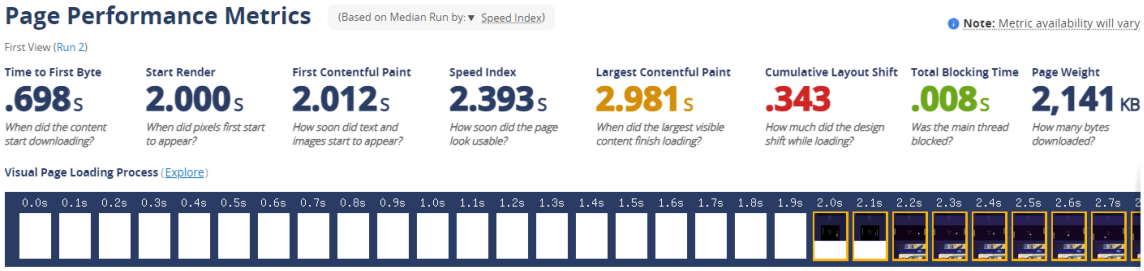
Sémanticky správný kód HTML usnadňuje práci vyhledávačům, jako je Google. Pokud je kód napsán správně, mohou vyhledávací roboti rychle analyzovat webový dokument. Dobré značky přístupnosti, jako např: "header", "nav", "footer", usnadňují přístupnost webu pro osoby se zdravotním postižením. Přehlednou strukturu kódu a sémantické použití prvků lze využít k optimalizaci načítání webového dokumentu.

Pokud je kód špatně strukturovaný a obsahuje velké množství chyb, může dojít k vysokým nárokům na zobrazení dat a uživatel si může myslet, že je stránka nedostupná nebo že došlo k neočekávané chybě. Snižuje se také výkon aplikace v důsledku snahy prohlížeče správně sestavit a zobrazit požadované informace. To často snižuje rychlost zobrazení obsahu, protože prohlížeč potřebuje čas na zpracování a změnu struktury pro co nejsprávnější interpretaci kódu.



Obrázek 19. Příklad špatného načítání webové stránky. Zdroj: Vlastní zpracování, PageSpeed Insights

Na tomto příkladu (obr. 19) lze vidět nízkou rychlost načítání webových stránek, která je způsobena naprostým nedostatkem zásad a metod pro zlepšení načítání a výkonu webové aplikace. V tomto případě musí prohlížeč provést řadu akcí sám, aby zobrazil obsah webové stránky bez jakýchkoli pomocných nastavení.



Obrázek 20. Zpráva o testu webové stránky od Webpage Test. Zdroj: Vlastní zpracování, Webpage test

Zde je jasně vidět (obr. 20), že díky obtížnému strukturování kódu a bez ovlivnění a zpracování obrázků trvá prohlížeči příliš dlouho, než zobrazí obsah stránky a všechny obrázky na ní. Také je vidět, že výsledky First Contentful Paint a Time To Interactive jsou vysoké, což přímo ovlivňuje výkon stránky jako celku.

4.2.3 Přidání metaznaček

V procesu optimalizace textových dat a přizpůsobení prohlížeče různým zařízením byly přidány určité metaznačky, které zapisují pravidla pro práci webové stránky s prohlížečem.

Metaznačky v jazyce HTML hrají důležitou roli při optimalizaci načítání webových stránek, zlepšování zobrazení obsahu a optimalizaci pro vyhledávače. Některé metaznačky mohou pomoci ve struktuře kódu webové stránky zlepšit rychlost načítání a vnímání zobrazení obsahu na webové stránce prohlížečem, přizpůsobit stránku různým uživatelským zařízením. Metaznačky pomohou dosáhnout rychlosti a zlepšit metriky First Contentful Paint a Time To Interactive.

Zde jsou některé z jejich příkladů:

- **<meta charset="UTF-8">**

UTF-8 (Unicode Transformation Format, 8-bit) je standard kódování znaků Unicode, který představuje délku kódování, kde znaky zabírají samostatný počet bajtů, což pomůže při zpracování textových dat.

Toto je metaznačka kódování stránky, aby prohlížeč pochopil, jaké kódování má použít. Atribut určuje kódování znaků použité na stránce. "UTF-8" znamená Unicode Transformation Format s osmi bity, který podporuje širokou škálu znaků a jazyků. Na webové stránce je to důležité pro správné zobrazení textu, zejména při použití různých jazyků a znaků, které nelze reprezentovat ve standardním kódování ASCII.

- **<meta name="viewport" content="width=device-width, initial-scale=1.0">**

Tato metaznačka je důležitá zejména při vytváření responzivních webových stránek, zejména pro zobrazení na různých koncových zařízeních. Pomáhá předcházet problémům se zvětšením obsahu a zajišťuje lepší čitelnost a uživatelský komfort při práci s webovou stránkou.

- **<meta http-equiv="X-UA-Compatible" content="ie=edge">**

Metaznačka označuje kompatibilitu s prohlížečem Internet Explorer. Jako příklad může uvádět, že pro správnou funkci webu je nutné dosáhnout kompatibility s co největším počtem prohlížečů obecně. Dříve byla jednou ze standardních metod optimalizace, nyní se nepoužívá.

- **<meta name="description" content="Popis stránky">**

Tato metaznačka byla použíta ke zlepšení výkonu prohlížeče a webových stránek z hlediska SEO a vyhledávačů. Vyhledávač tak najde stránku rychleji, což ovlivní uživatelský zážitek a propagaci stránky.

- **<meta name="robots" content="index, follow">**

Metaznačka s podmínkami pro vyhledávací roboty, aby stránku rychle našli a zaindexovali.

- **<meta http-equiv="refresh" content="time_in_seconds;url=new_URL">**

Automatické přesměrování je často užitečné k obnovení stránky nebo k obejití některých problémů s načítáním původní webové stránky, což pomůže zajistit přesný a bezproblémový provoz webové aplikace. Z jiného pohledu může způsobit znepokojení uživatele, protože přesměrování na jinou stránku při čekání může mít nepříznivý vliv na uživatelský zážitek.

Pomocí výše uvedených metaznaček byla provedena práce na zlepšení rychlosti zobrazování obsahu na webové stránce s cílem dosáhnout vysokého výkonu a otestovat metody implementace akcelerace prohlížeče. Kromě standardních metaznaček byly přidány speciální tagy Facebooku pro lepší interpretaci stránky v jiných zdrojích. To pomůže zlepšit zobrazitelnost a poskytne více informací o webové stránce.

```html
<head>
  <title>@@title</title>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <meta
    name="description"
    content="True place for buying your favourite games"
  />
  <meta name="robots" content="index, follow" />
  <meta
    http-equiv="refresh"
    content="3000;https://salmix1326.github.io/portfolio/dist/index.html"
  />
  <meta
    property="og:url"
    content="https://salmix1326.github.io/portfolio/dist/index.html"
  />
  <meta property="og:type" content="page" />
  <meta property="og:title" content="The right place for your dreams" />
  <meta
    property="og:description"
    content="What games are the most playable?"
  />
  <meta
    property="og:image"
    content="https://picsum.photos/200/300"
  />
  <link rel="stylesheet" href="css/style.min.css" />
  <link rel="icon" type="image/x-icon" href="@img/logo-icon.png" />
</head>
```

Obrázek 21. Přidání speciálních meta značek pro zlepšení výkonu webové stránky. Zdroj: Vlastní zpracování

Obrázek 22. Zpráva o analýze stránky při použití metaznaček od PageSpeed Insights. Zdroj: Vlastní zpracování, PageSpeed Insights

Z výše uvedené analýzy (obr. 22) je patrné, že došlo k výraznému zlepšení postupů SEO a metrik při analýze webových stránek. Skóre SEO má 100% úspěšnost, což pomůže vyhledávačům velmi rychle najít a zpracovat webovou stránku.

4.2.4 Média dotazování

Mediá dotazy hrají důležitou roli při optimalizaci načítání webové aplikace a mají pozitivní dopad na uživatelský zážitek. Vytvoření responzivního návrhu ovlivňuje strukturu zobrazení webové stránky a umožňuje nastavit pravidla pro zpracování dat pro různá koncová zařízení.

Načítání pouze určitých stylů umožní prohlížeči ušetřit čas při zobrazování prvků, což bude mít pozitivní dopad na dobu první interakce s uživatelem.

Správným nastavením použití stylů pro různá zařízení dojde k výraznému zlepšení přenosu dat na server.V tomto příkladu byla použita řada mediálních dotazů pro optimalizaci zobrazení webové aplikace, což mělo pozitivní vliv na rychlost zobrazení webové stránky, zlepšení metrik načítání a analýzu výkonu webové stránky pomocí vyhledávacích robotů pro co nejrychlejší indexaci a nalezení stránky na internetu. Ve většině případů to mělo významný dopad na metriky SEO.

```
@media (min-width: 162.5em) {
  .intro {
    height: 62.5rem;
  }
}

@media (max-width: 20em) {
  .main-page-section-sellers {
    margin-top: 0.9375rem;
  }

  .main-page-section-sellers {
    margin-bottom: 0.9375rem;
  }
```

Obrázek 23. Použití media dotazů pro sekce. Zdroj: Vlastní zpracování

V tomto příkladu(obr.23) byly v sekci sellers použity media dotazy, které mění strukturu kódu v reálném čase. To pomohlo správně vykreslit obsah webové stránky a zlepšit metriky rychlosti zobrazení uživatelského rozhraní.

4.2.5 Použití nástrojů Gulp a Webpack k optimalizaci souborů

Pomocí automatizačního nástroje, jako je Gulp, lze podobně opakující se úlohy, jako je komprese obrázků, seřadit. Gulp převezme provádění těchto úloh podle potřeby/plánu. Tyto úlohy se tak stanou automatizovanými, což ušetří spoustu času a sníží počet chyb.
Typická úloha Gulp se provádí ve třech krocích:

1. Načtení zdrojových souborů.
2. Provádění jednoho nebo více procesů nad soubory.
3. Zápis konečného výstupu do nového cílového adresáře.

Gulp umožní mnohem více se soustředit na kód, místo konfigurační části automatizace. Celé rozhraní Gulp API se skládá pouze ze čtyř funkcí, tj. gulp.src, gulp.dest, gulp.task a gulp.watch. Vše, co je potřeba ve projektu udělat, závisí na těchto čtyřech metodách. Pro Gulp je k dispozici široká škála zásuvných modulů.

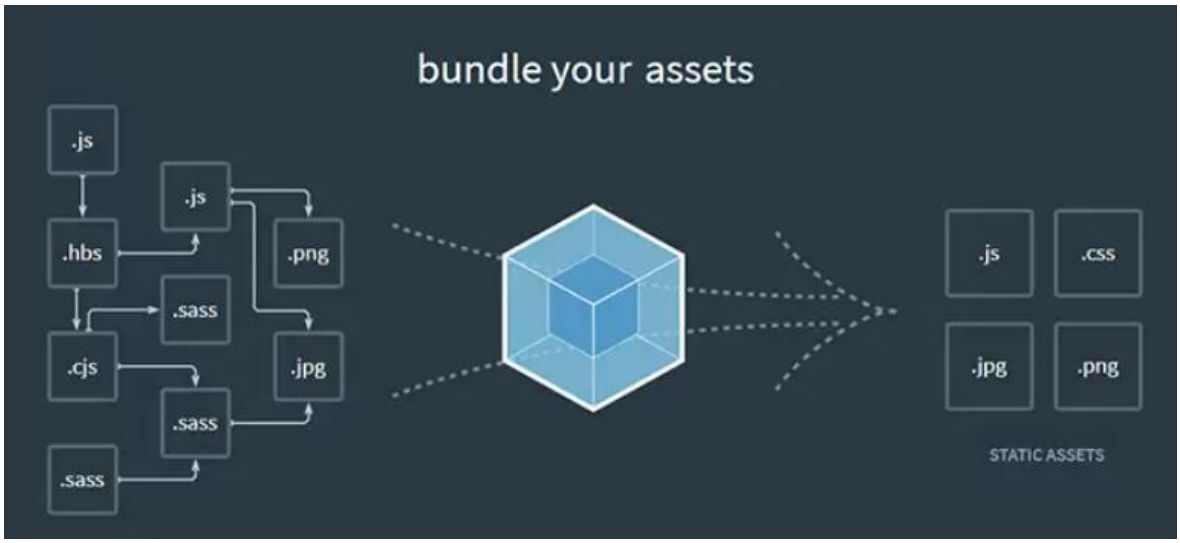
Webpack je nástroj pro sdružování modulů, který pomáhá plnit dva hlavní úkoly: Uspořádá všechny soubory JavaScriptu ve správném pořadí. To znamená, že zjistí, jak seřadit závislosti mezi soubory.

Sbalí je do jediného zdrojového souboru .js. Tento soubor pak může být přímo spuštěn při načítání webové stránky. Odpadá tak nutnost odesílat opakované požadavky HTTP na server. V době, kdy programování v jazyce JavaScript nebylo příliš rozšířené, byla správa kódu mnohem jednodušší. Většina aplikací měla buď jen několik řádků skriptů JS, nebo byla založena na kolekci menšího počtu souborů. Načítání a spouštění aplikace tak bylo poměrně jednoduché a rychlé.

V dnešní době, kdy se tolik skriptů nachází v mnoha samostatných souborech jednoho projektu, je obtížné je sledovat. Při načítání projektu se mohou všechny soubory načítat chaoticky. Pokud například soubor A obsahuje kód, který závisí na souboru B, ale soubor B se načte jako první, pak to vyvolá chyby "závislostí", protože by některé požadované moduly chyběly. Zde se projevují výhody balíčku Webpack.

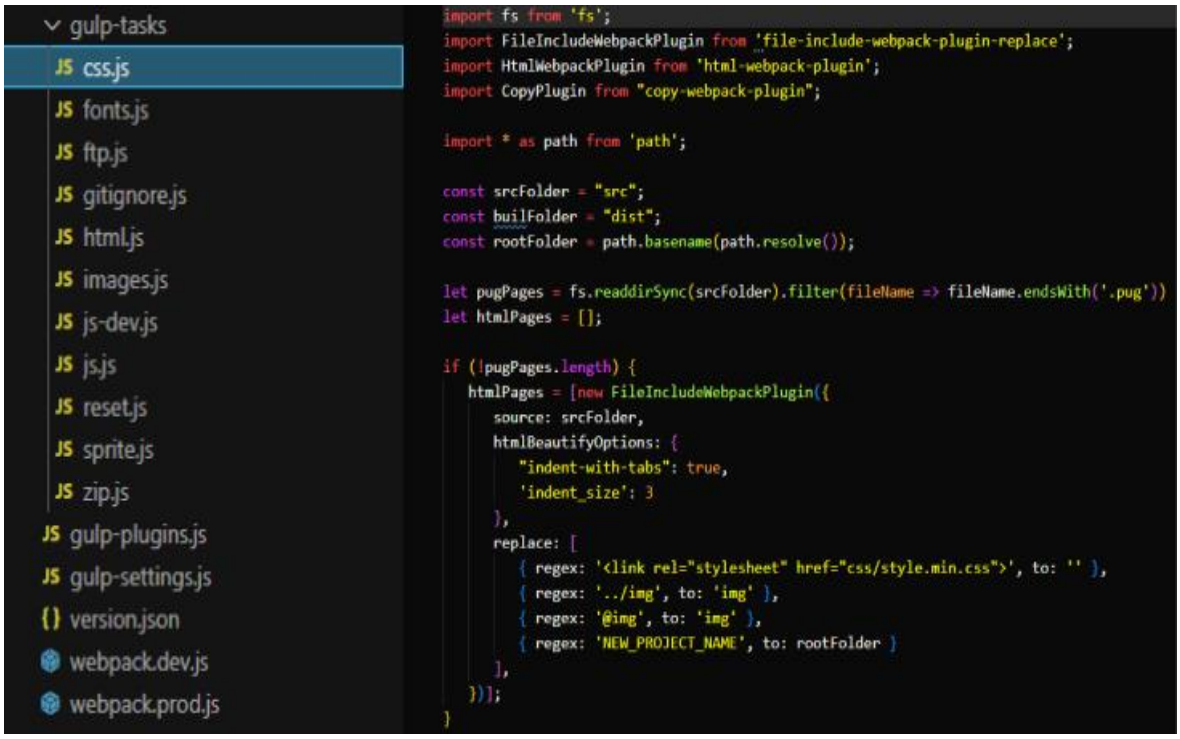
Když se webová stránka načítá v prohlížeči, vyžádá si množství souborů bez určitého pořadí a načte je všechny najednou. Webpack tyto soubory seřadí ve správném pořadí, aby nedošlo ke zmatku, a načte je podle potřeby. Spojení určitých modulů dohromady tak eliminuje velkou zátěž a zmatek způsobený asynchronními požadavky.Webpack také odstraňuje mrtvé prostředky a kód. To znamená, že detekuje a odstraní ty části kódu, které se nepoužívájí.

Podporuje živé i průběžné načítání. To první znamená, že lze v kódu provést libovolné množství změn a celý projekt se znovu načte, aby se změny okamžitě projevily na webové stránce v prohlížeči. V případě horkého načítání se tak stane, aniž by se musela znovu načíst celá aplikace, ale pouze ty soubory, které prošly nějakými změnami.



Obrázek 24. Příklad fungování nástroje Webpack. Zdroj: https://dev.to/hereserin/an-intro-to-webpack-l2n

Stejně jako u každé technologie, ani u WebPacku není vše dokonalé. Složitost konfigurace WebPacku je asi jeho největší nevýhodou. Nastavení nové instance WebPacku zabere hodně času a zabere hodně úsilí. S rostoucí velikostí může být kód pro Webpack značně nepřehledný - nebo přímo složitý.



Obrázek 25. Použití nástrojů Gulp a Webpack k optimalizaci struktury souborů. Zdroj: Vlastní zpracování

V této výzkumné práci byly tyto nástroje(obr. 25) použity k optimalizaci načítání souborů s kódem do prohlížeče a ke zrychlení zpracování kódu. Pomocí těchto pomocných nastavení byla získána optimální struktura obsahu souborů, což pomohlo dosáhnout vhodných metrik při analýze rychlosti načítání webové stránky.

4.2.6 Optimalizace obrázků

Obrázky hrají velmi důležitou roli při optimalizaci a zrychlování výkonu webových stránek. Jde o jeden z nejdůležitějších aspektů zrychlení výkonu a uživatelského komfortu. V této části studie byly vylepšeny funkce FCP (First Contentful Paint) a FID (First Input Delay), aby bylo možné vytvořit rychlou odezvu a pohodlné používání webové stránky.

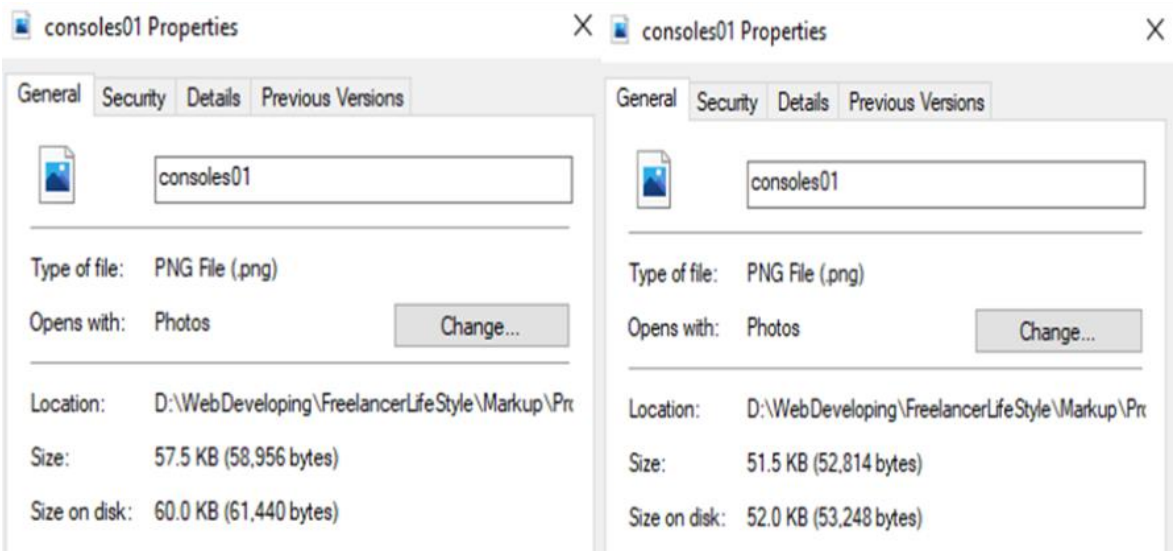
Jednou ze zásad práce s obrázky je volba formátu, který se má použít. Mezi mnoha formáty, jako jsou JPEG, PNG, RAW, GIF a další, je třeba jasně oddělit účel jejich použití na webu.

Existuje mnoho nástrojů pro kompresi obrázků. V této praci byly použity pluginy, jako jsou Gulp a Webpack. Pro automatickou optimalizaci obrázků lze také použít pluginy nebo skripty na serveru.

```
bpweb > config > gulp-tasks > JS images.js > [e] images
1   import webp from "gulp-webp";
2   import imagemin from "gulp-imagemin";
3
4   export const images = () => {
5       return app.gulp.src(app.path.src.images)
6           .pipe(app.plugins.plumber(
7               app.plugins.notify.onError({
8                   title: "IMAGES",
9                   message: "Error: <%= error.message %>"
10              }))
11          )
12          .pipe(app.plugins.newer(app.path.build.images))
13          .pipe(
14              app.plugins.if(
15                  app.isWebP,
16                  webp()
17              )
18          )
```

Obrázek 26. Konfigurace úkolu Gulp pro kompresi obrázků. Zdroj: Vlastní zpracování

Nastavení komprese pomocí nástroje Gulp (obr.26) pro optimalizaci obrázků. Pro nastavení byl použit programovací jazyk Javascript. To pomohlo dosáhnout výsledku komprese obrázků na minimální velikost bez ztráty kvality původního obrázku.



Obrázek 27. Srovnání obrázku před a po kompresi. Zdroj: Vlastní zpracování

Gulp poskytl možnost kompilovat soubory. Jedná se o rozhraní API pro spouštění úkolů a kompresi obrázků. Webpack pomohl vytvořit vlastní sestavení souborů pomocí modulárních aplikací a optimalizace zdrojů. Pomohl vytvořit spojení mezi jazyky Javascript a CSS. V tomto případě byly sestaveny všechny soubory CSS (SCSS) s vyčištěním nepotřebného kódu a kompresí obrázků.

Po použití technik pro kompresi obrázků, sestavování souborů a optimalizaci ukládání dat bylo dosaženo vysokého výkonu webové aplikace . Důležité je brát v úvahu, že tyto ukazatele se mohou lišit v závislosti na výkonnosti koncových zařízení uživatelů, protože různá zařízení mají odlišné parametry pro zpracování dat.

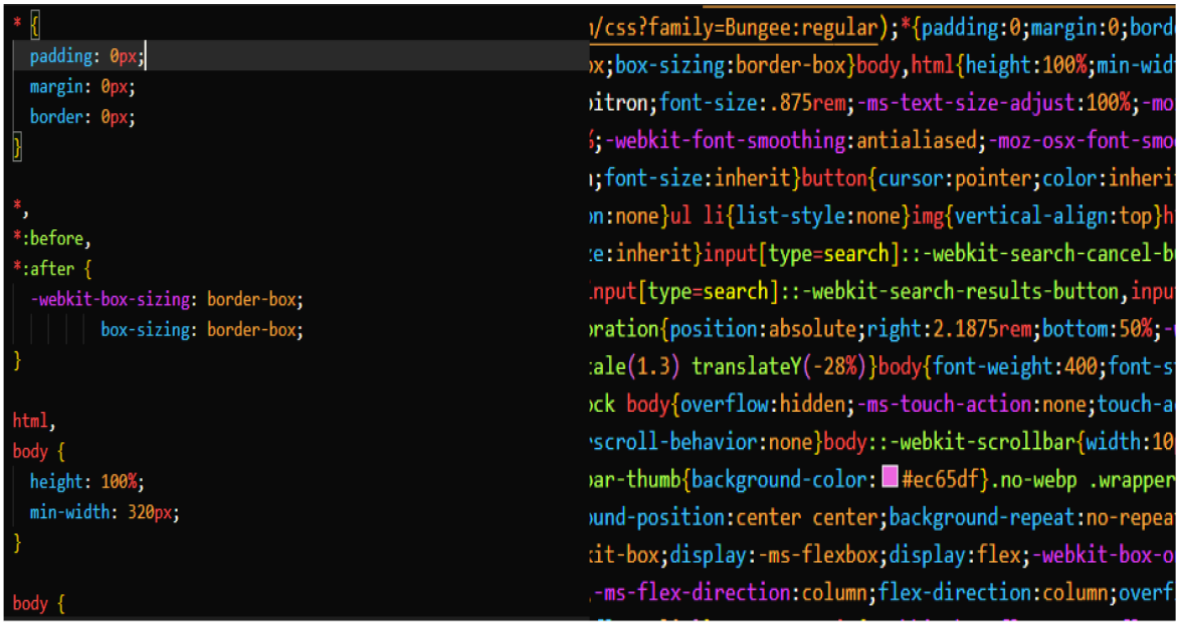
4.2.7 Komprese souborů CSS

Zpracování souborů CSS hraje při optimalizaci načítání důležitou roli. Existuje řada technik a nástrojů, které umožňují snížit hmotnost souborů a optimalizovat kód tak, aby bylo dosaženo vysokého výkonu a rychlosti nahrávání kódu CSS do prohlížeče.

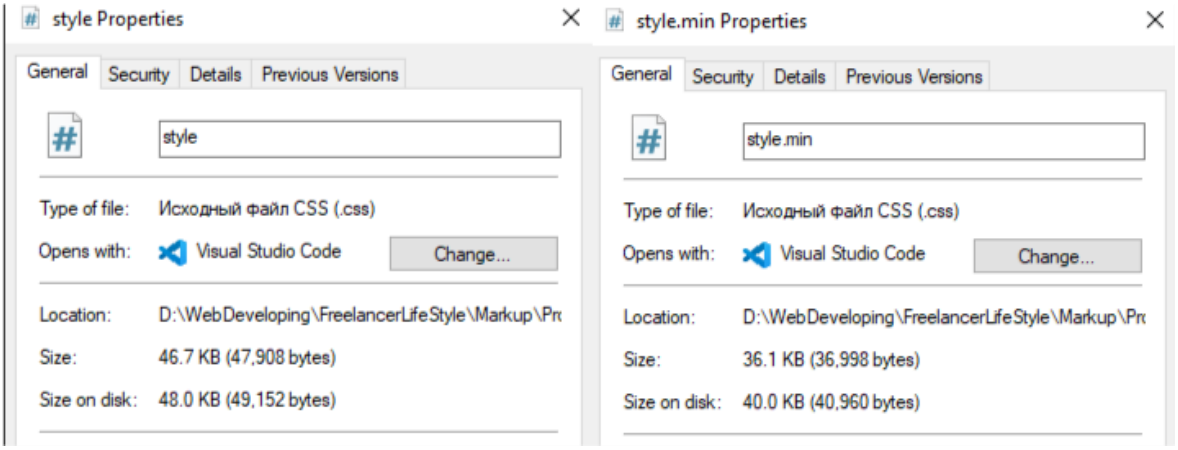
Optimalizační techniky umožňují minifikovat soubory, odstranit nepotřebný kód a mezery, které zabírají místo a nejsou pro použití v prohlížeči nijak potřebné. Tím se zlepší interakce mezi klientem a serverem, což je užitečné zejména v případě, že má uživatel nízké připojení k internetu.

Zmenšení velikosti souboru má také za následek zlepšení přenosu dat po síti, což je důležité zejména tehdy, když je přenos dat po síťovém protokolu nestabilní. Pokud je tento kód správně optimalizován, lze zvýšit rychlost vykreslování webové stránky, což bude mít pozitivní vliv na FID, FTP a TTI.

V této studii byly k optimalizaci souborů použity nástroje jako Gulp a Webpack, které shromáždily všechny soubory CSS do jednoho společného a následně je minifikovaly, odstranily mezery, nepotřebný kód a změnily strukturu syntaxe.



Obrázek 28. Soubor CSS před a po minifikace. Zdroj: Vlastní zpracování



Obrázek 29. Výsledek minifikace CSS souboru. Zdroj: Vlastní zpracování

Jak je vidět(obr. 29), minimalizace souboru CSS přinesla řadu vylepšení, včetně snížení hmotnosti souboru, což prohlížeči pomůže soubor zpracovat rychleji. Zároveň shromáždění všech souborů do jednoho pomohlo dosáhnout vysokého přenosu dat na server. Obecně je komprese souborů s kódem nezbytná pro správné fungování webové aplikace a měla by se používat ke zlepšení metrik načítání.

4.2.8 Optimalizace prvků stránky pomocí Javascriptu

JavaScript je běžným jazykem při vývoji mobilních a webových aplikací. Optimalizace JavaScriptu je díky své popularitě stále více nezbytná pro zlepšení výkonu aplikací. Soubory JavaScriptu jsou životně důležité aspekty procesu webové aplikace, ale rychlost webu a uživatelská zkušenost jsou pro úspěch webu rozhodující. Je tedy důležité optimalizovat soubory JavaScriptu, aby byl zajištěn bezproblémový výkon. Optimalizace souborů JavaScriptu řeší problém blokování vykreslování, doby načítání stránky, velké velikosti souboru a podobně.

Spuštění skriptu. Soubory JavaScriptu obsahující blokující vykreslování stránky se zpožděním kódu. Spuštění skriptu brání načítání jiného obsahu, což má za následek špatnou uživatelskou zkušenost. Velké soubory JavaScriptu se stahují déle, což má vliv na dobu načítání stránky. Špatně optimalizovaný kód JavaScriptu – jako jsou nadměrné smyčky, nadbytečné výpočty nebo neefektivní algoritmy – vede k blokům výkonu.

```
import * as flsFunctions from "./files/functions.js";

flsFunctions.isWebp();
flsFunctions.addTouchClass();
flsFunctions.menuInit();
flsFunctions.fullVHfix();
import "./libs/dynamic_adapt.js";
```

Obrázek 30. Přidání funkcí Javascript na stránku. Zdroj: Vlastní zpracování

V této praxi byly použity následující optimalizační metody:

1. Přidána funkce **isWebp**, která pomůže optimalizovat obrázky jejich převodem na příponu .webp. Tento formát má lepší zpracování obrazu a zabírá méně místa v souborech webových stránek.
2. Funkce **addTouchClass** umožňuje přizpůsobit obrazovku prohlížeče mobilním zařízením a odstranit zbytečné prvky, které narušují pohodlnou práci na webové stránce.
3. Funkce **fullVHfix** přidává funkcionalitu speciálně pro mobilní zařízení, kde je struktura zobrazení obsahu webové stránky odlišná
4. Na webovou stránku byla přidána **dynamická adaptace**. Pomocí této metody se struktura změní v reálném čase a přeuspořádá prvky stránky tak, aby odpovídaly určité velikosti okna prohlížeče. Díky této metodě není potřeba používat několik možností struktury pro každé mobilní zařízení. Tím se výrazně sníží velikost kódu a hmotnost souboru. To přímo ovlivňuje optimalizaci načítání webové aplikace, pomáhá prohlížeči zobrazovat obsah co nejrychleji a přenášet data na server.

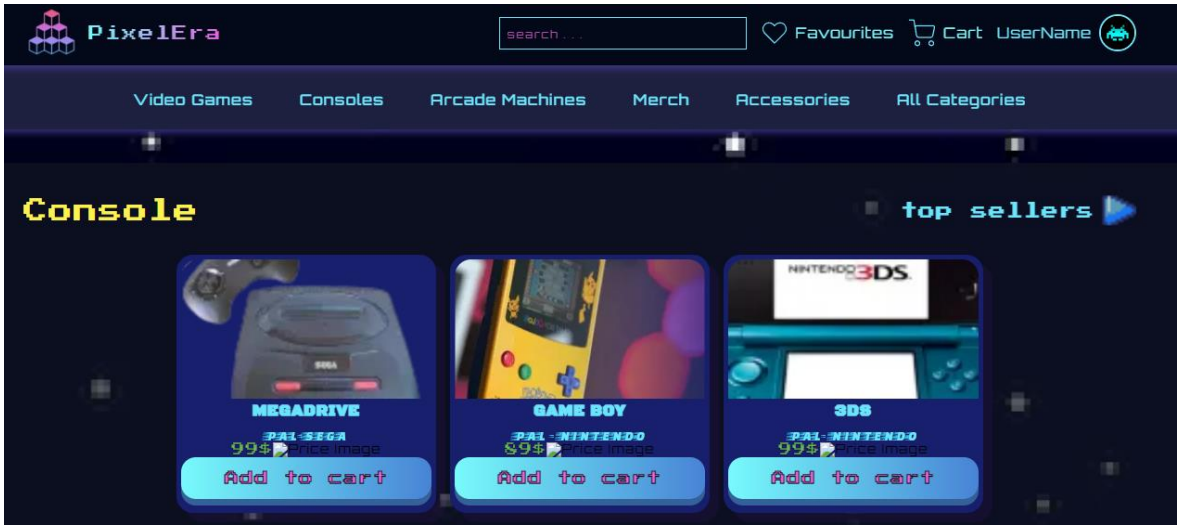
```
let isMobile = {
    Android: function() {
        return navigator.userAgent.match(/Android/i);
    },
    BlackBerry: function() {
        return navigator.userAgent.match(/BlackBerry/i);
    },
    iOS: function() {
        return navigator.userAgent.match(/iPhone|iPad|iPod/i);
    },
    Opera: function() {
        return navigator.userAgent.match(/Opera Mini/i);
    },
    Windows: function() {
        return navigator.userAgent.match(/IEMobile/i);
    },
    any: function() {
        return isMobile.Android() || isMobile.BlackBerry() || isMobile.iOS() || isMobile.Opera() || isMobile.Windows();
    }
};
```

Obrázek 31. Příklad práce s optimalizací obrázků pomocí Javascriptu. Zdroj: Vlastní zpracování

Na tomto příkladu(obr.31) byly použity funkce pro kompresi a převod obrázků do formátu webp, které výrazně snížily dobu zpracování při zobrazování prvků webu. Tento formát umožňuje načítat obrázky předem a umožňuje co nejrychlejší práci na stránce. Při použití této metody bylo dosaženo vysokých metrik FID a TTI.

5 Výsledky a diskuse

5.1 Výsledek vytváření testovací webové stránky



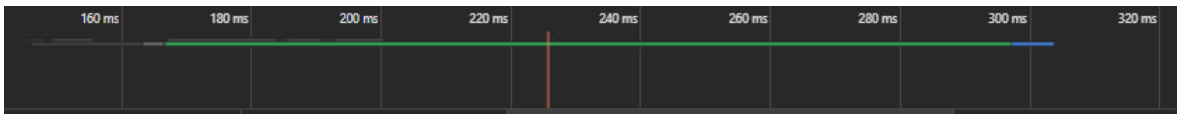
Obrázek 32. Testovací webová stránka. Zdroj: Vlastní zpracování

Výsledek uplatnění znalostí v oblasti metod, technik a nástrojů pro tvorbu webových stránek(obr 32). Pomocí struktury a kódování souborů byla vytvořena testovací webová stránka k další analýze jejího výkonu a zvýšení metrik načítání. Při analýze indikátorů optimalizace výkonu pro webovou aplikaci byla opakovaně měněna struktura a přidána řada dalších postupů pro měření ukazatelů jejího chování v prohlížeči.

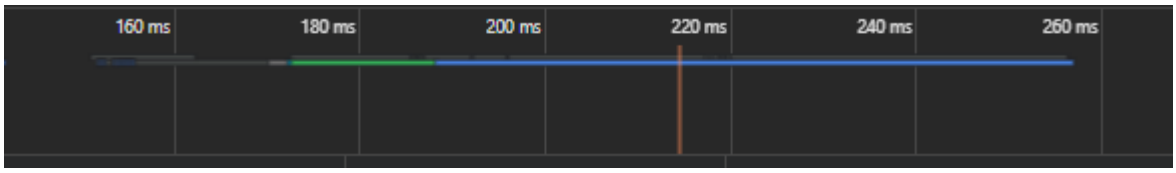
5.2 Rychlost načítání stránky a využití paměti prohlížeče

Výsledkem této výzkumné práce bylo zlepšení výkonu webové stránky. Ke struktuře webu a jeho jednotlivých prvků byly použity různé metody a techniky. Praktická část byla věnována podrobnému zkoumání dopadu optimalizace načítání souborů stránek na využití paměti prohlížeče, počítače uživatele a nastavení kvalitního připojení k serveru.

Výsledky využití paměti prohlížeče a dopad zpracování jednotlivých prvků, které přímo ovlivňují hlavní metriky výkonu webu. Byl analyzován vztah mezi kvalitou zpracování obrázků a textů ve webové aplikaci.

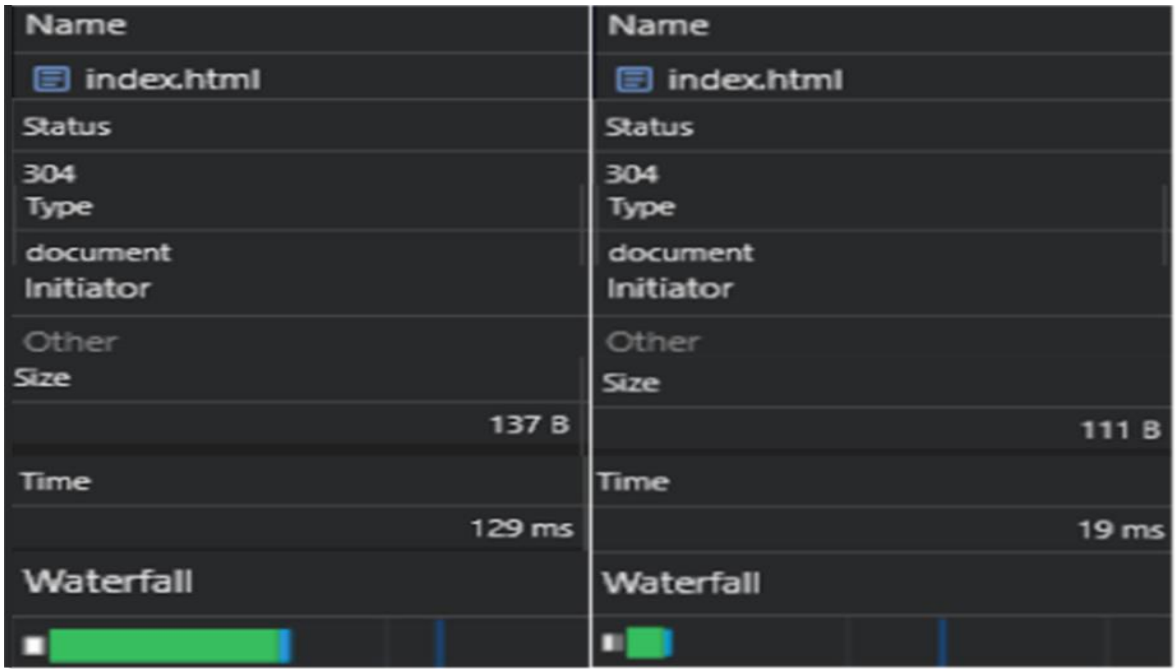


Obrázek 33. Rychlost načítání webové stránky (obrázek A). Zdroj: Vlastní zpracování



Obrázek 34. Rychlost načítání webové stránky (obrázek B). Zdroj: Vlastní zpracování

Tyto obrázky ilustrují rychlost úplného zobrazení prvků na stránce, přičemž při použití ovládání prvků webu pomocí jazyka Javascript bylo dosaženo následujících změn v rychlosti načítání webu. Při nesprávném přístupu k nastavení načítání bylo dosaženo hodnot 310 milisekund od začátku přechodu na stránku. Po zlepšení výkonu bylo dosaženo výsledku 260 milisekund od plného zpracování aplikace. Obrázek A byl výsledkem bez optimalizace prvků. Na obrázku B je výsledek s použitou metodou komprese obrázků na webu. Praxe ukazuje, že čím menší jsou hodnoty Largest Contentful Paint v milisekundách, tím lepší je obecně výkon stránky.

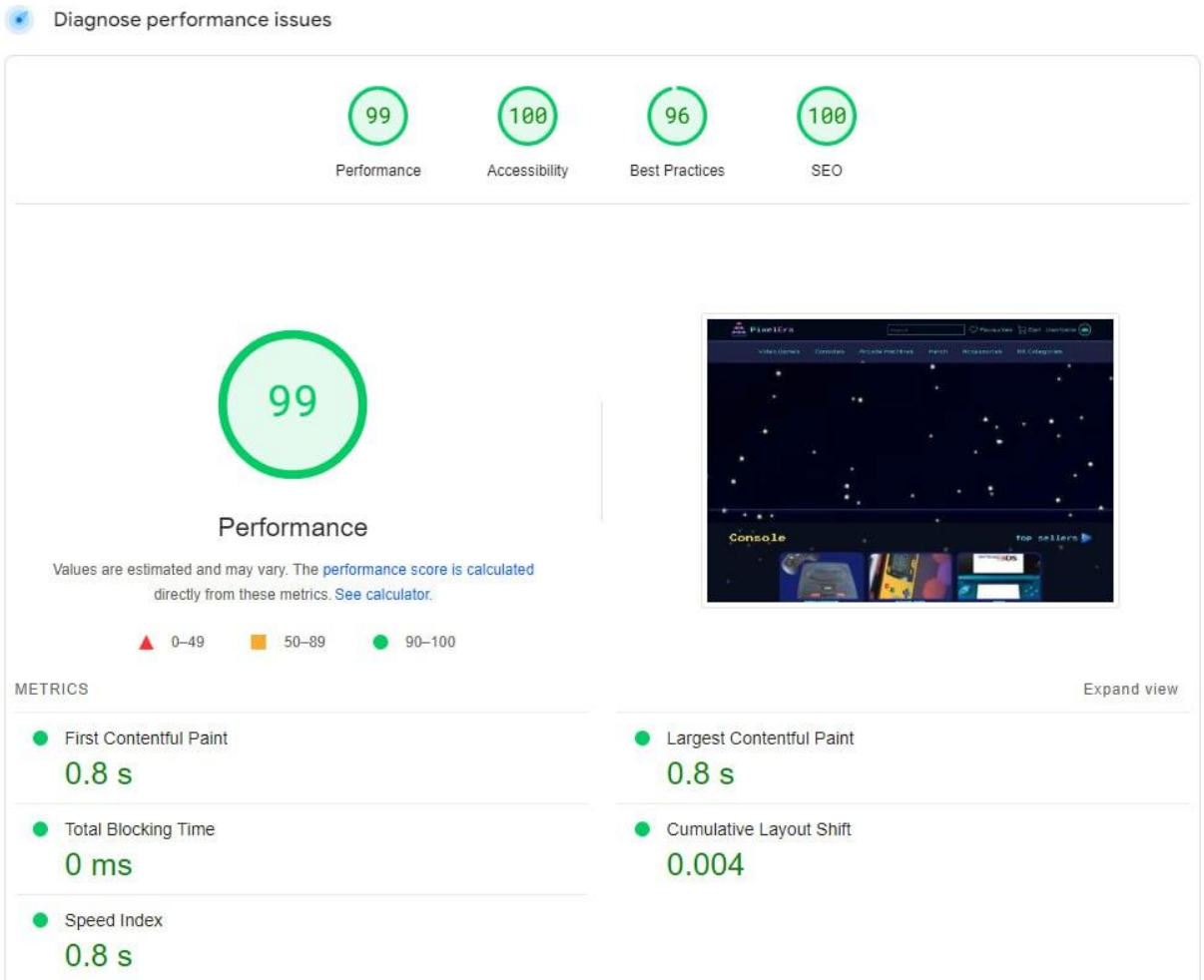


Obrázek 35. Využití paměti prohlížeče před a po optimalizaci. Zdroj: Vlastní zpracování

5.3 Zlepšení struktury kódu HTML a CSS

Jedním z nejdůležitějších aspektů zvyšování výkonu webové aplikace je správná struktura kódu HTML a CSS. Ta výrazně pomáhá optimalizovat využití paměti a manipulaci se zdroji ze strany prohlížeče.

Použitím speciálních technik psaní kódu byl nalezen způsob, jak zlepšit metriky webu z hlediska výkonu. Na značky a atributy byla aplikována pravidla a byly přidány speciální metaznačky, které webovou aplikaci vynesou na přední místa ve výsledcích vyhledávání, když ji zpracovávají vyhledávací roboti.

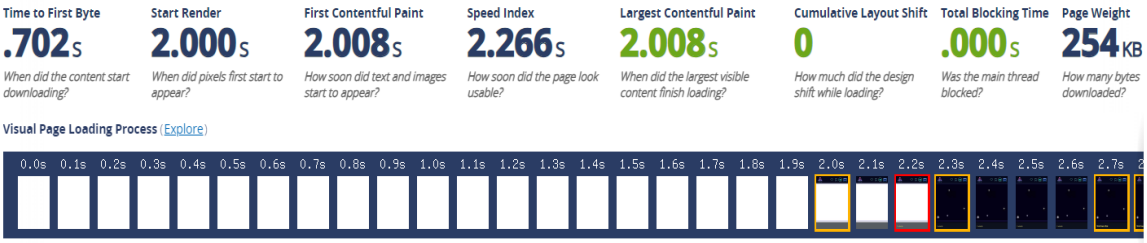


Obrázek 36. Výsledky analýzy metrik webové stránky. Zdroj: Vlastní zpracování, PageSpeed Insights

Tento obrázek ukazuje(obr.36), jak se metriky webu zlepšily po použití všech technik optimalizace výkonu, včetně zápisu správné struktury kódu. Metriky Perfomance (výkon) a Accessibility (přístupnost) ukazují rychlé načítání a vysoké hodnoty FCP, FID a TTI.

5.4 Výsledky zlepšení výkonu webu pomocí Javascriptu

Po získaných výsledcích analýzy výkonnosti webové stránky s využitím různých metod zápisu funkcí v programovacím jazyce Javascript, které významně přispěly k optimalizaci struktury kódu a upravily prvky webové stránky, lze konstatovat, že metriky se v porovnání načítání webové stránky bez použití výše zmíněných funkcí popsaných v praktické části práce výrazně zvýšily.



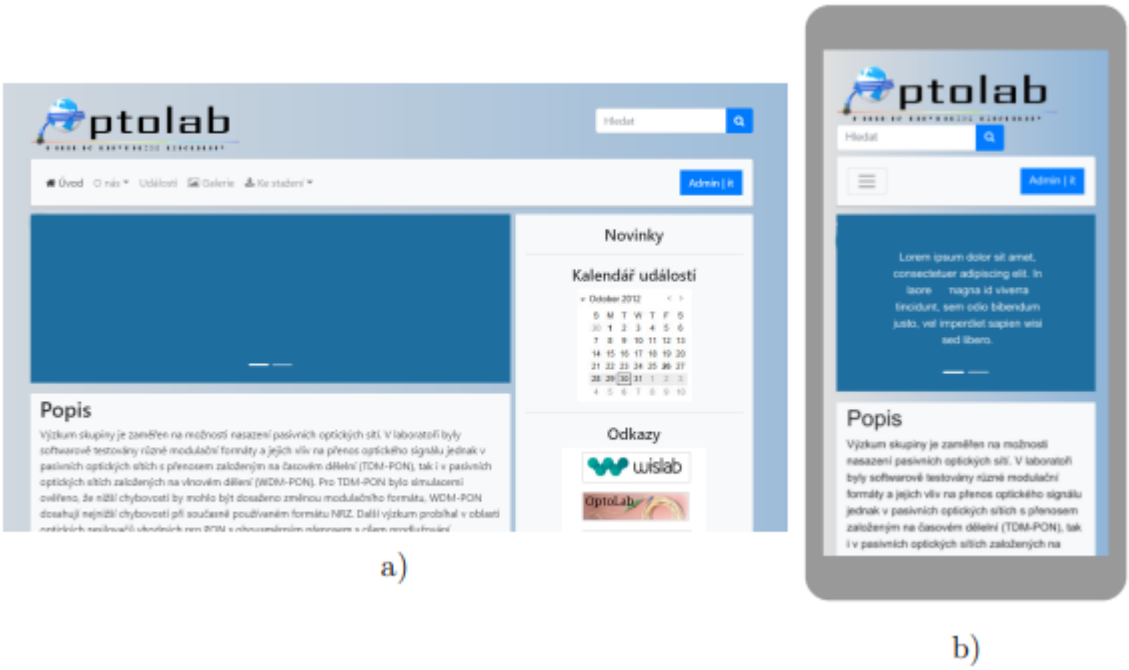
Obrázek 37. Výsledek analýzy načítání FID a LCP. Zdroj: Vlastní zpracování, Webpage Test

Tento obrázek(obr.37) ukazuje výrazné zlepšení rychlosti načítání a doby do první interakce se stránkou. V porovnání s výsledky bez technik optimalizace struktury kódu a prvků stránky jsou tyto metriky téměř na nejlepších hodnotách.

5.5 Srovnání použití optimalizačních technik s výzkumem jiných autorů

Uvedené výzkumy této práci se nedotýkají použití frameworků pro psaní struktury kódu, ale za pozornost stojí například výzkum s použitím frameworku Bootstrap, kde autor dosáhl zlepšení zobrazení prvků stránky:

„Jednotlivé návrhy již byly vytvořeny pomocí HTML a CSS kódu s podporou knihovny Bootstrapu, díky které se přizpůsobovaly velikosti zobrazovacího zařízení. Stránky jsou tvořeny pomocí předem nadefinovaných tříd objektů z knihovny Bootstrap, které jsou následně upraveny dle potřeby. Díky využití těchto tříd jsou veškeré komponenty nadefinovány na několik různých šíří daného prohlížeče.“[ZÁMEČNÍK, s. 32, VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, 2018][12]



Obrázek 38. Návrh první pro a) desktopovou, b) mobilní verzi. [ZÁMEČNÍK, 2018, s32]
Zdroj:https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=175710

Autor dosáhl úpravy struktury stránky pomocí knihovny, která pomohla dosáhnout změn zobrazení v reálném čase.[12]

Další autor uvádí výzkum vlivu technologií CSS a Javascript na zobrazení struktury webových stránek, kde zmiňuje špatnou optimalizaci školních stránek se zhoršením zobrazení prvků bez použití výše zmíněných webových front-end technologií:

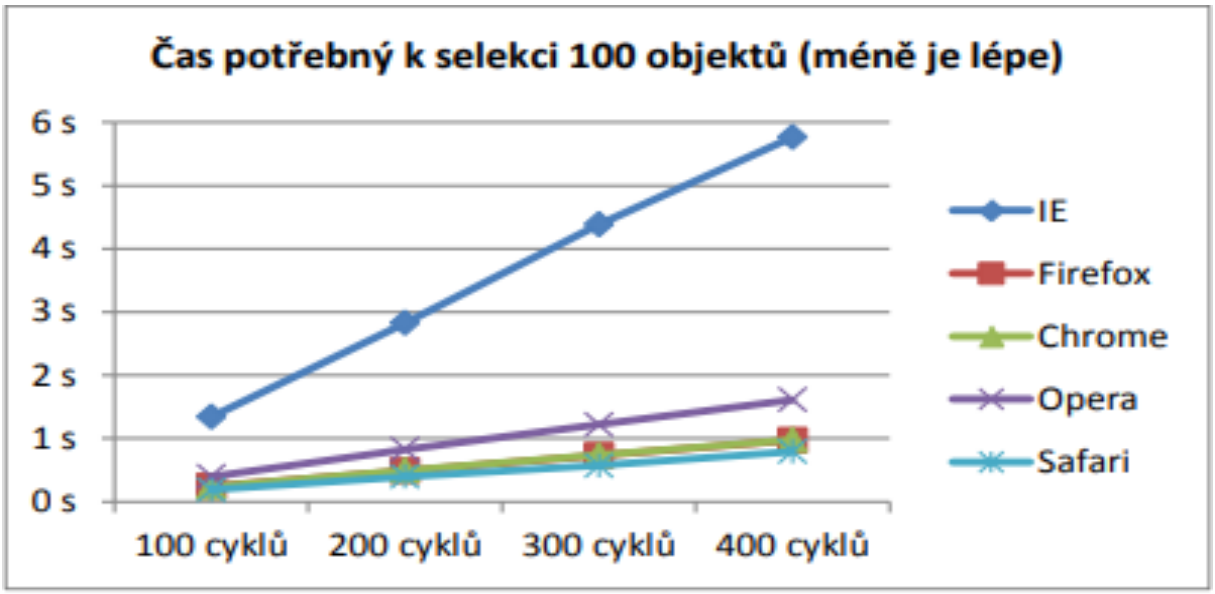
„Vypnuté CSS styly odhalily zvláště chyby v rozvržení webových stránek. Navigační panely se u mnoha škol objevily již na začátku stránky, takže nevidomému uživateli hlasová čtečka nejprve přečte kvantum informací z různých menu a až poté následuje obsah s informacemi o školském zařízení. V tomto testování se objevily u testovaných subjektů bariéry, které zhoršují přehlednost na stránce.“[NEVRLÝ, s. 41, UNIVERZITA PALACKÉHO V OLOMOUCI, 2012][13]

Srovnání práce lze provést s autorem, který použil knihovnu jQuery založenou na programovacím jazyce Javascript k optimalizaci výkonu webové stránky s testováním výkonu v různých prohlížečích. K ovládání webové aplikace prostřednictvím samotné knihovny použil takové selektory, jako jsou objekty DOM, AJAX (asynchronní JavaScript a XML) a ASP.NET:

„Vytvořenou stránku s veškerým testovaným kódem nalezneme na CD (podkapitola 6.2) v dokumentech SelectionTest.htm a Scripts/selection.js Máme 2 testy, oba v neoptimální i optimalizované podobě. Všechny tyto testy spustíme postupně 10x na všech prohlížečích s hodnotami 100, 200, 300 a 400 cyklů. V každém jednotlivém cyklu testu se vybraný DOM objekt vyprázdní příkazem empty() a zase naplní hodnotou „addedDiv“.

V prvním testu testujeme rozdíl mezi výběrem dceřiných objektů pomocí selektorů „#myId2>“ a příkazem children(). Podle podsekce 2.4.2.1 by měl být příkaz children() rychlejší.*

V druhém testu testujeme rozdíl mezi výběrem třídy pouze pomocí $(“.className”) a $(“#myId2 .className”). Dle podsekce 2.4.2.3 by měl být druhý uvedený příklad o poznání rychlejší. V obou optimalizovaných testech je ještě použita optimalizace ukládáním domezipaměti. Ve všech testech vybíráme 100 objektů.“[KUTIL, s. 20, MASARYKOVA UNIVERZITA, 2011][14]



Obrázek 39. Srovnání výkonu prohlížeců. [KUTIL, s. 20, MASARYKOVA UNIVERZITA, 2011] Zdroj: https://is.muni.cz/th/ekc3g/bc20111230print.pdf

6 Závěr

Výsledkem této bakalářské práce bylo prozkoumání teoretických poznatků z oblasti front-end technologií a jejich další aplikace v praktické části pro splnění cílů optimalizace načítání webové aplikace. Na základě syntézy teoretických a praktických poznatků byl zkoumán vliv technik a metod pro optimalizaci rychlosti a zvýšení výkonu webové aplikace změnou struktury kódu a správy souborů testované webové stránky.

Byly získány data z analýzy metrik výkonnosti po aplikaci pravidel pro strukturování kódu značkovacího jazyka HTML a kaskádových stylů CSS. Tím se podařilo vyřešit problém s nesprávným a dlouhým zobrazováním prvků webové stránky a dosáhnout změn struktury obsahu v reálném čase, což kromě vyřešení výkonnostních problémů pomohlo zlepšit uživatelský zážitek na stránce. Kromě základní struktury kódu byly přidány určité metaznačky, které pomohly zlepšit interakci webové aplikace s vyhledávacími roboty, což přispělo k vysokému vnějšímu výkonu stránky na internetu.

Jedním z hlavních přínosů v oblasti analýzy výkonu bylo použití programovacího jazyka Javascript. Právě pomocí tohoto nástroje bylo po uplatnění technik práce s kódem a soubory webové stránky, mezi něž patřila komprese obrázků, minifikace souborů CSS a přidání funkcí, které mění zobrazení prvků na webové stránce, dosaženo výrazného zlepšení v oblasti snížení spotřeby paměti prohlížeče a rychlosti přenosu dat na server. To pomohlo prokázat nezbytný vliv Javascriptu na optimalizaci výkonu webové stránky jako celku.

Splněním jednoho z hlavních cílů této práce bylo dosažení kladných metrik FID, TTI a LCP, o kterých bylo řečeno v teoretické části. Tyto metriky přímo ovlivňují výkonnost stránky jako celku.

Z tohoto závěru lze odvodit následné příležitosti pro další výzkum. Práce byla provedena bez použití různých frameworků a programů pro správu a úpravu kódu, což umožňuje analyzovat fungování webu pomocí výše uvedených nástrojů a poskytuje možnost zkoumat chování webové stránky pomocí detailnějších technik řízení souborů. Tyto techniky a metody umožňují vytvářet weby a aplikace s vysokou optimalizací a výkonem. Prolínání značkovacích a programovacích jazyků spolu úzce souvisí, což je důležité vzít v úvahu při plánování tvorby kvalitních webových stránek.

7 Seznam použitých zdrojů

1. DAVIDMILLS, Chris a Andrew PFEIFFER, Populating the page: how browsers work MDN Web Docs [online], 2023, 2023, 1 [cit.2023-11-10]. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/Performance/Fundamentals
2. DAVIDMILLS, Chris a Andrew PFEIFFER, ed. Performance fundamentals: What is performance?. Performance fundamentals. MDN Web Docs [online], 2023, 2023, 1 [cit.2023-11-10]. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/Performance/Fundamentals
3. DAVIDMILLS, Chris a Andrew PFEIFFER, JavaScript performance optimization. MDN Web Docs [online], 2023, 2023, 1 [cit.2023-11-10]. Dostupné také z: https://developer.mozilla.org/en-US/docs/Learn/Performance/JavaScript
4. SHIOTSU, Yoshitaka. JavaScript Optimization Tips: JavaScript is a truly amazing tool for front-end programming. JavaScript Optimization Tips [online]. 2021, 2021, 1 [cit.2023-11-10]. Dostupné také z: https://www.upwork.com/resources/javascript-optimization-tips
5. RAI, Bikash. First Contentful Paint (FCP) and First Meaningful Paint (FMP) Explained. First Contentful Paint (FCP) and First Meaningful Paint (FMP) Explained [online]. 2020, 2020, 1 [cit.2023-11-11]. Dostupné také z: https://www.acmethemes.com/blog/first-contentful-paint-and-first-meaningful-paint/
6. MONTTI, Roger. First Input Delay – A Simple Explanation. First Input Delay – A Simple Explanation [online]. 2021, 2021, 1 [cit.2023-11-11]. Dostupné také z: https://www.searchenginejournal.com/core-web-vitals/first-input-delay/#close
7. BUĆKO, Ziemek. *First Input Delay: A Complete Guide to Optimizing FID [online]*. 2023, **2023**, 1 [cit.2023-11-12]. Dostupné také z: https://www.onely.com/blog/what-is-first-input-delay/
8. WALTON, Philip. Time to Interactive (TTI). Time to Interactive (TTI) [online]. 2022, 2022, 1 [cit.2023-11-12]. Dostupné také z: https://web.dev/tti/
9. NOTERMANS, Thierry. What is Time to Interactive? and How can I improve TTI? [online]. 2022, 2022, 1 [cit.2023-11-12]. Dostupné také z: https://kadiska.com/what-is-time-to-interactive-and-how-can-i-improve-tti/

10. HOLMAN, Alexander (ed.). <meta>: The metadata element. The <meta> HTML element represents metadata that cannot be represented by other HTML meta-related elements, like <base>, <link>, <script>, <style> or <title> [online]. Roč. 2023, s. 1 [cit.2023-12-05]. Dostupné také z:https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta
11. JACKSON, Brian. KINSTA. How To Optimize Images for Web and Performance: What Does It Mean to Optimize Images? [online] 2023, 1 [cit.2023-12-05]. Dostupné také z: https://kinsta.com/blog/optimize-images-for-web/
12. ZÁMEČNÍK, Ondřej. *NÁVRH A REALIZACE MODERNÍCH WEBOVÝCH STRÁNEK*. Online, BAKALÁŘSKÁ PRÁCE, vedoucí Ing. Petr Münster, Ph.D. Brno: VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ, ÚSTAV TELEKOMUNIKACÍ, 2018. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=175710. [cit. 2024-03-11].
13. NEVRLÝ, Martin. *Přístupnost školních webových stránek*. Online, Bakalářská práce, vedoucí doc. PhDr. Miroslav Chráska, Ph.D. Olomouc: UNIVERZITA PALACKÉHO V OLOMOUCI, PEDAGOGICKÁ FAKULTA, Katedra technické a informační výchovy, 2012. Dostupné z: https://theses.cz/id/6ylgv4/nevrly_bakalarska_prace.pdf. [cit. [2024-03-11].
14. KUTIL, Jan. *Optimalizace zpracování jQuery v moderních webových prohlížečích*. Online, BAKALÁŘSKÁ PRÁCE, vedoucí Mgr. Jan Konečný. Brno: MASARYKOVA UNIVERZITA, FAKULTA INFORMATIKY, podzim 2011. Dostupné z: https://is.muni.cz/th/ekc3g/bc20111230print.pdf. [cit. [2024-03-11].

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků