Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra Informačních Technologií

Bakalářská práce

Vliv vybraných vlastností a nástrojů CSS na rychlost načítání webových stránek

Vlasák Vojtěch

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Vojtěch Vlasák

Informatika

Název práce

Vliv vybraných vlastností a nástrojů CSS na rychlost načítání webových stránek

Název anglicky

The impact of selected CSS properties and tools on webpage loading and performance

---

Cíle práce

Hlavním cílem práce je zhodnotit vliv a identifikovat limity vybraných vlastnosti a nástrojů CSS na načítání webových stránek a aplikací.

Dílčí cíle
- Vybrat vlastnosti a nástroje CSS pro další analýzu.
- Vytvořit vhodné testovací webové stránky.
- Navrhnout a provést experimentální měření výkonu načítání testovacích stránek.

Metodika

Práce je založena na studiu odborné a vědecké literatury, ze získaných znalostí budou vybrané vhodné vlastnosti a nástroje CSS, a dostupné optimalizační metody. Ty budou následně otestované a zhodnocené. Na základě poznatků z případové studie a z analýzy odborných zdrojů bude syntezován závěr práce.

Doporučený rozsah práce

40-50 stran

Klíčová slova

CSS front-end výkon web design optimalizace

Doporučené zdroje informací

BERMES, Barbara. Lean Websites. Melbourne: SitePoint, 2015. ISBN 9780992279462.

DOWDEN, Michael a Martin DOWDEN. Architecting CSS: The Programmer's Guide to Effective Style Sheets. New York: Apress, 2020. ISBN 978-1484257517.

HOGAN, Lara Callender. Designing for Performance: Weighing Aesthetics and Speed. Kalifornie: O'Reilly Media, 2014. ISBN 1491902515.

SKVORC, Bruno, Craig BUCKLER, Maria Antonietta PERNA a Ivan CURIC. Front-end Performance. Melbourne: SitePoint, 2017. ISBN 9781925836103.

UZAYR, Sufyan bin. Web Performance Optimization: A Practical Approach. Boca Raton: CRC Press, 2022. ISBN 9781003203735.

WAGNER, Jeremy. Web Performance in Action: Building Faster Web Pages. New York: Manning Publications, 2017. ISBN 1617293776.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Alexandr Vasilenko, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 14. 7. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 27. 10. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 10. 02. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vliv vybraných vlastností a nástrojů CSS na rychlost načítání webových stránek" 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 15.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Alexandru Vasilenkovi, Ph.D za ochotu být vedoucím mé práce a pomoc při psaní.

Vliv vybraných vlastností a nástrojů CSS na rychlost načítání webových stránek

Abstrakt

Bakalářská práce je zaměřena na roli CSS v oblasti výkonu a načítání webových stránek. V kapitolách teoretické části je popsána problematika rychlosti načítání stránek, popsáno CSS a jeho oblasti které mohou mít dopad na výkon, spolu s existujícími optimalizačními metodami. V praktické části je provedeno měření vybraných oblastí a následné zhodnocení jejich dopadu.

Klíčová slova: CSS front-end výkon web-design optimalizace

The impact of selected CSS properties and tools on webpage loading and performance

Abstract

The focus of this thesis is on CSS in relation to website performance and loading speed. The theoretical part outlines the issue of website performance, introduces and describes CSS including its areas with potential impact and role in performance of a site. It also describes existing methods of their optimization. Experimental testing and analysis is performed in the practical part of the thesis, and the selected areas are then evaluated based on the obtained measurements.

Keywords: CSS front-end performance web-design optimization

Obsah

1 Úvod

Problematika rychlosti načítání webových stránek je řešena již od počátku internetu, jedná se totiž o první poznatek, který si uživatel po přístupu na stránku odnese. Od té doby existuje konstantní souboj mezi rychlostí, a nově realizovatelnou komplexitou kterou přináší vývoj softwaru a hardwaru. CSS je jednou ze složek stránek které není v oblasti výkonu věnováno příliš pozornosti, avšak stále je potřeba znát její vliv.

Cílem této práce je zhodnotit vliv základních oblastí CSS právě na výkon načítání stránek, tak aby bylo při procesu optimalizace předem jasné, na jaké složky se zaměřit, a aby byla implikace CSS na výkon možná zohlednit již při samotném vytváření stránek.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je zhodnotit vliv a identifikovat limity vybraných vlastností a nástrojů CSS na načítání webových stránek a aplikací.

Dílčí cíle

- Vybrat vlastnosti a nástroje CSS pro další analýzu.
- Vytvořit vhodné testovací webové stránky.
- Navrhnout a provést experimentální měření výkonu načítání testovacích stránek.

2.2 Metodika

Práce je založena na studiu odborné a vědecké literatury, ze získaných znalostí budou vybrané vhodné vlastnosti a nástroje CSS, a dostupné optimalizační metody. Ty budou následně otestované a zhodnocené. Na základě poznatků z případové studie a z analýzy odborných zdrojů bude syntetizován závěr práce.

3 Teoretická východiska

3.1 Rychlost webových stránek

Na začátek je dobré si ujasnit co takový pojem “rychlost webových stránek” vůbec znamená, a proč je nutné ho řešit. Webové stránky jsou komplexní strukturou složenou z mnoha složek, které společně tvoří výslednou stránku, ale nejsou na sobě nutně závislé. Není tedy možné ohodnotit rychlost stránky pouze jedním výsledkem a mít tím úplný přehled o tom, jak se stránka chová.

Složky webových stránek

- HTML
- CSS
- JavaScript
- Obsah (obrázky, fonty, audio, video, data)

Každá z těchto složek se liší způsobem jakým ji prohlížeč získá a na stránku vykreslí, a má proto různě vysoký dopad na její načítání. Tento fakt je proto nutné zohlednit při měření a hodnocení výkonu stránky. [1]

3.1.1 Měření rychlosti

Ze složek popsaných výše je patrné že měřit jde celá řada věcí, ale je nutné si uvědomit které metriky jsou opravdu důležité zohlednit a které dokážou vhodně reprezentovat jaké to je stránky používat. Jsou to právě koncoví uživatelé, kteří si při používání stránek vytvoří názor na to, jestli se jedná o příjemný zážitek.

S tímto uvědoměním se tak velká řada webů a služeb, či vestavěných nástrojů přímo ve webových prohlížečích mimo detailní analýzu načítání individuálních složek snaží také zahrnout metriky které simulují pohled potencionálního uživatele a jeho zkušenost se stránkou. [2]

Příklady metrik:

First Contentful Paint

Kategorie zaznamenávající čas uplynulý mezi navigací na stránku, a prvním vykreslením viditelného obsahu, který předá uživateli zpětnou vazbu, že stránka se skutečně vykresluje a prohlížeč aktivně pracuje na jejím načítání.

Time to Interactive

Tato kategorie zaznamenává čas, který uplyne, než jsou prvky stránky schopné včasně reagovat na vstup uživatele. Je tedy blízce spojena s předchozí kategorií a je proto důležité příliš neupřednostňovat jednu na úkor té druhé. Situace kdy se stránka zdá načtená, ale není možno s ní pracovat může být více frustrující než její opak.

Above Fold Time

Kategorie reprezentující čas, který je potřeba k kompletnímu vykreslení viditelné části stránky při jejím načtení, zatímco obsah "níže" ještě nemusí být načten.

Total Blocking Time

Kategorie zaznamenávající čas po který je stránka zaneprázdněna nad rámec přiměřeného času na odezvu po vstupu uživatele. Např. dle RAIL modelu od Google je tento přiměřený čas 50ms, to znamená že čas nad 50ms, který proces potřebuje na dokončení je počítán jako tzv. blokující, z názvu „blocking time".

Cumulative Layout Shift

Kategorie hodnotící pohyb prvků na stránce po jejich prvním načtení. Nečekaně poskakující obsah může rozhodit soustředěnost uživatelů, a způsobit např. neúmyslné kliknutí. Naopak pohyb prvků podmíněný vstupem uživatele je často chtěný, např. při rozkliknutí navigačního menu a není proto zahrnut do této metriky.

Každé z těchto kategorií je přiděleno skóre na základě porovnání velkého množství stránek, a očekávání uživatelů. Pro ideální uživatelský zážitek je důležité brát v potaz všechny kategorie zároveň, snažit se optimalizovat jednu kategorii přesunutím výpočtu do jiné je tak nebezpečná úvaha. [3]

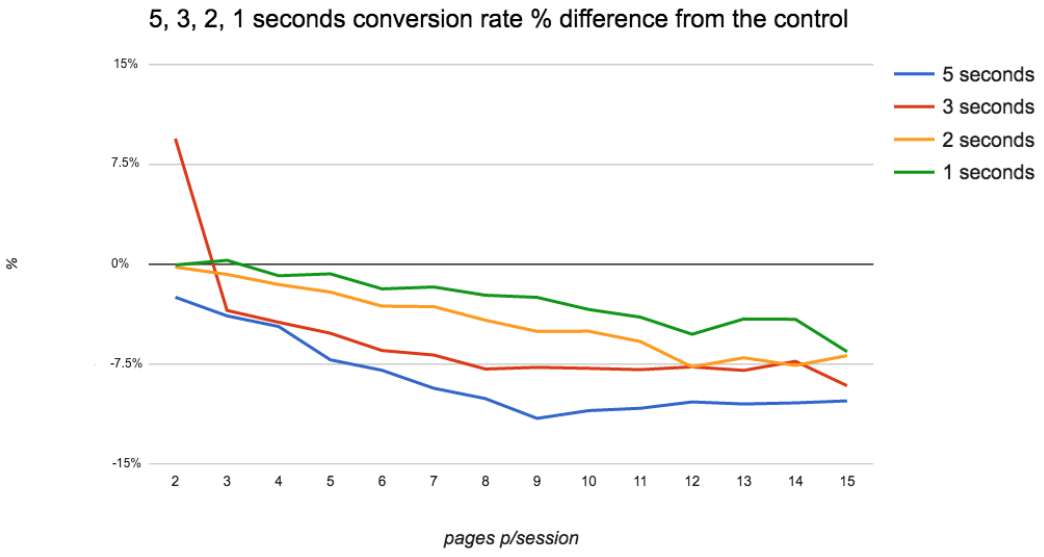
3.1.2 Vliv rychlosti načítání stránek

3.1.2.1 Lidský

Rychlost načítání stránky je součástí uživatelského zážitku stejně tak jako její design, přístupnost, nebo přehlednost, a ovlivní, jestli budou uživatelé na stránce pokračovat, anebo se raději podívají jinam. "*Web performance is user experience" – [Hogan, 2014]*

Google na svém blogu zveřejnil výsledek iniciativy, která v roce 2018 přidala rychlost jako jednu z metrik pro zobrazené výsledky mobilním uživatelům. Důsledkem zaznamenali o 15-20% lepší metriky týkající se rychlosti stránek ve spodní třetině výsledků, a stejně tak i o 20% nižší počet uživatelů kteří stránku po načtení ihned zase opustí. [4]

Při vytváření nové FT.com stránky byl zas proveden experiment při kterém byla stránka uživatelům úmyslně načítána pomaleji, a byl monitorován počet stránek které uživatelé navštíví. Při tří vteřinovém zpoždění se přes 10 stránek podívalo zhruba o 7.5 % uživatelů méně než bez zpoždění.[5]



Obrázek 1 Graf návštěvnosti uživatelů FT.com [24]

3.1.2.2 Technický

Druhým způsobem, jakým rychlost načítání ovlivní webovou stránku není přímý dopad na uživatele, ale na pozici ve výsledcích vyhledávačů. Počet návštěvníků na stránku

je tak ovlivněn bez toho, aby měli vůbec šanci se sami rozhodnout, jestli je stránka pro jejich potřeby adekvátně rychlá či nikoliv. [1]

Google nezveřejňuje váhy svého SEO algoritmu, avšak dle analýzy od First Page Sage, firmy specializující se na SEO optimalizace se u výkonu jedná o zhruba 2 %. [6]

3.1.3 Vývoj stránek

S dobou se mění přístup k používání i vytváření webových stránek, a jedním takovým přístupem jsou jednostránkové webové aplikace (SPA). Oproti klasickým webovým stránkám, které načítají pokaždé nový html soubor a s ním i spojené podsoubory, SPA staví pouze na jednom jediném html souboru pro celý běh aplikace. Do tohoto hlavního souboru je pomocí JavaScriptu dynamicky načítaný obsah podle toho kam se uživatel naviguje, bez nutnosti klasického "načtení" které uživatel velmi lehko zaznamená. Tento způsob si obvykle vyžaduje relativně velký obsah dat při prvním načtení stránky, ale již jen menší a velmi specifický obsah při další navigaci, či interakci. Mnohé aplikace běžící na tomto principu jsou také schopné fungovat bez dalšího přesunu dat a dají se tak používat dále i bez internetového připojení.

Takovéto aplikace také mohou efektivně využít mezipaměti, a opakované načtení tak ještě více značně urychlit.

Samozřejmě s sebou nesou i jisté nevýhody, jednou z nich jsou nároky na výkon koncového zařízení, na kterém je stránka načtena. Obsah musí být do stránky pokaždé vložen pomocí JavaScriptu, a zážitek se tak může velmi lišit mezi zařízeními. Je tedy složitější poskytnout dobrý zážitek pro každého z pozice správce stránky. Tento přístup také často využívá nové prohlížečové funkce, které musí zařízení podporovat, to může velmi omezit přístupnost stránky a může být pro mnohé nepoužitelná.

Jednostránkové aplikace tak dokážou v uživatelích vzbudit pocit obrovské plynulosti a responzivity, pokud mají dostatečně výkonné zařízení a podporu, ale také omezit přístupnost těm kteří je nemají.

Mimo výkon mají takovéto aplikace také horší přístupnost pro SEO boty, a bývají obvykle náročnější a rozsáhlejší na implementaci, avšak oddělený front-end od back-endu představuje možnost vytvořit relativně jednoduché nativní aplikace pro více platforem.[7]

3.1.4 Vývoj zařízení

Další změnou doby, která se netýká pouze webových stránek je vývoj v oblasti koncových zařízení a jejich dostupnosti. Počítače jsou běžně dostupné pro individuální použití, a chytré mobilní telefony nabízí téměř každému přístup k internetu, obzvlášť v rozvojových zemích. Ačkoliv vývojáři a prohlížeče stále přichází s novými funkcemi, je třeba dbát na velkou část uživatelů kteří stále používají starší a pomalejší zařízení.

Chris Zacharias popisuje svoji zkušenost při práci na stránce YouTube, kde se na podmět svého seniorního kolegy o velikosti stránky s přehrávačem pokusil stránku optimalizovat s zaměřením na rychlost a lehkost. Z původní stránky o velikosti 1.2 MB bylo pomocí optimalizačních nástrojů jako Closure Compiler, použitím nového HTML5 video přehrávače a manuální úpravou CSS, JS a obrázků docíleno výsledné velikosti 98 KB. Po týdnu sbíraní dat a jejich následné analýze se ale na první pohled nabídl nečekaný závěr, že uživatelům v průměru trvá stránku načíst déle. Tato skutečnost byla však způsobena novou skupinou uživatelů z méně vyspělých států Asie a Afriky, pro které byla stránka před optimalizací kvůli vysokým nárokům na načítání absolutně nepoužitelná. [8]

Z této situace je nutné si odnést že při analýze dat o rychlosti načítání stránek je nutné dbát ohled na cílovou skupinu uživatelů, jejich zařízení a rychlost připojení. Nové zařízení s dobrým připojením mohou i neoptimalizovanou stránku načíst rychle, zatímco na starých může i perfektní stránka vykazovat pomalé načítání.

3.2 CSS

3.2.1 Co je to CSS

CSS je stylovací jazyk který slouží k aplikaci stylu příslušnému dokumentu, v tomto případě HTML, který poté bude prohlížečem vykreslen. CSS bylo oficiálně poprvé prezentováno v roce 1994, jako jednoduchý jazyk na stylování dokumentů, který respektuje požadavky autora, i uživatele. Do té doby byly styly implementované autory prohlížečů, nebyly nijak standardizované a uživatelům často umožňovaly jen malou hrstku kontroly nad výsledným vzhledem. [9]

Ačkoliv CSS je často spojováno pouze se slovem "styl", tak značně ovlivňuje, jak se stránkou uživatel pracuje a na jakých zařízeních je vůbec použitelná.

3.2.2 Specifikace

CSS specifikace je publikována skupinou W3C, a na jejím vývoji se podílí celá komunita vývojářů. Návrhy prochází kritikou a diskuzemi, než jsou nakonec přijaty a publikovány jako W3C doporučení. Specifikace byla dříve aktualizována plošně ve verzích, ale vydáním verze 2.1 se signifikantně zvedl její objem, a tak bylo rozhodnuto specifikaci rozdělit do modulů, které budou samostatně iterativně aktualizované a rozšiřované. Nyní tedy máme specifikace CSS, a moduly různých úrovní, skupina W3C průběžně publikuje takzvané snapshoty, ve kterých popisuje, v jakém stavu jednotlivé moduly jsou.[10]

3.2.3 Podpora prohlížečů

Je důležité si uvědomit že CSS je pouze deklarativní jazyk, a nikoliv imperativní. To znamená že se nezabývá tím, jak jsou styly aplikovány, ale pouze tím, jaké mají být. Implementace je závislá na prohlížeči, který by měl co nejlépe splňovat specifikaci od skupiny W3C, tak aby stránky vypadaly a fungovaly konzistentně napříč prohlížeči pro všechny uživatele.

Historicky se podpora stále zlepšuje, avšak stále existují vlastnosti, které se napříč prohlížeči chovají jinak, či nejsou některými vůbec podporované. Z toho důvodu existují webové stránky, které sledují podporu jednotlivých vlastností napříč velkou řadou prohlížečů, aby si vývojáři dokázali ověřit, zda má smysl vlastnost použít, např. caniuse.com.

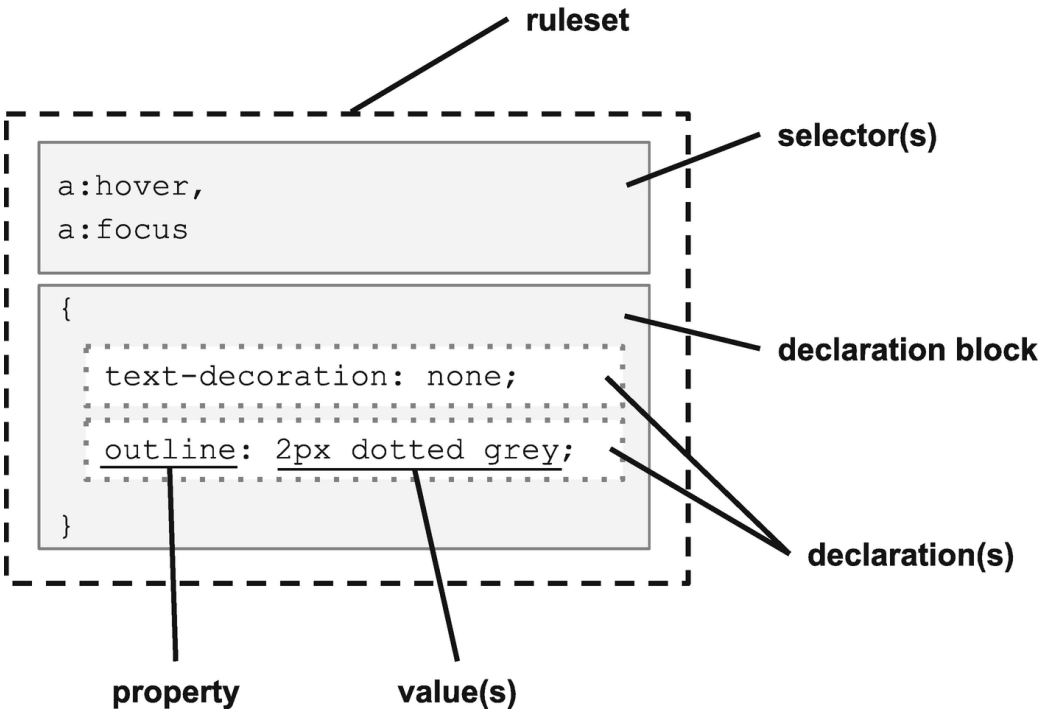
Samotné CSS má také funkce, které umožnují vytvářet podmíněná pravidla podle podpory prohlížeče, a je tak možné implementovat alternativu v případě že podporované není. Také je skutečností že sama specifikace často nespecifikuje, jak přesně daného efektu docílit, takže i v případě že dva prohlížeče specifikaci splňují, může být výsledný vzhled odlišný a nežádaný.

Podpora je tak stálý závod prohlížečů, které musí nové vlastnosti a funkce implementovat, a vývojářů kteří požadují dostatečnou podporu, aby mělo smysl funkce využívat, a nakonec uživatelů kteří musí své zařízení a prohlížeče aktualizovat. Je tak velmi důležité si být vědom tohoto omezení, a nepovažovat existenci funkce ve specifikaci jako potvrzení že je rozumné ji použít. Jednou ze základních vlastností CSS ale je že prohlížeče ignorují neznámé či nepodporované vlastnosti, takže i v případě že něco není podporované by stránka měla být stále použitelná.

Prohlížeče se také liší v základním stylu, který je použit v případě, že stránka nepoužívá vlastní, takže i bez použití jakéhokoliv CSS ze strany vývojáře není jistotou že každý uživatel uvidí to stejné. Jednou z běžně používaných metod je proto tzv. CSS normalizace, při které jsou všechny hodnoty nastavené vývojářem explicitně, a každý prohlížeč by tak měl vykreslit stránku stejně. [11]

Jednou z výjimečných situací jsou ale programy Emailových klientů, které často nepodporují nové CSS funkce a obsah pro ně musí být vytvářen s opatrností.

3.2.4 Struktura



Obrázek 2 Znázornění syntaxe [25]

Vlastnosti

Vlastnosti jsou identifikátory které označují co se bude prvku na stránce nastavovat. Prohlížeče mohou také disponovat svými experimentálními či nestandardními vlastnostmi, které je možné použít pomocí speciálního prefixu pro konkrétní prohlížeč, např. “-webkit-“ pro prohlížeče stavěné na jádře Chromium a většinu iOS prohlížečů nebo “-moz-“ pro Firefox.

Vlastnosti obvykle přijímají jen jím specifické hodnoty, např. procento pro délku, nebo název barvy pro pozadí, avšak všechny vlastnosti přijímají hodnoty “Initial“, “Inherit“, a “Unset“.

Initial – Nastaví vlastnost na její defaultní hodnotu, případně na hodnotu nastavenou prohlížečem.

Inherit – Nastaví vlastnost na hodnotu rodičovského prvku.

Unset – Nastaví vlastnost na hodnotu rodičovského prvku, pokud se jedná o dědičnou vlastnost, v opačném případě se chová jako “initial“.

Hodnoty

Hodnoty definují, jak má prohlížeč danou vlastnost zpracovat. Hodnoty musí splňovat správný formát pro danou vlastnost, aby byly prohlížečem aplikovány. Platné hodnoty mohou být celá a desetinná čísla, velikosti následovné jednotky, URL a URI odkazy, barvy, textové řetězce, či klíčová slova jako např. “Auto“, “None“, “Initial“.

Speciální hodnotou je klíčové slovo “!important“ které označí hodnotu jako důležitou a ovlivní tak její nastavení v případě že existuje vice pravidel cílících na stejný prvek se stejnou vlastností.

Deklarace

Páru vlastností a hodnot se říká deklarace, jsou od sebe oddělené dvojtečkou, a necitlivé na velikost písmen. Úkolem prohlížeče je aplikovat deklarace pro všechny prvky na stránku kterým patří.

Bloky

Deklarace mohou být vkládány do bloků, ohraničením do vlnitých závorek. Bloky mohou být prázdné, a deklarace uvnitř musí být odděleny středníkem.

Selektory

Selektor označuje prvky na stránce, kterým bude následující blok deklarací nastaven. Selektory obvykle cílí na typ, třídu, identifikátor či atribut prvku. Každý z těchto typů selektoru má svoji váhu, se kterou prohlížeč pracuje v případě, že na nějaký element cílí více pravidel.

Součástí specifikace také přibývá množství pseudotříd které lze v selektoru využít, kterými lze cílit na prvky např. dle toho co obsahují, pomocí “:has()“ nebo tím čím nejsou použitím “:not()“.

Pravidla

Aby bylo možné přiřadit určitý styl jen vybraným prvkům, existují pravidla. Pravidla vznikají kombinací jednoho nebo více selektorů, a bloku. Pravidla nemohou obsahovat další pravidla.

@Pravidla

Tento typ pravidel necílí na prvky stránky, ale říkají prohlížeči, jak se má chovat. Slouží např. k importování dalších CSS souborů, k aplikování pravidel pouze při určitém rozlišení nebo pouze když podporuje specifikovanou funkci.

Preprocesory

Existují různé preprocesory snažící se uživateli zpříjemnit a zefektivnit psaní CSS, a to zpřístupněním syntaktických funkcí kterými CSS (zatím) nedisponuje. Takto psané CSS je nutné před použitím zkompilovat do formy která splňuje specifikaci, aby ji prohlížeč dokázal zpracovat. Zkompilované CSS může ale získat neoptimální podobu které by se uživatel obvykle snažil vyvarovat, ale ze syntaxe preprocesoru nebyla na první pohled jasná. [12]

3.2.5 Kaskáda

Jedním ze základních chování které CSS styly obsahují je právě kaskáda, jedná se o algoritmus který má za cíl zjistit které pravidlo bude na prvek aplikováno, na základě toho kde se nachází. Styly je možné pro účely kaskády rozdělit dle tří původů, a vrstev které je možné vytvořit pomocí "@layer".

User-agent – Základní styly aplikované prohlížečem, mohou být pouze simulované v kódu, nebo využívat skutečný interní CSS soubor.

User – Styly, které jsou vytvořeny uživatelem, vytvořením CSS souboru v správné složce prohlížeče nebo využitím doplňku.

Author – Styly aplikované autorem stránky.

Navrch ještě získávají speciální prioritu pravidla v "<style>" bloku HTML souboru, hodnoty nastavené pomocí "transition" a "animation" vlastností, a označení hodnoty jako "!important", které převrátí prioritu vrstev. [13]

3.2.6 Specificita

Specificita úzce souvisí s algoritmem kaskády, jedná se o váhu pravidla, dle které je vybráno jedno z cílících na stejný prvek stránky. Tento výběr probíhá na pravidlech již vybraných v procesu kaskády, tudíž ten nejvíce specifický selektor nemusí být vždy vybrán, např. pokud se nachází v externě importovaném souboru který algoritmus kaskády vyřadí.

Hodnoty specificity jsou přiděleny selektoru do tří skupin, od největší specificity po nejnižší ve formátu X-X-X.

ID – té přidávají selektory cílící na ID prvku, které může být na stránce pouze jednou.

Třída – do které spadají selektory cílící na třídu, atribut, pseudotřídu.

Typ – které přispívají selektory cílící na typ, a pseudotypy přístupné pomocí dvojtečkové notace, např. "::placeholder".

Specificita je porovnávána po skupinách zleva doprava, v případě že má selektor ve skupině vyšší hodnotu tak je vybrán, a na následující skupiny není brán ohled, jsou-li všechny skupiny stejné, pak vyhrává selektor, který je v souboru jako poslední. Např. specificita 2-1-1 má vyšší váhu než 1-4-4, ale nižší než 3-0-0

Hodnota specificity jde ruku v ruce s tím, kolik by měl selektor vybrat prvků stejného typu. ID může být vždy jen jednou, a prvku určitého typu s třídou bude vždy stejně nebo méně než prvků toho typu celkem. Existuje však několik výjimek této logice.

- Pro pseudotřídy ":not()", ":has()", ":is()", jejich parametry je seznam selektorů, je přidáno do skupiny dle parametru selektorů, samotné pseudotřídy nepřidávají na specifitě třídy jak by se mohlo zdát.

- Pseudotřída ":where()" ačkoliv jejím parametrem je také seznam selektorů, nepřidává specifitu žádné skupině. Je tedy možné vytvořit selektor se specificitou 0, stejně tak jako použitím univerzálního selektoru.

- Styly deklarované přímo v “<style>“ atributu prvku, takové mají vždy prioritu nezávisle na jejich specificitě. Tímto způsobem jsou často aplikované styly pomocí JavaScriptu, a je tak velmi složité je přepsat.

- Poslední výjimku tvoří jako u kaskády označení “!important“, které přebije každý selektor který tak označen není.

Dobrým zvykem je vytvářet selektory s co nejmenší specificitou, které pak nevyhrávají nad selektory, které jsou skutečně míněné specificky. Právě výše zmíněná pseudotřída “:where()“ se zde stává velmi užitečnou. [14]

3.2.7 Dědičnost

Řada vlastností je tzv. dědičných, které nemají žádnou vlastní defaultní hodnotu, a získávají ji tak od jím nadřazeného prvku, či od stylu prohlížeče. Vlastnostem, které dědičné nejsou je možné explicitně přiřadit hodnotu “inherit“ a její skutečnou hodnotu tak zdědí. Pro vrácení původní hodnoty prvků je možné využít hodnoty “initial“ nebo “unset“.

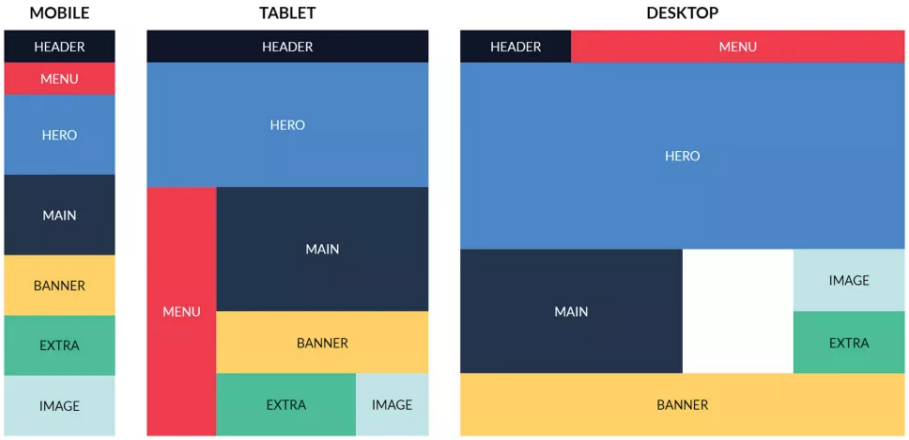
Na dědičnost ale nelze vždy spoléhat, v určitých případech může prohlížeč specifikovat defaultní styl původně dědičným prvkům. [15]

3.2.8 Využití CSS

CSS obsahuje stovky vlastností které jsou dle specifikace stabilní a doporučené pro implementaci prohlížeči, a další stovky v různých fázích vývoje. CSS nám umožňuje mimo jiné měnit barvy prvků, barvy textu, přidávat prvkům efekty a animace, měnit jejich tvar, či upravovat, jak se chovají, když s nimi uživatel zachází. Ale největší a nejdůležitější část těchto vlastností se týká manipulace rozložení prvků na stránce, s pomocí takzvaného box modelu.

3.2.8.1 Responzivní design

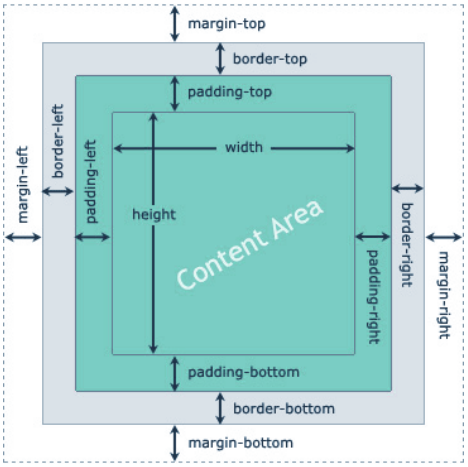
Responzivní design je v dnešní době stále rozšířenější, webové stránky jsou navštěvovány na velkém množství různých zařízení a je potřeba aby se dokázali přizpůsobit. S vývojem CSS jsou opakovaně přidávány nové funkce, které zjednodušují a zefektivňují práci s rozložením prvků na stránce.



Obrázek 3 Znázornění responzivního rozložení [26]

Box Model

Každý prvek na stránce je reprezentován čtvercem o určité velikosti, obsahem, vycpáním uvnitř, hranou, a odsazením z vnější strany. Všechny tyto hodnoty spolu pracují pro výpočet správné velikosti, a umístění prvku na správnou pozici na stránce, a vůči ostatním prvkům. Tyto hodnoty však vůbec nemusí být pevně nastavené, a často ani není žádoucí, aby byly. Moderní CSS umožní nastavit prvkům takový styl, aby se dokázaly samy harmonicky rozložit na stránku dle nadefinovaných pravidel, a to nezávisle na velikosti či typu zobrazovacího zařízení.



Obrázek 4 Znázornění box modelu [27]

Pryč jsou také dny, kdy bylo nutné určovat zvlášť každému prvku, jak se chovat. S implementací “flexbox“ a “grid“ vlastností je možné vytvořit kontejnery které si do velké části prvky rozloží samy podle námi nastavených pravidel, a efektivně tak využijí místo bez složitého manuálního zásahu a plánovaní.

Media Queries

Druhým nástrojem k dosažení responzivního designu jsou media queries, @pravidla která umožnují aplikovat styl pouze v případě že rozlišení koncového zařízení, či preference nastavené uživatelem splňují zadanou podmínku.

Dobrým zvykem je považovat zobrazení pro mobilní zařízení jako základ stránky, často totiž bývá jednodušší a blíže defaultnímu stylu. Pomocí media queries stačí styly doplňovat, a předcházet tak psaní zbytečných stylů které budou v jednom ze zobrazení přepsány. Také umožnují vytvořit speciální styl např. pro tisk stránky. [10]

3.2.8.2 Animace

Animace jsou relativně obsáhlou a částečně novou sekcí CSS, dalo by se o nich říci, že jsou na hraně toho o co by se deklarativní jazyk měl snažit. Při vytváření animací je žádoucí mít kontrolu nad jejich průběhem a časováním, a to pro CSS vytváří komplexitu kterou se ještě nepodařilo úplně vyřešit. CSS animace tak aktuálně poskytují dobrý a efektivní základ pro jednoduché animace, a WAAPI (Web Animations API) poskytuje standardizovanou knihovnu pro animování pomoci JavaScriptu.

Animace v CSS jsou dostupné využitím tří způsobů:

Pseudotřídy

Pseudotřídy jsou třídy které element dostane v případě že se s ním uživatel snaží jakkoliv zacházet, např pseudotřída ":hover" při najetí myši, či ":focus" při vybrání např. textového pole. Pro tyto třídy je možné vytvořit pravidla, které se automaticky použijí, pokud daný element tuto třídu vlivem uživatele získá. Animace tímto způsobem jsou vhodné pro indikaci že je s nějakým prvkem vůbec možné zacházet, či dát uživateli zpětnou odezvu že stránka reaguje na jeho akce.

Transition

"Transition" vlastnost umožňuje definovat, jak má vypadat přechod mezi změnou vlastnosti prvku, např. změna barvy z modré na bílou. Lze nastavit, jak dlouho má přechod trvat, jaký má mít průběh či jestli má mít nějaké zpoždění. Přechody je možné

definovat zvlášť pro individuální vlastnosti, ale pouze vybrané vlastnosti jsou podporované.

Vlastnost také pouze nastaví jak má přechod mezi stavy probíhat, samotná hodnota pak musí být změněna JavaScriptem nebo např. ":hover" pseudotřídou.

Animation

"Animation" vlastnost poskytuje maximum kontroly nad animacemi které CSS nabízí, umožňuje nadefinovat řadu klíčových bodů, ve kterých lze přesně nastavit jak a kdy má ke změně vlastnosti dojít. Animace je také možné kombinovat s různým zpožděním, a docílit tak relativně složité animace spojením více animací za sebou.

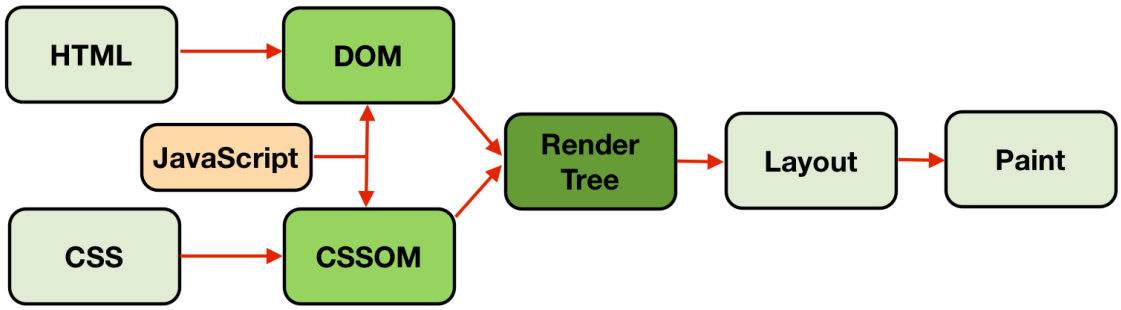
Animacím lze také nastavit délka, počet iterací, směr, či jestli má prvek na konci animace zůstat, nebo se vrátit. Je možné také průběh pozastavit a opět spustit, avšak pro efektivní využití je potřebné využít JavaScriptu. [10]

Speciální pozornost si zaslouží vlastnost "**transform**", kterou je typické kombinovat s výše popsanými způsoby. Vlastnost umožňuje prvkům nastavit transformace jako je rotace, naklonění, posunutí, zvětšení / zmenšení, či změnu perspektivy. Existuje také ve 3D variantě, která aplikuje transformace v 3D prostoru. Transformace pomocí této funkce jsou provedeny tak aby změna neovlivnila tok prvků stránky, její využití je tak rozdílné od klasických vlastností, které manipulují s pozicí a velikostí.

3.3 CSS výkon

CSS má velkou řadu funkcí a vlastností, které se liší komplexitou, a také implementací, je proto zřejmé že se také liší nároky na výkon. Také samotné využití a načítání CSS souborů ovlivní rychlost načtení stránky více než jen stáhnutí potřebného souboru.

3.3.1 Render Blocking CSS



Obrázek 5 Proces sestavení stránky [28]

Než je prohlížeč schopen začít vykreslovat stránku, musí nejdřív načíst všechny CSS soubory které jsou nalinkované v html souboru tagem ‘link’, či navíc importované v některém CSS souboru. Vykreslit stránku bez aplikovaných stylů a pak ji znovu překreslit by bylo pro uživatele nejen velmi nepříjemné, ale i neefektivní co se výkonu týče. Avšak to znamená že velký objem CSS souborů může být limitující faktorem v této fázi načítání.

Po načtení CSS souboru je prohlížeč ještě musí přečíst a vytvořit CSSOM (CSS Object Model), který slouží jako API přes které je možné manipulovat s CSS funkcemi pomocí JavaScriptu.

Tento proces může hrát větší roli při využívání CSS frameworků, protože i při využití malé části jejich funkcí může být potřeba načíst celý soubor který může být relativně obsáhlý.

Celý proces vedoucí k vykreslení stránky se nazývá “critical rendering path”. [2]

3.3.2 Drahé CSS vlastnosti

Ne všechny vlastnosti, které můžeme měnit mají stejně velký vliv na sebe, či ostatní prvky na stránce, a tak mají i odlišnou výpočetní komplexitu.

Obrázek 6 Proces vykreslení stránky [29]

Změna nějakého stylu obvykle vyžaduje jednu nebo více akcí prohlížeče, těmi je “layout“, “paint“, a “composite“.

Layout

Proces má za úkol projít všechny prvky v DOMu, zjistit které jsou viditelné, a správně vypočítat jejich geometrii a rozložení. I relativně jednoduchá stránka obsahuje několik prvků, které se navzájem ovlivňují, a které zas samotné ovlivňuje jejich obsah.

Paint

Proces má za úkol správně identifikovat pořadí ve kterém mají být prvky vybrané v předchozím kroku vykresleny, aby byla správně respektována hierarchie dokumentu a z-index, a nežádaně se nepřekrývaly. Následně jsou tyto prvky rasterizovány

Composite

Proces má za úkol sestavit finální podobu stránky. Stránka z předchozích procesů může být rozložena do více vrstev, které jsou zvlášť uchovány v paměti. Tyto vrstvy jsou v tomto procesu pozicovány a vykresleny do finální podoby stránky, tato poslední fáze v moderních prohlížečích typicky běží ve svém vlastním vlákně, a je nejméně náročná.

Je nutné dbát na to že každý proces pracuje s daty získanými v tom předchozím, takže vlastnost, která vyžaduje změnu v layoutu si vynutí zopakování paint i composite procesu. [16]

Vlastnosti, které si vynucují více akcí při změně lze považovat za drahé vlastnosti, a mělo by být zváženo v jaké míře je používat, např. v animacích. Obecně se jedná o vlastnosti, které mění velikost či pozici prvků, a ovlivňují tak tok stránky. Jelikož implementace vlastností záleží na prohlížeči, tak i to jaké vykreslovací procesy jejich změna vyvolá se může lišit.

V dnešní době prohlížeče využívají komplexních algoritmů pro vhodné vybrání pouze těch prvků, které je skutečně nutné přepočítat a překreslit, a změny v DOMu nebo CSSOMu si již nevyžadují kompletní přepočítání a překreslení celé stránky, nicméně se stále jedná o jednu z náročnějších operací které prohlížeč provádí. Proces, který je zodpovědný za tento výběr se nazývá “Style Invalidation”. [17]

3.3.3 Selektory

Selektory jsou prohlížečem čteny zprava doleva, selektor nejvíce napravo se nazývá klíčový selektor. Podle tohoto klíčového selektoru prohlížeč vybere všechny prvky které ho splňují, a pak u každého ověří zda splňuje i zbytek selektoru, čtený postupně doleva.

Je tedy vhodné, aby klíčový selektor byl co nejspecifičtější a prohlížeč tak ověřil co nejmenší množství prvků, a zároveň aby celý selektor byl co nejjednodušší. [18]

Selektory jsou úzce spojené s výše popsaným procesem “Style Invalidation”, při kterém na základě existujících selektorů prohlížeč ověřuje, zda změna na stránce bude vyžadovat převýpočet či nikoliv, to znamená že pří velkém počtu pravidel je vysoká šance že pro změněný prvek bude existovat nějaký selektor, a bude nutné ověřit zda opravdu potřebuje být převypočítán, ačkoliv přímo na něj nemusí být pravidlo cíleno.

Např. Při existenci selektoru “a b” je nutné ověřit jakoukoliv změnu na prvku “a” (např. vložení nového prvku “a“ na stránku), zda nemá potomka “b” na který je cílený selektor, i přes to že na prvek “a” samotný žádný styl cílený není. [17]

3.3.4 Animace

Cílem animací je předejít okamžitým skokům na stránce, a vytvořit místo nich plynulé přechody z jednoho stavu na druhý. Faktem ale je že animace s sebou nesou své nároky na výkon, a mohou stránku ještě více zpomalit a znepříjemnit. Animace jsou ve webových prohlížečích typicky cíleny na 60fps, čili se prvek musí změnit 60krát za vteřinu pro vytvoření přechodu který uživatel považuje za plynulý. V případě že je ale animace moc náročná může být tohoto cíle nemožno dosáhnout, prohlížeč začne vynechávat jednotlivé snímky a výsledkem bude trhaný přechod, animace tím vyvolá opačný dojem, než kterého se snaží dosáhnout. [19]

3.4 Optimalizační metody a nástroje

3.4.1 Optimalizovat Kritické CSS

V předchozí kapitole bylo popsáno “Render blocking CSS”, v této kapitole bude vysvětleno jak této skutečnosti využít pro zrychlení načítání stránky.

Základem je zbytečně nerozdělovat CSS do více souborů, každý soubor navíc může od prohlížeče vyžadovat vytvořit nové spojení se serverem, které s sebou nese zpoždění, které zpomalí načítání celé stránky. Pro velmi obsáhle stránky ale může být vhodné CSS soubor rozdělit aby nebylo načtení až příliš zdlouhavé. Na jeden soubor který obsahuje základ který je využívaný hlavní částí stránky, která se při navigace na stránce příliš nemění, a může tak být znovu použita z mezipaměti bez opakovaného načítání, a druhý soubor obsahující styly nutné pouze pro aktuální stránku.

Druhým krokem je identifikovat kritické CSS, neboli CSS které je potřeba pro prvky které jsou viditelné ihned po načtení stránky, taková část stránky je nazývaná “Above the Fold” (ATF). Toto kritické CSS je možné prioritizovat, například vložením stylu přímo do sekce v HTML dokumentu. Takovýto přístup sice načítání zrychlí, ale nese s sebou své problémy, například nemožnost tento styl efektivně načítat z mezipaměti, či nutnost ho vložit do každého HTML dokumentu stránky. Při využití této metody pouze pro styly, které budou okamžitě viditelné může být i s nevýhodami načítání zrychleno. Zbytek CSS, které je

potřebné k správnému vykreslení stránky je načteno bez blokování načítání pomocí vztahu “preload“. [20]

3.4.2 Optimalizovat pravidla

Další částí, která může být optimalizována jsou samotná pravidla. S vývojem stránek se může stát, že dříve vytvořená pravidla již nejsou na stránce využitá, ale i přes to musí být staženy a prohlížečem zpracovány. Při této optimalizace je nutné brát v ohled všechny stránky u kterých je ten stejný CSS soubor nalinkován, na první pohled nevyužité styly mohou být využity pouze na jedné stránce, prohlížeče a nástroje většinou porovnávají využití pouze na aktuální stránce.

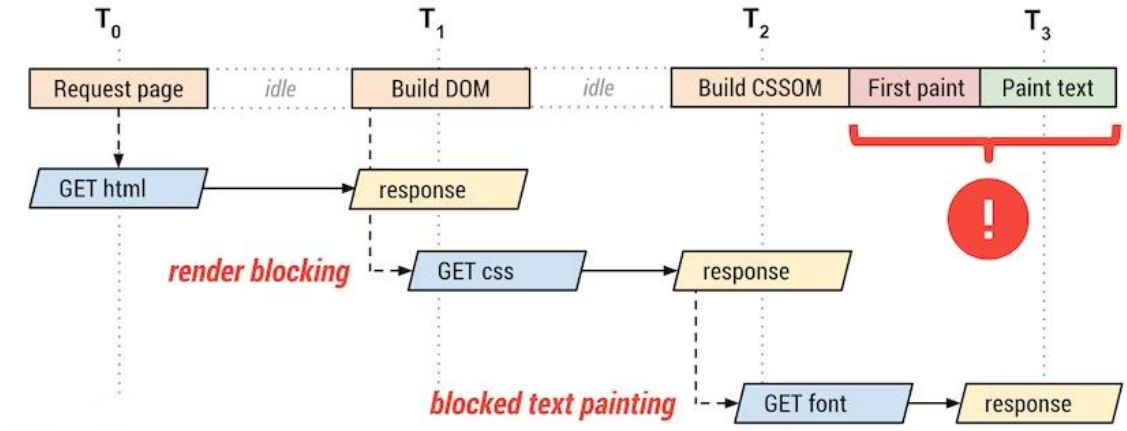
Další možností je kondenzovat pravidla které mají společné či podobné vlastnosti a hodnoty. Pro opakovaně používané vlastnosti jako jsou např, ohraničení, odsazení, velikost textu je vhodné vytvořit konkrétní třídy které budou prvkům přiřazené. Zmenší se tak nejen velikost samotného souboru, ale také bude přehlednější a lehčí na údržbu, a styl bude konzistentní na všech prvcích. Spolu s tímto lze také vhodně využít takzvaných “short-hand” vlastností, které kombinují více vlastností stejného charakteru (např, tloušťku, barvu a typ ohraničení) do jedné. Společné vlastnosti lze tedy nastavit hromadně, a jednotlivě odlišné nastavit zvlášť. [1]

3.4.3 Optimalizovat načítání fontu

Ačkoliv fonty nejsou čistě záležitostí pouze CSS, a lze je využít i bez jeho použití, CSS nad nimi poskytuje lepší kontrolu.

Základem je správně si vybrat, fonty jsou dostupné v různých formátech, které se liší kompatibilitou a velikostí. Většina moderních prohlížečů nyní již podporuje formát WOFF2, který je vytvořen pro využití na webu a velmi efektivně data komprimuje. Fonty se také liší komplexitou, pokrytí více abeced, či obsahu více tlouštěk může značně zvětšit velikost požadovaného fontu.

Například Google se na svém “Google Fonts“ API snaží poskytnout co obsahově nejmenší obsah který splní požadavek, je možné i stáhnout pouze určitý počet písmen v případě že je font využit jen pro titulek stránky. [2]



Obrázek 7 Proces načítání stránky [30]

Po vhodném výběru je nutné ještě správně font na stránce použít. Základním chováním prohlížečů je font začít stahovat až v případě že prohlížeč při vykreslování stránky zjistí že font na stránce opravdu bude vidět, a pak musí čekat na jeho dokončení.

Pokud text bude na stránce v každém případě použit, je vhodné ho manuálně vybrat a nastavit k němu atribut “preload”, stažení fontu tak bude zahájeno dříve v procesu načítání stránky.

To že se prohlížeč pokusí font stáhnout ještě neznamená, že se včas podaří, či vůbec. Proto je vhodné vybrat správný způsob kterým bude na stránce použit. K tomu slouží vlastnost “font-display”, která umožní specifikovat, jak se má prohlížeč při načítání fontu chovat, využitím následujících tří stavů.

Block

V tomto stavu ještě font není načten, a tak nemůže být text vykreslen. Namísto toho je text vykreslený neviditelně, a pokud se font stihne načíst do konce této fáze tak bude použit.

Swap

Tento stav nastane ihned po blokovém, pokud při něm není font ještě načten, tak je namísto toho text vykreslen s rezervním fontem, a v případě že se do konce fáze původní font načte tak je nahrazen.

Failure

V tomto stavu je font označen za selhaný, a je namísto něj použit definovaný rezervní font.

Hodnoty funkce “font-display” specifikují jaký má být interval těchto stavů, z 5 podporovaných variant. Při nenastavení hodnoty je na prohlížeči se rozhodnout jak se s načtením fontu vypořádat, a chování není standardizované. [21]

Auto

Chování je přenecháno prohlížeči, typické bývá chování podobné hodnotě swap

Block

Font má krátkou fázi stavu “block“ (doporučením dle specifikace jsou zhruba 3 sekundy), která následuje nekonečná fáze swap, po kterou se prohlížeč snaží font stále stáhnout a nastavit. Tato hodnota je užitečná především pokud text bez daného fontu není čitelný či ztrácí svůj význam.

Swap

Font má velmi krátkou blokovací fázi (dle specifikace je doporučeno 100ms a méně), a nekonečnou fázi stavu swap.

Fallback

Font má opět velmi krátkou blokovací fázi, po které nastává krátká fáze stavu swap, pokud stále není možné font načíst tak je textu ponechán nastavený rezervní font.

Optional

Při této hodnotě se prohlížeč snaží načíst font okamžitě, nemělo by tedy dojít k žádnému blokování či změně viditelné uživateli. Prohlížeč může oddálit vykreslení textu pokud identifikuje, že font bude bez dalšího zpoždění načten. Prohlížeč také nemusí ani začít stahování fontu, v případě, kdy identifikuje že by se tak nestalo dostatečně rychle.[22]

3.4.4 Použití will-change vlastnosti

"Will-change" je CSS vlastnost která slouží pro předání informace prohlížeči, že prvek bude v blízké době změněn, či přímo jak bude změněn. Dle této informace se prohlížeč pokusí prvek vhodně připravit na změnu ještě, než začne, aby nedošlo k zahlcení přesně v momentě akce uživatele, která změnu způsobí.

Je třeba se však vyvarovat zneužití této vlastnosti a použitím po celé stránce. Důsledky optimalizace může být zpomalení běžné funkcionality, či zaplnění paměti použité prohlížečem. Je doporučeno vlastnost aplikovat s dostatečným předstihem pouze prvku který skutečně očekává změnu, např. při najetí myší na nadřazený prvek nebo zmáčknutí klávesy, a po změně ho zase odebrat. [23]

3.4.5 Minifikovat a komprimovat CSS soubor

Při psaní CSS je nutné zvolit takový styl, aby se v něm autor dokázal vyznat, a byl schopen ho upravovat, pokud je to nutné. Avšak takovýto formát vhodný člověku není nutný pro prohlížeč, a už vůbec ne ten nejefektivnější. Soubor je proto vhodné před použitím na stránce minifikovat, odstranit všechny znaky a sémantické pomůcky které nemají vliv na jeho funkčnost. Nejprve je však nutné si původní soubor zkopírovat pro pozdější úpravy.

Další způsob, kterým je možné snížit velikost souboru je využitím komprese. Webservery často poskytují vestavěnou funkci pro využití komprese a soubor tak automaticky před odesláním uživateli vhodným algoritmem zkomprimují, např. do formátu gzip, a vloží mu vhodnou hlavičku, aby ho prohlížeč dokázal zpracovat.

4 Vlastní práce

Obsahem vlastní části práce je vybrání vhodných CSS oblastí, které na základě otestování poskytnou reprezentativní výsledky o vlivu CSS na výkon a načítání stránek. Na základě těchto výsledků bude zhodnoceno, zda se při optimalizaci stránek vyplatí zaměřit na CSS, a případně na jaké konkrétní oblasti.

Testované oblasti:

1. Vlastnosti podle procesů zpracování, tzv. drahé vlastnosti
2. využití vlastnosti will-change
3. využití vlastnosti font-display a přednačtení fontů
4. selektory
5. minifikace CSS a komprese
6. optimalizace above the fold obsahu

4.1 Testování CSS vlastností dle vykreslovacích procesů

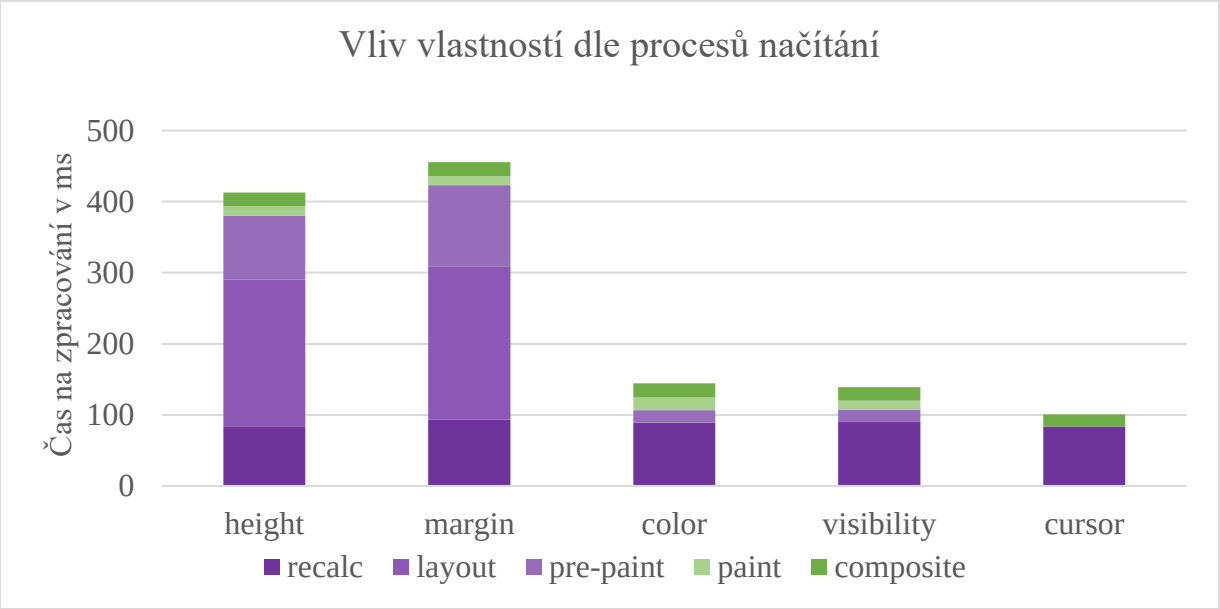
V první oblasti jde o vlastnosti vybrané na základě toho, jaké procesy z množiny “layout“, “paint“ a “composite“ vyvolá v prohlížeči jejich změna. Od každé varianty jsou vybrány 2 vlastnosti.

1. Vybrané vlastnosti, které vyžadují proces “layout“: margin, height
2. Vlastnosti, které vyžadují proces “paint“: visibility, color
3. Vlastnosti, které vyžadují pouze proces “composite“: transform, cursor

Pro testování byla vytvořena stránka obsahující pouze základní HTML strukturu, a 5000 prvků typu “div” kterým je testovaná vlastnost nastavena. Styl je nadefinován v HTML souboru v sekci “<style>“, po kliknutí na tlačítko je pomocí JavaScriptu nastavena třída kontaineru pro prvky “div”, díky které se styl stane platným, a prohlížeč ho vykreslí.

Průběh nastavení byl sledován ve vývojářském okně vestavěné v prohlížeči, a byla vybrána pouze sekce záznamu která je podmíněná aktivací tlačítka, změna prvků je tak možné izolovat od počátečního načtení stránky.

4.1.1 Měření prohlížeče Google Chrome



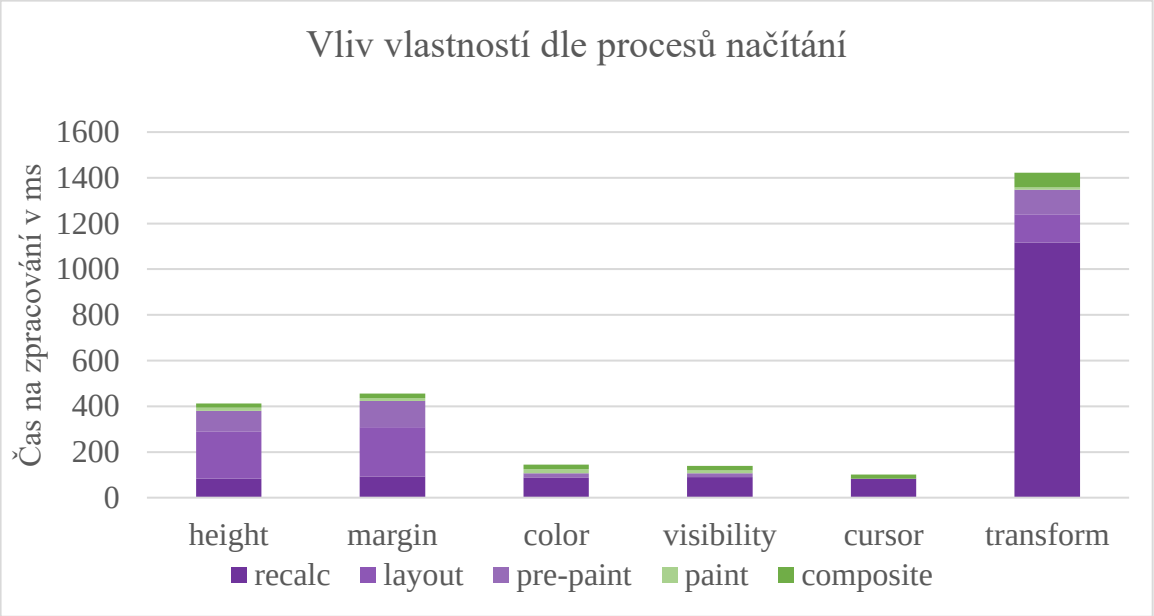
Graf 1 Vliv vlastností dle procesů načítání – Chrome
Zdroj: autor

	height	margin	color	visibility	cursor
recalc	83.27 ms	93.49 ms	88.83 ms	89.70 ms	83.39 ms
layout	206.82 ms	214.80 ms	0.00 ms	0.00 ms	0.00 ms
pre-paint	89.73 ms	114.94 ms	17.61 ms	17.21 ms	0.00 ms
paint	13.79 ms	12.43 ms	17.96 ms	13.22 ms	0.00 ms
composite	19.30 ms	19.81 ms	20.08 ms	18.73 ms	17.35 ms
Celkem	412.91 ms	455.46 ms	144.47 ms	138.86 ms	100.75 ms

Tabulka 1 Vliv vlastností dle procesů načítání – Chrome
Zdroj: autor

Z naměřených hodnot je vidět vztah nad celkovou dobou aplikace stylu, a procesy které jsou k tomu nutné. U vlastností, které mají vliv na pozici ostatních prvků v struktuře stránky je největším podílem právě výpočet rozvržení, tedy proces “layout“. Zatímco procesy “paint“ a “composite“ se mezi vlastnostmi příliš nemění, jelikož je měněné pokaždé stejné množství prvků a nejsou vytvářeny nové vrstvy pro kompozitor.

K velkému rozdílu ale došlo při aplikaci vlastnosti “transform“, která by měla využít především funkce kompozitoru a grafické akcelerace. Při aplikace na takto velké množství prvků přestane stránka reagovat na vstupy a stane se nepoužitelnou.

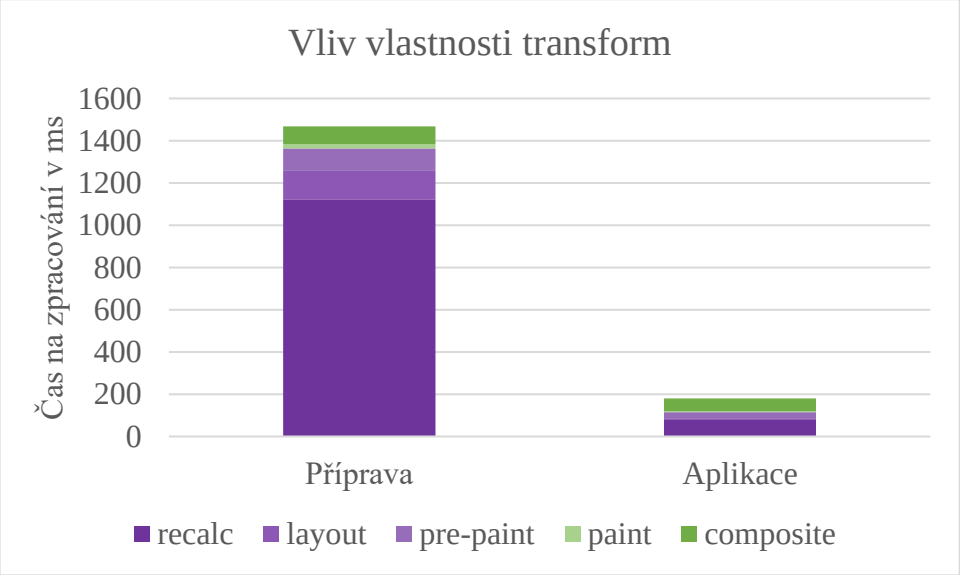


Graf 2 Vliv vlastností dle procesů načítání včetně transform
Zdroj: autor

Při bližším pohledu je vidět že pro využití funkce “transform“ byl prohlížeč nucen si každý prvek na který bude aplikována převést na svou vlastní vrstvu, s kterou pak kompozitor dokáže pracovat. Právě tato příprava je velmi náročná na výpočetní výkon, prohlížeč musí v tomto případě vytvořit tisíce vrstev zabírající až gigabyte paměti, a stránka je při jakékoliv interakci bez reakce.

Rozdělením tohoto procesu do dvou částí pomocí manuálního vytvoření vrstvy prvku nastavením “will-change” vlastnosti na hodnotu “transform” je vidět jaká část jde do přípravy, a jaká do provedení skutečné změny na stránce.

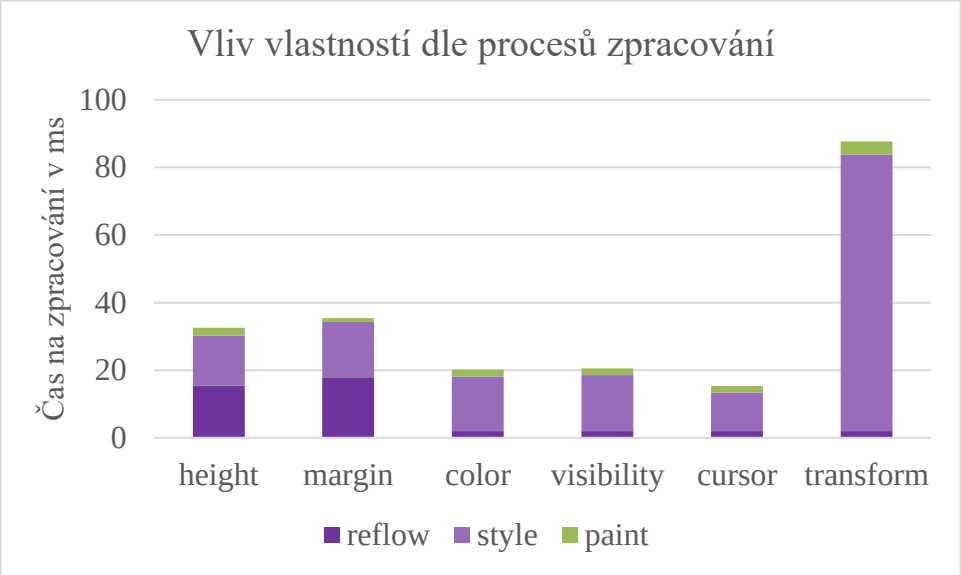
V tomto případě byla “will-change” vlastnost aplikována 3 sekundy po načtení stránky, a vlastnost “transform” až 10 sekund po načtení, aby bylo možné izolovat obě operace.



Graf 3 Vliv vlastnosti transform
Zdroj: autor

4.1.2 Měření prohlížeče Mozilla Firefox

Prohlížeč Mozilla Firefox se od prohlížeče Google Chrome liší svým vykreslovacím jádrem, a proto je vybrán pro porovnání. Prohlížeč se také liší nástrojem pro měření výkonu, a jsou proto dostupné jiné údaje reprezentující proces načítání. Zároveň je nutné zdůraznit že oba nástroje si spolu nesou nároky na výkon, které se mezi sebou mohou lišit a nejsou tak tyto hodnoty vhodné k porovnání výkonu prohlížečů mezi sebou.



Graf 4 Vliv vlastností dle procesů načítání – Firefox
Zdroj: autor

	height	**margin**	**color**	**visibility**	**cursor**	**transform**
reflow	15.4 ms	17.92 ms	1.98 ms	2 ms	2 ms	2 ms
style	14.8 ms	16.32 ms	16.1 ms	16.58 ms	11.36 ms	81.8 ms
paint	2.4 m	1.2 ms	2.04 ms	2 ms	1.98 ms	3.9 ms
Celkem	32.6 ms	35.44 ms	20.12 ms	20.58 ms	15.34 ms	87.7 ms

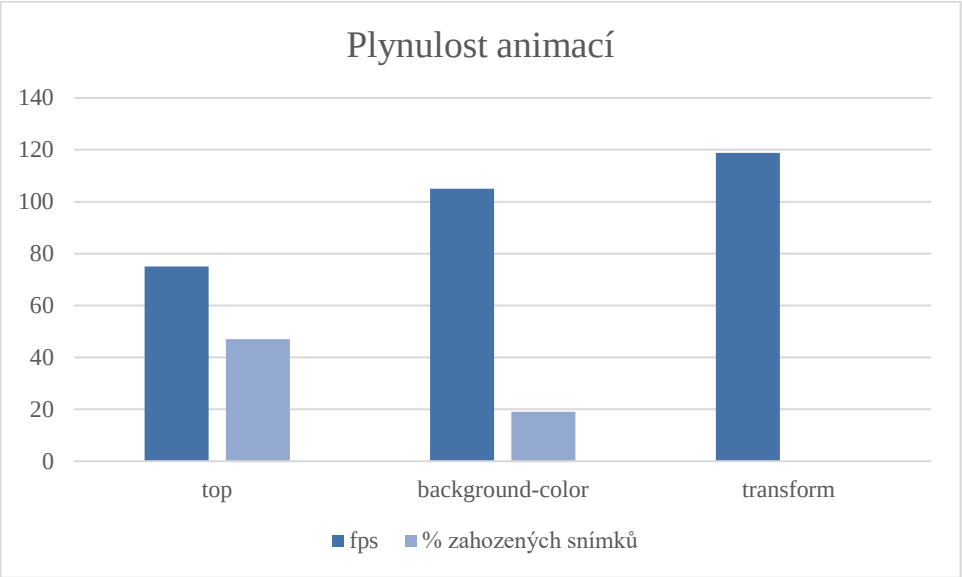
Tabulka 2 Vliv vlastností dle procesů – Firefox
Zdroj: autor

V prohlížeči Firefox je možné také pozorovat vztah mezi procesy a celkovým časem potřebným k nastavení vybrané vlastnosti. Narozdíl od Google Chrome ale Firefox neřadí při měření zpracování do 3 fází “layout“, “paint“ a “composite“, i přes to že k nim tak přistupují ve své dokumentaci a technických materiálech. Firefox ve svém vykreslovacím modelu spojuje procesy “paint“ a “composite“ do jedné skupiny “paint“, která je zpracována ve vlastním vlákně a využívá grafické akcelerace.

Vlastnost “transform“ je i v tomto prohlížeči spojena s nárokem na přípravu, avšak je provedena v řádově stejném čase a se stránkou jde dále pracovat.

4.2 Animace a vlastnost will-change

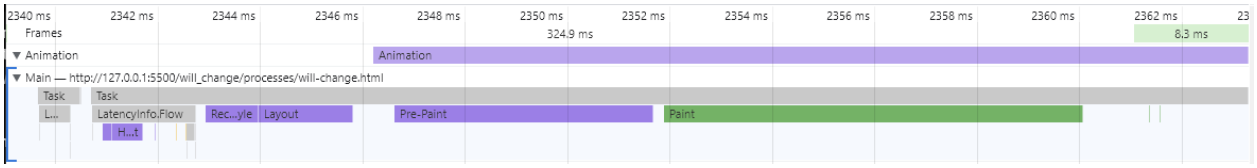
Vlastnosti dle procesu byly také otestovány při použití v animacích, u kterých byl měřen počet snímku za sekundu (FPS), a procento zahozených snímku. Informace o vykreslování stránky za běhu jsou k dispozici v prohlížeči Google Chrome pomocí nastavení "Frame Rendering Stats", které ukazuje okamžité informace o vykreslení. A také pomocí experimentálního nastavení "#show-performance-metrics-hud" které poskytuje kumulativní hodnoty.



Graf 5 Plynulost animací
Zdroj: autor

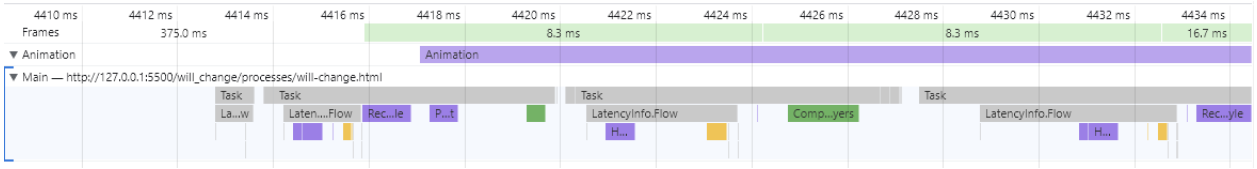
Z průběhu animací je také zaznamenán vztah plynulosti a podstoupeným procesům, které vlastnosti vyvolávají. Vzhledem k náročnosti animací je animace nastavena pouze na 21 prvcích. Při takovém množství nemusí prohlížeč při využití vlastnosti "transform" vytvářet příliš velké množství nových vrstev, a díky grafické akceleraci animace dosahuje maximálního počtu snímků displeje a není podstoupeno zahození žádných snímků.

Nastavení vlastnosti "will-change" na hodnotu "transform" před zahájením animace v prohlížeči vytvoří novou vrstvu předem, a díky tomu je tak menší zátež při samotném začátku vykreslování animovaných snímků.



Obrázek 8 Začátek vykreslení animace na stejné vrstvě
Zdroj: autor

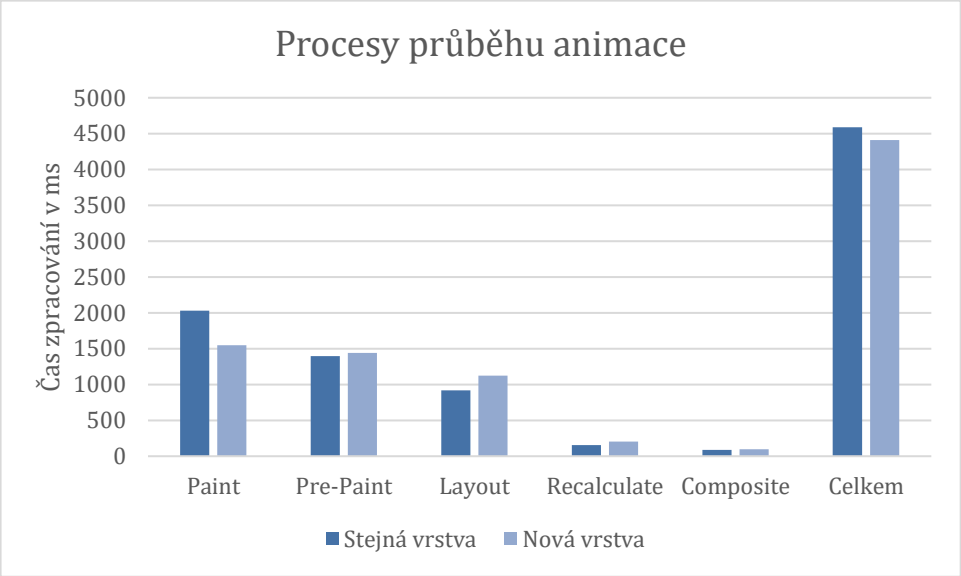
Při předem vytvořené vrstvě odpadá před startem animace nutnost znovu vykreslit obsah na existující vrstvy a snímky je tak možné začít vykreslovat dříve. Avšak ani bez jejího využití nebyl zaznamenán žádný citelný zásek či zahozený snímek.



Obrázek 9 Začátek vytvoření animace na vlastní vrstvě
Zdroj: autor

Vytvořením nové vrstvy pro animované prvky je zaznamenán i rozdíl u vlastností které využívají i procesy “layout“ a “paint“, které nejsou standartně zpracovávány na nové vrstvě pomocí vlákna kompozitoru.

Při šesti sekundovém úseku animace pomocí vlastnosti “margin-top“ k posunutí prvku na stránce byl změřen rozdíl v procesech při vytvoření nové vrstvy pro animovaný prvek. Animovaný prvek ovlivňoval další prvky v toku stránky a nebylo tak možné animaci provést pouze na izolované vrstvě.



Graf 6 Procesy průběhu animace
Zdroj: autor

I při ovlivnění ostatních prvků je vidět změna především u procesu “paint“ u kterého prohlížeč strávil méně času, a u procesů “layout“ a “recalculate“ u kterých strávil času naopak více. Za celou dobu však na výpočtu strávil o téměř 4 % méně času. Samotný průběh animace byl však pouze mírně zlepšen, z minimální hodnoty FPS 40 na 48, a z 76 % zahozených snímku na 72 %.

Při vytváření testovacích stránek pro tento test bylo obtížné vytvořit situaci při které by vlastnost “will-change“ přinesla dostatečný pozitivní vliv, lze tedy dospět k závěru který se shoduje s dokumentací této vlastnosti, že její využití by mělo být zváženo až při existenci problému, a nikoliv preventivně.

4.3 Využití vlastnosti font-display a přednačtení fontů

4.3.1 Přednačtení fontu

Pomocí přednačtení fontů lze dosáhnout zkrácení celkového času jejich načtení, díky tomuto zrychlení mohou být k dispozici při prvním vykreslení stránky, a nedojde tak k posunutí obsahu po prvním načtení.

Otestována byla kombinace deklarace fontu v lokálním či externím CSS (z Google Fonts API), a přednačtení samotného souboru fontu.

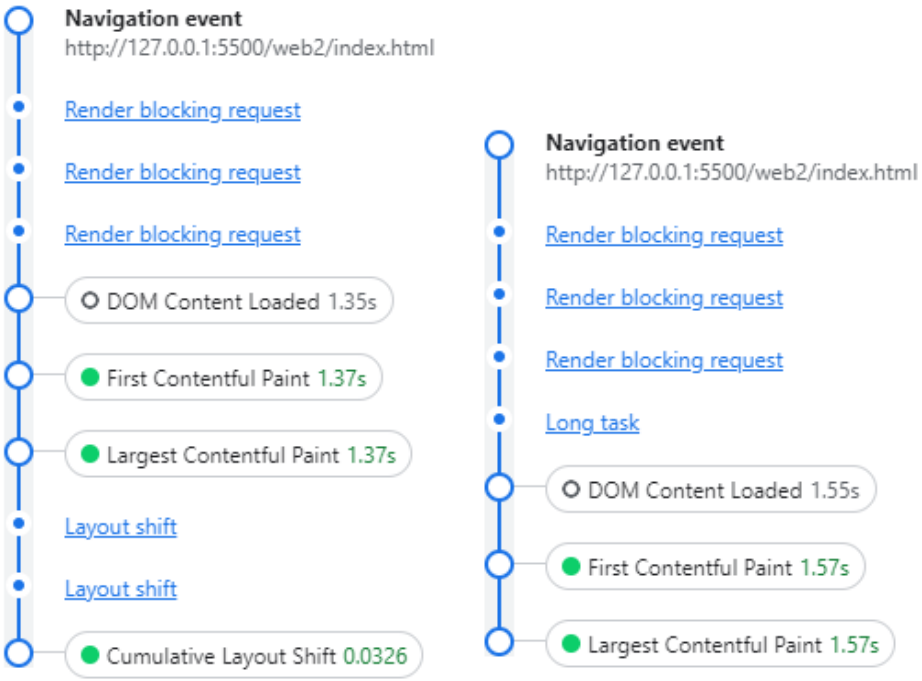
Varianta	CSS nalezeno	CSS načteno	Font nalezen	Font načten
Externí CSS + přednačtení	77 ms	2070 ms	75 ms	2370 ms
Externí CSS	88 ms	2090 ms	2210 ms	4540 ms
Lokální CSS + přednačtení	83 ms	123 ms	82 ms	2430 ms
Lokální CSS	71 ms	90 ms	108 ms	2420 ms

Tabulka 3 Přednačítání fontů
Zdroj: autor

Bez přednačtení je stahování fontu zahájeno až po tom, co si prohlížeč sestaví DOM a CSSOM, a usoudí že font bude opravdu použit, tento rozdíl lze vidět v prvních dvou řádcích tabulky. Ve druhém řádku je stahování fontu zahájeno až po stažení externího CSS obsahující jeho deklaraci, a tím je zpomaleno především o dobu která je potřebná k zahájení nového spojení se serverem, v tomto případě o vice než 2 vteřiny.

Ve všech testovaných případech byl ale font i při použití přednačtení načten až po složkách blokujících vykreslení stránky, a tím nedošlo k eliminaci nežádaného pohybu prvků které jeho zpožděné načítání způsobuje.

Při využití přednačtení fontu na stránce vytvořené v rámci předmětu web design se podařilo eliminovat nežádoucí pohyb prvků až při využití přednačtení u fontu s menším obsahem znaků. Pro zhodnocení načítání stránky pomocí byl využit nástroj “Performance insights“ v prohlížeči Google Chrome.



Obrázek 10 Rozdíl načtení stránky bez (vlevo) a s přednačtením fontu (vpravo)
Zdroj: autor

V tomto případě se po nastavení přednačtení fontů zároveň i prodlouží celkové načtení stránky, jak lze vidět na metrikách "DOM Content Loaded" a "First Contentful Paint".

Stejnou změnu v době načítání lze také zpozorovat na samotném CSS souboru který je na stránce načítán, který je při přednačtení fontu stažen o 170 ms později. Po změně jsou fonty načítány ve stejném úseku jako kritické složky stránky, které její vykreslení blokují, a tak zvýšením celkového objemu dat v tomto úseku trvá i stažení kritických složek déle.

Metriky	**Bez přednačtení**	**Přednačtení**
Načtení fontu	2.8 s	1.4 s
Layout shift	0.0326	0
Načtení CSS	0.702 s	0.88 s
First Contentful Paint	1.37 s	1.57 s

Tabulka 4 Délka načítání stránky s a bez přednačítání fontu
Zdroj: autor

4.3.2 Font-display

Na stejné stránce byl také změřen vliv vlastnosti font-display, která je nastavitelná při deklaraci fontu, při tomto testování nebylo zároveň využito přednačtení. Tato vlastnost pouze mění, jak prohlížeč k nasazení fontu přistoupí, a objem dat který je nutný přenést zůstává stejný.

V tabulce je zaznamenána hodnota posunutí obsahu při načítání (layout shift), čas vykreslení prvního obsahu (FCP), a vykreslení hlavní části obsahu (LCP).

Font-display	layout shift	FCP	LCP
block	0.0326	1.51 s	2.78 s
swap	0.0326	1.5 s	1.5 s
fallback	0.0326	1.5 s	1.55 s
optional	0	1.47 s	1.47 s

Tabulka 5 Měření vlastností font-display
Zdroj: autor

Čas pro první vykreslení se mezi hodnotami podstatně neměnil, vyčnívajícími hodnotami vyšly najevo “block” a “optional”.

Při hodnotě “block” není vykreslen žádný text v úseku před stažením nastaveného fontu, a jeho vykreslení je tak o dobu potřebnou ke stažení oddáleno oproti ostatním variantám.

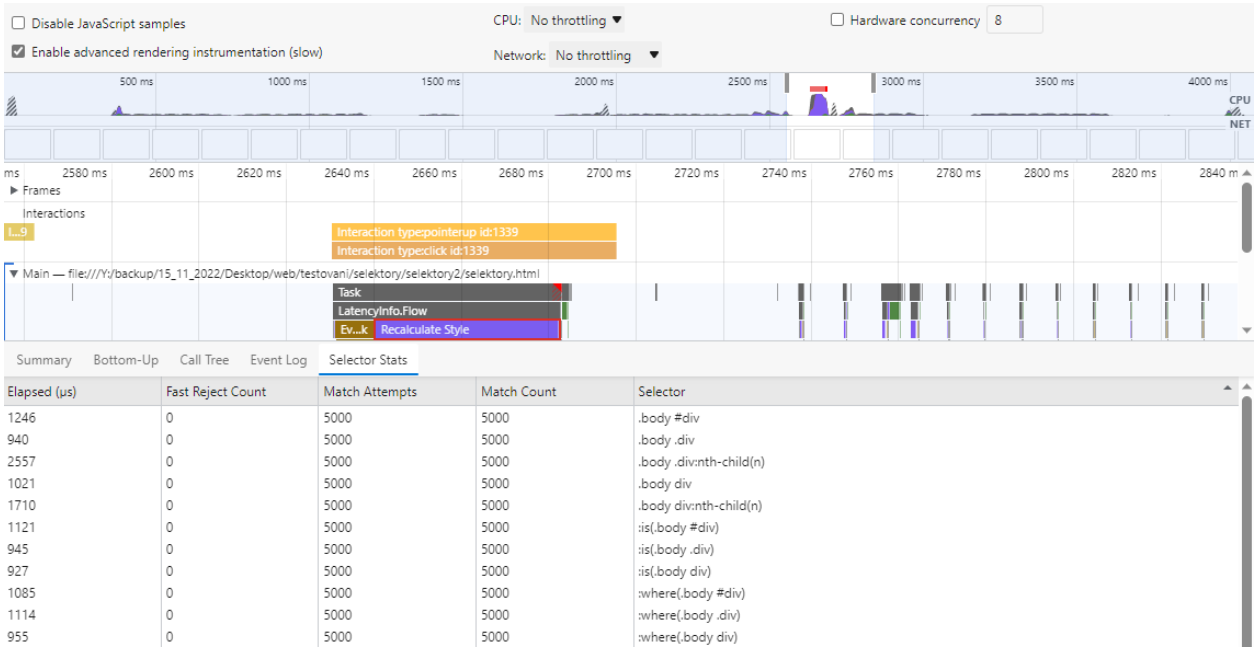
Při hodnotě “optional” prohlížeč kvůli dlouhé době na stažení fontu tak nastavený font vůbec nepoužil, a text byl od prvního momentu vykreslen alternativním fontem. Nedošlo tak k žádné změně velikosti textu, ani rozložení prvků. Podařilo se tak eliminovat hodnotu “layout shift“. Je nutné podotknout že i přes to že font nebyl při tomto vykreslení použit, tak byl prohlížečem úspěšně stáhnut a při dalším načtení byl již načten z mezipaměti prohlížeče.

Mezi hodnotami “swap” a “fallback” nebyl při tomto měření zaznamenán rozdíl, doba stažení nepřekročila hodnotu, při které by prohlížeč u hodnoty “fallback” vzdal nastavení fontu.

4.4 Selektory

Pro měření dopadu na rychlost selektorů byl použit prohlížeč Microsoft Edge, který jako jediný disponuje detailním přehledem zpracování individuálních selektorů, a ne pouze bloku “Style Recalculation”.

Informace o jejich zpracování je v prohlížeči dostupná při nahrání záznamu stránky v záložce “Performance” se zaškrtnutou možností “Enable advanced rendering instrumentation”. Po vytvoření záznamu je pro každý blok “Recalculate Style” k dispozici záložka “Selector Stats” která obsahuje informaci o délce zpracování, počtu vybraných prvků, počtu pokusů a počtu rychlých vyřazení pro každý selektor.



Elapsed (µs)	Fast Reject Count	Match Attempts	Match Count	Selector
1246	0	5000	5000	.body #div
940	0	5000	5000	.body .div
2557	0	5000	5000	.body .div:nth-child(n)
1021	0	5000	5000	.body div
1710	0	5000	5000	.body div:nth-child(n)
1121	0	5000	5000	:is(.body #div)
945	0	5000	5000	:is(.body .div)
927	0	5000	5000	:is(.body div)
1085	0	5000	5000	:where(.body #div)
1114	0	5000	5000	:where(.body .div)
955	0	5000	5000	:where(.body div)

Obrázek 11 Přehled zpracování selektorů prohlížeče MS Edge
Zdroj: autor

První testovaná stránka byla vytvořena pro obecný přehled mezi různými typy selektorů, stránka obsahuje kontainer s 100 prvky typu “div” s unikátní třídou, a 5970 menších kontainerů následující struktury.

```
<div class='container'>
    <div class='first'>
        <a data='x' data2='this is long attribute text'>link</a>
    </div>
    <a class='container__link'>link</a>
    <div class='second'></div>
    <div class='third'></div>
    <div class='fourth'></div>
    <div class='fifth'></div>
</div>
```

Obrázek 12 Kód kontaineru testovací stránky
Zdroj: autor

Pravidla se selektory nenastavují žádné vlastnosti, aby došlo k izolaci pouze vlivu vyhodnocení selektorů. Všechny selektory jsou vázány na třídu kontaineru “.body”, která je stisknutím tlačítka pomocí JavaScriptu nastavena, a vyvolá tak v prohlížeči rekalkulaci stylů na stránce, vliv selektorů je tak izolován i od počátečního načtení stránky.

První skupinou selektorů jsou třídy, kombinátory a pseudotřídy typu “nth-child”.

Selektor	Doba zpracování	Rychlé vyřazení	Pokusy	Shody
.body .container > .second	17 ms	0	5970	5970
.body .container div.second	25 ms	0	5970	5970
.body .container div:nth-child(3)	136 ms	6070	35920	5970
.body .container div:nth-last-child(4)	125 ms	6070	35920	5970
.body .container div:nth-last-child(4n)	71 ms	6070	35920	5970
.body .container__link + .second	17 ms	0	5970	5970
.body .first ~ .second	18 ms	0	5970	5970
.body .second	10 ms	0	5970	5970
.second	6 ms	0	5970	5970

Tabulka 6 Délka zpracování selektorů
Zdroj: autor

Ze získaných dat vyplývá že pro vyhodnocení selektorů typu “nth-child“ musí prohlížeč vyzkoušet podstatně větší množství prvků než u ostatních. Selektor “nth-last-child(4n)“ je také o poznání rychlejší než “nth-last-child(n)“, i přes to že podstoupí stejně pokusů.

Nejrychlejším selektorem byla pouze samotná třída, i přes to že z pohledu člověka by selektor “.body .second” měl výběr zúžit, tak se díky složitější podmínce selektor naopak výrazně zpomalí.

Selektor	Doba zpracování	Rychlé vyřazení	Pokusy	Shody
.body div	10 ms	0	5000	5000
.body div:nth-child(n)	30 ms	0	5000	5000

Tabulka 7 Rozdíl vybraných selektorů
Zdroj: autor

Selektor “nth-child(n)” je z měření 3x pomalejší než selektor pouze cílící na typ i v případě kdy podstoupí stejné množství pokusů, a vybere stejné množství prvků.

Druhou skupinou selektorů jsou selektory cílící na atributy, při kterém by mělo jít o prosté textové porovnání dvou či více hodnot.

Selektor	Doba zpracování	Rychlé vyřazení	Pokusy	Shody
.body a[data2$=”text”]	21 ms	0	5970	5970
.body a[data2*=”text”]	32 ms	0	5970	5970
.body a[data2=”this is long attribute text”]	18 ms	0	5970	5970
.body a[data2^=”this”]	21 ms	0	5970	5970
.body a[data=”x”]	43 ms	0	5970	5970
.body a[data|=”x”]	28 ms	0	5970	5970

Tabulka 8 Měření selektorů cílících na atribut
Zdroj: autor

Překvapujícím výsledkem je selektor “a[data=”x”]”, i přes to že se jedná o nejkratší text a přesnou shodu, tak vychází znatelně nejpomaleji. Selektory které využívají výběru jiné než přesné shody jsou pro delší text předvídatelně pomalejší, nejhůře pak vychází tzv. “wildcard“ selektor.

Poslední skupinou této stránky jsou psuedoselektory “:is()” a “:where()” v porovnání s tradičním selektorem.

Selektor	Doba zpracování	Rychlé vyřazení	Pokusy	Shody
:is(.container div)	73 ms	0	35920	29850
:where(.container div)	105 ms	0	35920	29850
.container div	56 ms	6070	35920	29850

Tabulka 9 Měření nových pseudoselektorů
Zdroj: autor

Z tohoto měření vycházel selektor “:where()” témeř 2x pomaleji než klasický kombinační selektor, i při využití stejné podmínky.

Pro další měření zaměřené právě na tyto pseudoselektory byla použita stránka s jednoduchou strukturou pouze 5000 prvků typu “div”.

Selektor	Doba zpracování	Rychlé vyřazení	Pokusy	Shody
.body #div	9 ms	0	5000	5000
.body .div	9 ms	0	5000	5000
.body div	9 ms	0	5000	5000
:is(.body #div)	10 ms	0	5000	5000
:is(.body .div)	9 ms	0	5000	5000
:is(.body div)	10 ms	0	5000	5000
:where(.body #div)	11 ms	0	5000	5000
:where(.body .div)	10 ms	0	5000	5000
:where(.body div)	10 ms	0	5000	5000

Tabulka 10 Ověření rozdílu pseudoselektorů na jednoduché stránce
Zdroj: autor

Na této stránce se neprojevil rozdíl změřen na předchozí stránce, a nejedná se tak o univerzální nárok na výkon pro tyto selektory.

V dalším měření při kterém na stránce nebyla nastavena třída kontaineru, a neexistoval tak žádný prvek který by selektor splnil bylo zjištěno že pseudotřídy ":select()" a ":where()" nedokáží prvky rychle vyřadit.

Selektor	Doba zpracování	Pokusy	Shody	Rychlé vyřazení
.body div	16 ms	5000	5000	0
:is(.body div)	51 ms	0	5000	0
:where(.body div)	56 ms	0	5000	0

Tabulka 11 Měření selektorů bez shody
Zdroj: autor

V takové situaci trvá jejich vyhodnocení až třikrát déle oproti klasickému selektoru. I nevyužitá pravidla se selektory tohoto typu mohou tak mít na dost obsáhlé stránce mezi selektory relativně vysoký dopad.

Obdobně jsou na tom i selektory typu "last-child", které sice dokáží prvky rychle vyřadit, ale stále jsou v takové situaci více než 5x pomalejší než selektory cílené na třídu.

V poslední měření selektorů bylo zasazení výše popsaného kontaineru do struktury typické stránky, a vytvoření skupinky naivních selektorů cílené na tento kontainer. Stránka i s vloženým kontainerem obsahovala 125 prvků.

Selektor	Rychlé vyřazení	Pokusy	Shody
.container + a	0	20	0
.container div:nth-child(3)	20	25	1
.container__link ~ div	0	25	4
.container :where(.first, .second)	0	2	2

Tabulka 12 Selektory k kontaineru v existující stránce
Zdroj: autor

Tyto jednoduché selektory vytvořené pro tento kontainer při vyhodnocení podstoupily až 25 pokusů, to je zhruba 1/5 všech prvků na stránce. V případě stránky s velikým počtem prvků, u které jsou za běhu prováděné operace které podmíní prohlížeč provést rekalkulaci stylů, mohou takto cílené selektory mít již znatelný vliv, je proto nutné si uvědomit že selektory jsou vždy vyhodnocovány na celé stránce podle klíčového selektoru.

4.5 Minifikace a komprese

Při měření minifikace a komprese byla opět využita stránka vytvořena v rámci předmětu webdesign, která obsahuje CSS strukturu reálné webové stránky. Pro minifikaci byl použit online nástroj dostupný na https://www.minifier.org/, a pro gzip kompresi https://www.zickty.com/texttogzip/.

U každé varianty použití minifikace či komprese bylo změřené načtení samotného CSS souboru, a také jak se projeví na načtení celé stránky.

Varianta	Znaky	Velikost	Zmenšení	Načtení souboru	FCP
Původní	20668	21 Kb	0 %	144.3 ms	1.34 s
Minifikace	16461	16,8 Kb	20 %	106.1 ms	1.32 s
Komprese	7568	7.9 Kb	62 %	58 ms	1.27 s
Minifikace + Komprese	6928	7.3 Kb	65 %	43.2 ms	1.26 s

Tabulka 13 Měření minifikace a komprese
Zdroj: autor

Použitím minifikace se snížila velikost souboru o 20 %, a použitím komprese až o 62 %. Minifikace v případě CSS není tak účinná jako například u JavaScriptu, nelze totiž optimalizovat klíčová slova která jsou závislé na struktuře html stránky, názvy tříd, atributů či prvků.

O téměř stejné procentní zlepšení byla změřená i doba načítání samotného CSS souboru, která se pak pozitivně projevila při zhodnocení načtení celé stránky pomocí metriky “First Contentful Paint”.

4.6 Optimalizace pro Above the fold obsah

Posledním měřením je optimalizace pro above the fold obsah stránky, pro účely měření byla opět použita stránka vytvořená v rámci předmětu webdesign. Základem pro optimalizaci byl přehled ve vývojářském okně Google Chrome v záložce “coverage”, který vizualizuje, jaká procentuální část načteného CSS souboru byla použita pro vykreslení aktuální stránky, a také přímo označí i použitou část kódu.

Tento přehled však není perfektní, nijak nezohledňuje prvky, které nebyly bez posunu vykreslené, nebo označí celý “media query“ blok jako nutný, i v případě kdy obsahuje nepoužitý styl. Přehled také označil “@keyframes” bloky jako nevyužité, i přes to že byly skutečně vykreslené.

Dle tohoto přehledu bylo pro tuto stránku označeno pouze 53,3 % souboru jako použitého pro vykreslení.

Pro optimalizované CSS byl vybrán vhodně označený kód z přehledu, z kterého byl navíc manuálně vybrán pouze kód, který cílí na okamžitě viditelnou část stránky. Místo původního souboru tak vznikly soubory 2, první cílící na viditelnou část stránky, který při načítání blokuje vykreslení, a druhý který obsahuje zbytek stylů a načítání neblokuje.

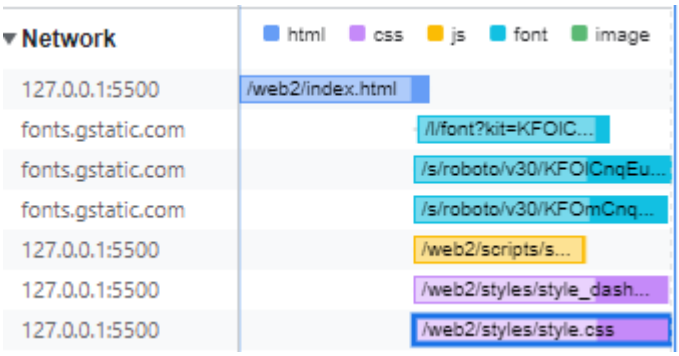
Soubory	**Velikost**	**Využití**
Původní soubor	22,50 Kb	53.3 %
Nový soubor 1	11,38 Kb	83.3 %
Nový soubor 2	11,22 Kb	23.1 %

Tabulka 14 Velikost a využití CSS souborů
Zdroj: autor

Pro načtení souboru bez blokování byl použit následující vzor kódu, který souboru po načtení přiřadí JavaScriptem vztah stylu.

```
„<link rel="preload" href="styles.css" as="style"
onload="this.onload=null;this.rel='stylesheet'">”
```

Při měření průběhu načítání takto optimalizované stránky však nedošlo k výraznému rozdílu, to je způsobeno téměř identickou velikostí výsledných souborů, které byly načteny ve stejný čas, a jejich celkový objem byl velmi podobný souboru původnímu.



Obrázek 13 Paralelní načítání CSS po rozdělení
Zdroj: autor

Při dalším testu proto bylo původní CSS stránky doplněno větším množstvím obsahu, pro simulaci obsáhlejší stránky. V takovém případě už měly výsledné CSS soubory rozdílnou velikost, a lišil se tak čas jejich načtení.

Celkový CSS soubor o velikosti 44 KB byl tak rozdělen na 2 soubory o velikosti 33 KB a 12 KB, ten větší byl na stránku načítán bez blokování.

Varianta	Doba načtení blokujícího CSS	FCP
Celý soubor	1.59 s	1.70 s
Rozdělený soubor	1.43 s	1.52 s
Styl v HTML	x	1.22 s

Tabulka 15 Porovnání načítání různých konfigurací
Zdroj: autor

Pro porovnání je v tabulce také načítání stránky s variantou, kde je styl vložen do <style> bloku v samotném HTML souboru. Tato varianta byla jednoznačně nejrychlejší, prohlížeč nebyl nucen navázat další spojení pro stažení CSS souboru, a CSS tak už nebylo limitující složkou vykreslení.

Při tomto měření už byl zaznamenán i rozdíl mezí rozdělením CSS, a původním souborem, a to dobou načtení blokujícího CSS a metriky “First Contentful Paint“. Při rozdělených souborech stačí načíst menší blokující CSS a může začít vykreslení, zatímco původní varianta čeká i na stažení nepotřebných stylů.

5 Zhodnocení výsledků

Celkem bylo provedeno měření pro šest oblastí, ve kterých by CSS mohlo mít značný vliv na průběh načítání stránky, na základě naměřených výsledků je možné u každé zhodnotit její vliv, a určit či je nebo není vhodné se na ni zaměřit při optimalizaci stránky.

5.1.1 Vlastnosti dle procesů zpracování

V této oblasti byl naměřen značný rozdíl mezi skupinami vlastností podle jejich procesů, a to v obou testovaných prohlížečích. Pro typickou statickou stránku je většina vlastností nastavena pouze jednou, a to při jejím prvním vykreslení, a celkový vliv na načtení stránky je omezený.

Většinu vlastností také nelze jednoduše zaměnit za optimálnější verzi bez změny na podobě vykresleného obsahu, a proto zde není příliš prostoru pro optimalizaci. Je ale namístě si být vědom které vlastnosti jsou považovány za drahé, a při jejích výskytu na velkém množství prvků zvážit jejich vliv.

5.1.2 Animace a vlastnost will-change

Výše zmíněné vlastnosti byly otestované také v animacích, ve kterých je pro uživatele podstatně jednoduší si všimnout zpomalení či záseků. Zde měly drahé vlastnosti velmi znatelný vliv, a k perfektně plynulým animacím došlo pouze při použití vlastností využívajících vlákna kompozitoru.

Ačkoliv zde stále platí že vlastnosti nelze jednoduše zaměnit za “lehčí“, využití drahých vlastností v animacích by mělo být důrazně zváženo, a mělo by co nejvíce být využito graficky akcelerovaných transformací pomocí vlastnosti “transform“.

U vlastnosti “will-change“ určené pro optimalizaci nebyl změřen podstatný pozitivní dopad na výkon v oblasti animací, a nelze proto definitivně říci za jaké situace a do jaké míry vlastnost pomůže. Doporučením je proto vlastnost zkusit použít až v případě nalezených problémů s výkonem přechodů a animací.

5.1.3 Font-display a přednačítání fontů

Při měření v oblasti přednačítání fontů byl naměřen značný pozitivní rozdíl, a podařilo se na vybrané testovací stránce eliminovat nechtěného posunu obsahu. Přednačítání je velmi jednoduché na implementaci, a v případě že jsou přednačteny pouze skutečně použité fonty nedojde k žádnému negativnímu vlivu.

Vlastnost font-display je spíše kompromisem než přímým zlepšením, při jejím použití je nutné zvážit jakou funkci má blok textu který bude vlastností ovlivněn. Defaultní chování nastavené prohlížečem může být ale v řadě případů již optimální.

5.1.4 Selektory

Při měření selektorů byl změřen rozdíl mezi typy selektorů, ale až na selektory typu nth-child byly jejich nároky řádově stejné. V absolutním čase však měly selektory nízký vliv i na velmi obsáhlé stránce.

Pokud není blok rekalkulace stylů při vykreslení nebo reakci na vstup uživatele příliš veliký tak není z pohledu výkonu potřeba selektorům věnovat hlubší pozornost. Je však dobrým zvykem nevytvářet příliš složité kombinace, a používat třídu jako klíčový selektor. Takové selektory mají mimo nízké výpočetní nároky také dobrou čitelnost a nevytváří problémy se specificitou.

5.1.5 Minifikace a komprese

Minifikace a komprese je oblast díky které je možné zlepšit rychlost načítání velmi jednoduše a bez nechtěných vedlejších účinků, v případě že byla doba stažení CSS souboru limitujícím faktorem při načítání.

5.1.6 Optimalizace pro Above the Fold obsah

Optimalizace pro Above the Fold obsah je komplexnější na implementaci, avšak také dokáže při načítání pomoci. Efektivita této optimalizace ale záleží na struktuře stránky a CSS souboru, a k pozitivnímu výsledku nemusí dojít, jak bylo také v jednom z případů naměřeno.

Mimo nejistotu pozitivního vlivu je optimalizaci nutné provést manuálně, a vyžaduje znalost struktury stránky. Rozdělení souboru pro optimalizaci také ovlivní jeho čitelnost a pohodlnost následujících úprav. Optimalizace tak může být vhodná pro konkrétní stránku v případě, že je identifikováno nízké využití celého načítaného CSS.

6 Závěr

V teoretické části práce byla představena problematika rychlosti načítání webových stránek, jaké má dopady a jak je vyhodnocována. Dále bylo představeno CSS zejména po technické stránce, a společně s jeho hlavním využitím bylo vytýčeno několik oblastí které se značně podílí na procesu načítání stránky, nebo mají jiný viditelný dopad na výkon. Společně s tím bylo také představeno několik způsobů jakýmž je možné CSS optimalizovat po výkonové stránce.

V praktické části práce byly na základě získaných informací vybráno šest oblastí vhodných k detailnímu měření, a zhodnocení jejich vlivu na rychlost načítání či plynulost fungování stránky. Pro účely měření bylo vytvořeno několik stránek pro izolaci specifických oblastí, a také bylo využito stránky vytvořené během studia která reprezentovala složení reálné stránky. Při měření bylo naraženo na obtížnost získání dat z prohlížečů z důvodu rozdílu vestavěných vývojářských nástrojů a prezentace jejich naměřených hodnot.

Vybrané oblasti byly nakonec zhodnoceny na základě provedených testů a měření, a o každé bylo rozhodnuto do jaké míry je vhodné se na ni zaměřit s cílem zlepšení výkonu.

7 Seznam použitých zdrojů

1. HOGAN, Lara Callender. Designing for Performance: Weighing Aesthetics and Speed. Kalifornie: O'Reilly Media, 2014. ISBN 1491902515.
2. BERMES, Barbara. Lean Websites. Melbourne: SitePoint, 2015. ISBN 9780992279462.
3. Lighthouse performance scoring. *Web.dev* [online]. Kalifornie: Google, 2019 [cit. 2022-08-31]. Dostupné z: https://web.dev/performance-scoring/#metric-scores
4. User experience improvements with page speed in mobile search. *Google Search Central* [online]. Kalifornie: Google, 2019 [cit. 2022-08-31]. Dostupné z: https://developers.google.com/search/blog/2019/04/user-experience-improvements-with-page
5. A faster FT.com. *Medium* [online]. Kalifornie: FT Product & Technology, 2016 [cit. 2022-08-31]. Dostupné z: https://medium.com/ft-product-technology/a-faster-ft-com-10e7c077dc1c
6. The 2022 Google Algorithm Ranking Factors. *Best SEO Agency* [online]. Kalifornie: First Page Sage, 2021 [cit. 2022-08-31]. Dostupné z: https://firstpagesage.com/seo-blog/the-google-algorithm-ranking-factors/
7. Single-page application vs. multiple-page application. *Medium* [online]. Kalifornie: Neoteric, 2021 [cit. 2022-08-31]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58
8. Page Weight Matters. *Chris Zacharias* [online]. Chris Zacharias, 2012 [cit. 2022-08-31]. Dostupné z: https://blog.chriszacharias.com/page-weight-matters
9. LIE, Håkon Wium a Bert BOS. *Cascading Style Sheets: Designing for the Web*. 3rd ed. Boston: Addison Wesley Professional, 2005. ISBN 0321193121.
10. What is CSS?. *MDN Web Docs* [online]. Kalifornie: Mozilla Corporation, 2020 [cit. 2022-08-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS
11. DOWDEN, Michael a Martin DOWDEN. Architecting CSS: The Programmer's Guide to Effective Style Sheets. New York: Apress, 2020. ISBN 978-1484257517.

12. Syntax. *MDN Web Docs* [online]. Kalifornie: Mozilla Corporation, 2020 [cit. 2022-08-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Syntax
13. Cascade. *MDN Web Docs* [online]. Kalifornie: Mozilla Corporation, 2020 [cit. 2023-02-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade
14. Specificity. *MDN Web Docs* [online]. Kalifornie: Mozilla Corporation, 2020 [cit. 2023-02-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity
15. Inheritance. *MDN Web Docs* [online]. Kalifornie: Mozilla Corporation, 2020 [cit. 2023-02-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/inheritance
16. Inside look at modern web browser (part 3). *Chrome Developers* [online]. Kalifornie: Google, 2018 [cit. 2022-08-31]. Dostupné z: https://developer.chrome.com/blog/inside-browser-part3/
17. CSS Style Invalidation in Blink. *Google Git* [online]. Kalifornie: Google, 2014 [cit. 2022-08-31]. Dostupné z: https://chromium.googlesource.com/chromium/src/+/master/third_party/blink/renderer/core/css/style-invalidation.md
18. SKVORC, Bruno, Craig BUCKLER, Maria Antonietta PERNA a Ivan CURIC. Front-end Performance. Melbourne: SitePoint, 2017. ISBN 9781925836103.
19. Animation performance and frame rate. *MDN Web Docs* [online]. Kalifornie: Mozilla Corporation, 2020 [cit. 2022-08-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Performance/Animation_performance_and_frame_rate
20. WAGNER, Jeremy. Web Performance in Action: Building Faster Web Pages. New York: Manning Publications, 2017. ISBN 1617293776.
21. Optimize WebFont loading and rendering. *Web.dev* [online]. Kalifornie: Google, 2019 [cit. 2022-08-31]. Dostupné z: https://web.dev/optimize-webfont-loading/
22. CSS Fonts Module Level 4. *W3C Working Draft* [online]. Massachusetts: W3C, 2021 [cit. 2023-03-10]. Dostupné z: https://www.w3.org/TR/css-fonts-4/#font-display-desc

23. CSS Will Change Module Level 1. *W3C on GitHub* [online]. Massachusetts: W3C, 2022 [cit. 2022-01-14]. Dostupné z: https://w3c.github.io/csswg-drafts/css-will-change/#will-change
24. A faster FT.com.[Obrázek] Medium [online]. Kalifornie: FT Product & Technology, 2016 [cit. 2022-08-31]. Dostupné z: https://medium.com/ft-product-technology/a-faster-ft-com-10e7c077dc1c
25. DOWDEN, Michael a Martin DOWDEN. Architecting CSS: The Programmer's Guide to Effective Style Sheets. [Obrázek] New York: Apress, 2020. ISBN 978-1484257517.
26. A comprehensive guide to using CSS Grid. [Obrázek] Creative Bloq [online]. Bath: Ahmed Abuelgasim, 2021 [cit. 2022-08-31]. Dostupné z: https://www.creativebloq.com/advice/a-comprehensive-guide-to-using-css-grid
27. CSS Box Model. [Obrázek] Tutorial Republic [online]. Bath [cit. 2022-08-31]. Dostupné z: https://www.tutorialrepublic.com/css-tutorial/css-box-model.php
28. BERMES, Barbara. Lean Websites.[Obrázek] Melbourne: SitePoint, 2015. ISBN 9780992279462.
29. Performance best practices for Firefox front-end engineers [Obrázek]. Firefox source docs [online]. Kalifornie: Mozilla Corporation [cit. 2023-03-10]. Dostupné z: https://firefox-source-docs.mozilla.org/performance/bestpractices.html
30. Optimize WebFont loading and rendering. [Obrázek] Web.dev [online]. Kalifornie: Google, 2019 [cit. 2023-03-10]. Dostupné z: https://web.dev/optimize-webfont-loading/

8 Seznam obrázků, tabulek a grafů

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam grafů