PAGESPEED.CZ

*Na rychlosti záleží*

# Checklist rychlosti webu 2021

# OBSAH

Pojďme udělat rok 2021... rychlým! V českém překladu přinášíme kontrolní seznam pro optimalizaci rychlosti frontendu webů. Obsahuje téměř vše, co potřebujete vědět, abyste dnes mohli na webu zajistit rychlou uživatelskou zkušenost – od metrik, přes nástroje až po vývojářské techniky.

*Tento kontrolní seznam je překladem Front-End Performance Checklist 2021.*
*Děkujeme Smashing Magazine.*

# 1

# Příprava: Plánování a metriky

- **Tvořte „performance culture“, kulturu podporující rychlost webu.**

  Rychlost webu nemůže být dlouhodobě udržitelná bez investice. Prostudujte si běžné stížnosti přicházející na zákaznickou podporu a zjistěte, jak zvýšení rychlosti může pomoci zmírnit některé z těchto problémů. Vytvořte případovou studii s reálnými daty a obchodními metrikami, přizpůsobenou vaší společnosti. Už při návrhu webu si naplánujte postup načítání webu a případné kompromisy.

- **Buďte o 20 % rychlejší než váš nejrychlejší konkurent.**

  Shromažďujte data o zařízeních, která návštěvníci vašeho webu využívají. Upřednostňujte skutečná zařízení před simulacemi. Vyberte si mobil Moto G4/G5 Plus; zařízení Samsung střední třídy (Galaxy A50, S8); dobré zařízení střední třídy jako Nexus 5X, Xiaomi Mi A3 nebo Xiaomi Redmi Note 7 a pomalé zařízení jako Alcatel 1X nebo Cubot X19. Alternativně můžete mobilní OS emulovat na počítači testováním na omezené rychlosti sítě (např. 300 ms RTT, 1,6 Mb/s download, 0,8 Mb/s upload) s omezeným CPU (5 × zpomalení). Poté přepněte na běžné 3G, pomalé 4G (např. 170 ms RTT, 9 Mb/s download, 9 Mb/s upload) a Wi-Fi. Posbírejte data, sestavte tabulku, uberte z výsledků 20 % a stanovte si své cíle (*„performance budgets“, limity rychlosti*).

- **Vyberte správné metriky.**

  Ne každá metrika má z pohledu optimalizace stejnou hodnotu. Prostudujte si, na kterých metrikách záleží nejvíce: obvykle budou souviset s tím, jak rychle se začnou vykreslovat *nejdůležitější pixely* a jak rychlá je odezva vstupu. Zaměřte se tedy na načítání stránky podle priorit vašich návštěvníků. Obvykle záleží na *délce času do interakce* (Time To Interactive – TTI), *první nečinnost procesoru (First Input Delay – FID*), *době vykreslení „hero“ elementu*, *největším vykreslení obsahu* (Largest Contentful Paint – LCP), *celkové době blokování* (Total Blocking Time – TBT) a *kumulativním posunu layoutu* (Cumulative Layout Shift – CLS). Nikoliv na: První smysluplné vykreslení (First Meaningful Paint).

- **Nastavte profily testování: „čistý“ a „zákaznický“.**

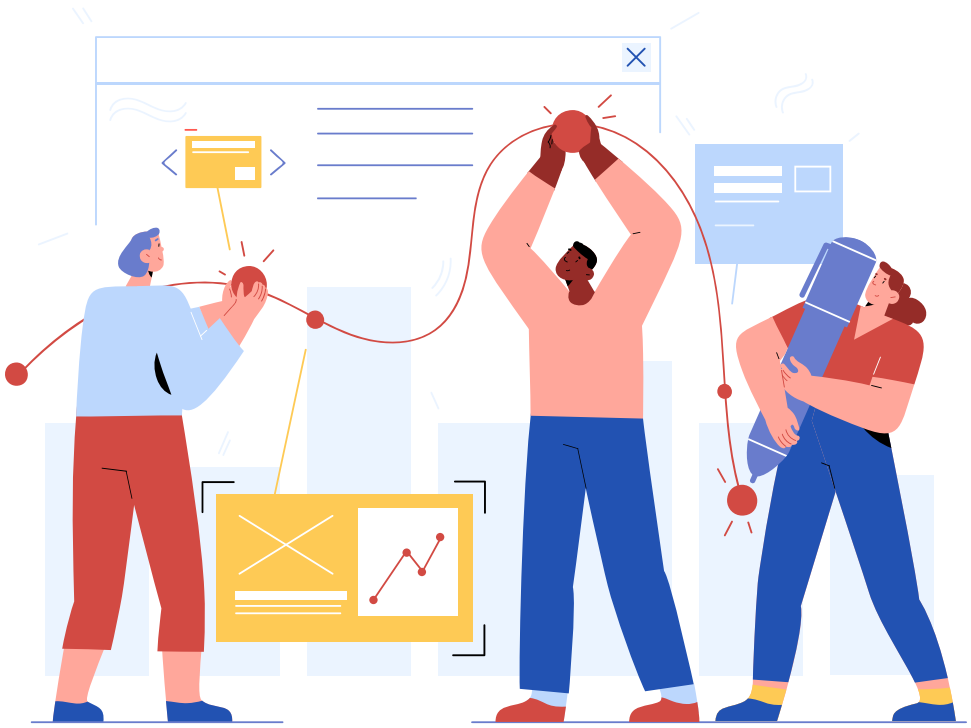
  Vypněte antivirové programy a úlohy vytěžující CPU na pozadí, zastavte přenosy dat na pozadí a otestujte s „čistým“ uživatelským profilem bez rozšíření prohlížeče, abyste zabránili zkreslení výsledků. Prostudujte si, která rozšíření vaši zákazníci používají a otestujte také speciální profil „zákazníka“.

- **Sdílejte Checklist rychlosti webu se svými kolegy.**

  Ujistěte se, že každý člen týmu zná tento Checklist rychlosti webu. Každé rozhodnutí má vliv na výkon a vašemu projektu výrazně prospěje, pakliže se vlastnictví rozdělí napříč celým týmem. Současně mapujte rozhodnutí o návrzích a stavte je proti výkonnostnímu rozpočtu.

# 2

# Nastavení realistických cílů



- **Doba odezvy 100 milisekund, 60 snímků za vteřinu.**

  Každý rámeček animace by měl být vykonán během 16 milisekund — v ideálním případě 10 milisekund, čímž dosáhnete požadovaných 60 snímků za vteřinu (1 sekunda ÷ 60 = 16,6 milisekund). Buďte optimista a uvážlivě využijte doby nečinnosti; u náročných činností, jakými jsou animace, je lepší nedělat na stránce nic jiného. Tam kde to nejde, dělat pouze absolutní minimum. Odhadovaná latence vstupu by měla být nižší než 50 ms. Využijte doby nečinnosti s přístupem *Idle Until Urgent*.

- **LCP < 2,5 sekund; FID < 100 ms; CLS < 0,1; *TTI* < 5 sekund na 3G.**

  Zvážíte-li jako výchozí zařízení mobil se systémem Android v ceně kolem 4 tis. Kč na pomalé 3G síti, emulovaný při rychlosti přenosu 400 ms RTT a 400 kB/s, zaměřte se na *dobu času do interakce (TTI)* < 5 sekund a na opakované návštěvy pod 2–3 sekund. Zaměřte se na *největší vykreslení obsahu* < 2,5 sekund a minimalizujte *celkovou dobu blokování (TBT)* a *kumulativní posun layoutu (CLS)*. Snažte se o snížení těchto hodnot na minimum.

- **Strop velikosti souboru < 170 kB.**

  Prvních 14–16 kB HTML kódu je nejkritičtější — jde o jedinou část, která je doručená v prvním balíčku přenášeném po síti („first roundtrip“). Abyste dosáhli výše stanovených cílů, pracujte s velikostí souborů o maximální gzipované velikosti 170 kB (0,7 – 0,8 MB dekomprimované). Ujistěte se, že vaše limity pro rychlost se mění podle síťových podmínek a hardwarových omezení.

# 3

# Definice vývojového prostředí

- **Dobře vyberte a nastavte své sestavovací nástroje.**

  Příliš se nezabývejte tím, jestli jde o „cool“ nástroj nebo ne. Pokud rychle dosahujete výsledků a nemáte problémy s udržováním vašeho procesu tvorby, pak je vše v pořádku. Výjimkami mohou být Webpack nebo Parcel, které poskytují užitečné optimalizační techniky jako je rozdělení kódu (code-spliting). Pokud tyto techniky ještě nepoužíváte, pak se podívejte na již zmíněný code-splitting a na tree-shaking.

- **Použijte ve výchozím stavu progresivní vylepšení (progressive enhancement).**

  Nejprve navrhněte a vytvořte základní rozhraní, aby web splňoval základní funkci. Poté vylepšete rozhraní pomocí pokročilých funkcí pro různé prohlížeče, čímž vytvoříte odolné a stabilní rozhraní pro různé uživatele s různými zařízeními. Pokud váš web běží rychle na pomalém zařízení, se špatnou obrazovkou a ve špatném prohlížeči v neoptimální síti, pak poběží rychleji i na rychlém zařízení, s dobrým prohlížečem a na slušně rychlé síti.

- **Nastavte laťku výkonu vysoko.**

  JavaScript obnáší největší náklady na uživatelskou zkušenost. S limitem 170 kB, který už obsahuje kritická data a cesty – HTML / CSS / JavaScript, router, správa stavu, nástroje, rozhraní a aplikace – důkladně prozkoumejte časové náklady na síťový přenos, parsování, kompilaci a náklady na dobu běhu frameworku.

- **Vyhodnoťte každý framework a každou závislost.**

  Ne každý projekt potřebuje framework, ne každá jednostránková aplikace (single page application – SPA) potřebuje načíst framework. Vybírejte rozvážně; vyhodnoťte JS třetích stran zkoumáním funkcí, přístupnosti, stability, rychlosti, ekosystémů balíčků, křivky učení, dokumentace, nástrojů, záznamů, týmu, kompatibility a zabezpečení. Next.js (React), Gatsby (React), Vuepress (Vue) a Preact CLI poskytují slušné výchozí hodnoty pro rychlé načítání na průměrném mobilním hardwaru.

- **Vybírejte moudře: React, Vue, Angular, Ember a spol..**

  Ujistěte se, že framework podle vašeho výběru poskytuje vkreslování na straně serveru (SSR) nebo předběžné vykreslování (prerendering). Než se rozhodnete, změřte na mobilních zařízeních časy vykreslování jak na serveru, tak u klienta. Je důležité znát „šrouby a matice“ frameworku, na který se budete spoléhat. Podívejte se na návrhový vzor PRPL a architekturu App Shell. Dobrými možnostmi jsou Preact, Inferno, Vue, Svelte, Alpine, Polymer.

- **Optimalizujte výkon vašich API.**

  API se snadno může stát úzkým hrdlem, když k němu v jeden čas přistupuje více služeb. Zohledněte možnosti využití jazyka GraphQL, s jehož pomocí můžete spouštět dotazy nad typovým systémem, který vytvoříte nad vašimi daty. Narozdíl od RESTu, GraphQL dokáže získat všechna data v jednom dotazu, aniž by jich stáhnul zbytečně moc nebo naopak málo, což se často děje v RESTových architekturách.

- **Zkusíte technologie Google AMP nebo Facebook Instant Articles?**

  I bez těchto technologií můžete mít dobrý výkon, ale AMP nabízí velmi dobrý výkonnostní základ a globální CDN. Instant Articles vám pomohou získat především viditelnost na Facebooku a až 4x rychlejší načtení, než běžné mobilní weby.

- **Vybírejte vaši CDN moudře.**

  Zvažte, jak moc jste závislí na dynamických datech. Můžete minimálně část vašeho obsahu generovat staticky, nahrát na CDN a tím se vyhnout nadbytečným dotazům do databáze (JAMStack). Zkontrolujte, že CDN provádí kompresi obsahu a konverzi obrázků (např. optimalizace a změnu velikosti nebo formátu) a že podporuje serverové workery.

# 4

# Optimalizace souborů

- **Použijte Brotli pro kompresi.**

  Brotli je nový bezeztrátový datový formát, podporovaný ve všech moderních prohlížečích. Je efektivnější než GZip a Deflate. Komprimace je velmi pomalá, ale dekomprimace rychlá. Předkomprimujte statické assety pomocí Brotli a Gzipu na nejvyšší úroveň, HTML komprimujte dynamicky za běhu pomocí Brotli (level 1-4). Zkontrolujte také podporu Brotli na vašem CDN. Ujistěte se, že server zpracovává „content negotiation" pro Brotli nebo Gzip správně.

- **Použijte responsivní obrázky, AVIF a WebP.**

  Pokud je to možné, použijte responzivní obrázky s vlastnostmi srcset, sizes, a image-set. Použijte AVIF a WebP formáty s a JPEG/PNG jako fallback nebo detekujte podporu pomocí Accept hlavičky. Pozn. s WebP snížíte datový objem, ale s JPEG můžete vylepšit díky vlastnosti „progresivního vykreslování" vnímanou rychlost.

- **Jsou obrázky správně optimalizovány?**

  Použijte mozJPEG pro kompresi JPEG, SVGO pro kompresi SVG, Pingo pro PNG nebo Squoosh pro všechny z nich (včetně AVIF). Pro kontrolu účinnosti vašeho responzivního markupu můžete použít nástroj *imaging-heap*. U kritických obrázků použijte progresivní JPEG a rozmažte zbytečné řásti (aplikováním Gaussova rozostření) a odstraňte kontrast (můžete jej znovu nastavit pomocí CSS filtrů). Ujistěte se, že máte nastaveny atributy width a height pro všechny vaše obrázky. Přidejte automatickou kompresi obrázků do svých pull requestů a prozkoumejte lazy loading méně kritických obrázků.

- **Jsou videa správně optimalizována?**

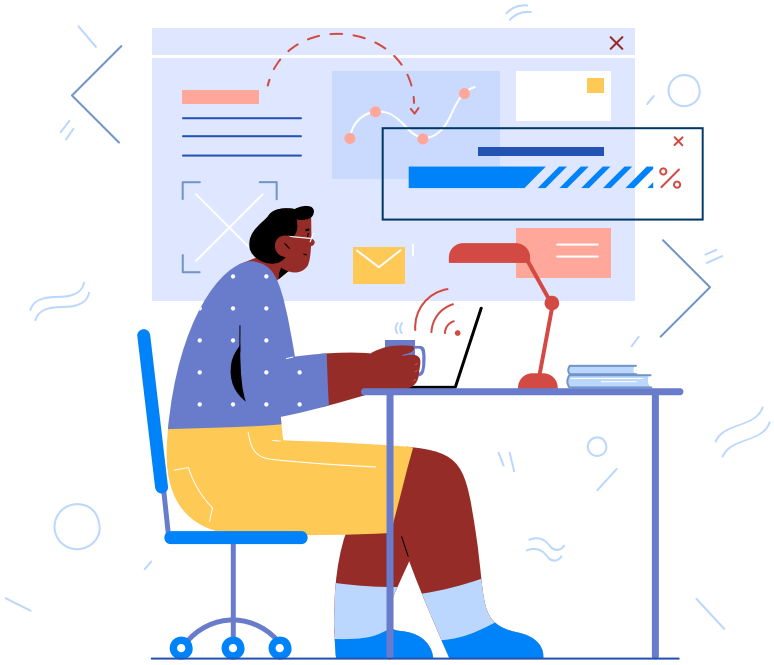
  Místo animovaných GIFů použijte buď animované WebP (a GIF jako fallback) nebo vložte smyčky s inline HTML5 videem. Ujistěte se, že vaše MP4 soubory jsou zpracovány multipass-encodingem, rozmazané pomocí efektu *„frei0r iirblur"* (je-li k dispozici), *moov atom* metadata jsou přesunuty do hlavičky soubory, a váš server akceptuje Byte serving. Kdykoliv je to možné, použijte video v AV1 formátu, ale poskytněte zároveň fallback pro starší formáty. Upravte rozměry videí a pomocí adaptivního poskytování medií poskytněte návštěvníkům obsah, který mohou přehrát na svých zařízeních bez prodlev.

- **Jsou optimalizovány fonty?**

  Nastavte u fontu správný subsetting. Upřednostněte WOFF2 a použijte WOFF jako fallback. Pomocí fallback fontu zobrazte obsah okamžitě, vlastní font načtěte asynchronně a až poté font přepněte. Ideální řešení: dvoustupňové vykreslení, nejprve s malým supersubsetem a zbytek načtěte asynchronně později. Přednačtěte (preload) 1-2 fonty od každé rodiny. Vyhněte se použití *local()* u deklarace fontu, ale zvažte použití fontů nainstalovaných v OS. Nezapomeňte použít *font-display:optional* a použijte Font Load události pro překreslení. Použijte metriky *Web Font Reflow Count* a *Time To Real Italics*.

# 5

# Optimalizace buildu



- **Nastavte správně priority.**

  Zkontrolujte všechny vaše zdroje (JavaScript, obrázky, fonty, skripty třetích stran, „drahé“ moduly na stránce) a rozdělte je do skupin. Definujte základní zážitek (plně přístupný základní obsah pro starší prohlížeče), vylepšený zážitek (obohacený, plnohodnotný pro moderní prohlížeče) a doplňky (prostředky, které nejsou bezpodmínečně vyžadovány a které lze lazy loadovat).

- **Použijte v produkci nativní JavaScript moduly.**

  Ve starších prohlížečích podporujte základní prožitek, plnohodnotný jen v moderních prohlížečích. Pro načtení JavaScriptu použijte ES2017+ `<script type="module">`. Moderní prohlížeče interpretují script jako JavaScript modul a spustí jej podle očekávání. Starší prohlížeče jej ignorují.

- **Spouštění JavaScriptu je drahé, takže jej zkroťte.**

  U SPA aplikací potřebujete nějaký čas na inicializaci aplikace, než můžete stránku vykreslit. Hledejte moduly a techniky pro urychlení počáteční doby vykreslování, např. *progressive hydration* a importem na interakci (u starších mobilů jsou časy jsou 2-5x vyšší).

- **Použijte tree-shaking, scope hoisting a code-splitting ke snížení načítání.**

  Tree-shaking je způsob, který při sestavení použije jen kód, který se skutečně používá. Code-splitting rozdělí kód do „bloků“, které jsou načítány na vyžádání. Scope hoisting detekuje, kde lze zřetězení importů sloužit a převést na vloženou funkci se zachováním funkčnosti (např. pomocí Webpacku). Použijte granular chunking a přesuňte část renderování z klienta na server. Definujte rozdělení sledováním, které bloky CSS/JS se používají a které ne. Zvažte také code-splitting na úrovni balíčku.

- **Můžete přesunout JavaScript do Web Workeru nebo WebAssembly?**

  Jak se vyvíjí kód, začnou se zobrazovat slabá místa výkonu uživatelského prostředí. Stává se proto, že operace DOMem běží vedle vašeho JS v hlavním vlákně. Zvažte přesun těchto drahých operací do procesů na pozadí, které běží v jiném vlákně pomocí webworkerů. Typický případ: přednačítání dat do PWA. Zvažte přesun výpočetně náročných úkolů do WebAssembly, který funguje nejlépe pro náročné webové aplikace, jako jsou hry.

- **Kód pro starší prohlížeče posílejte jen těmto starším prohlížečům.**

  Použijte *babel-preset-env* k transpilaci ES2017+ funkcí nepodporovaných moderními prohlížeči, které podporujete. Pak nastavte dvě verze sestavení, jednu pro moderní a druhou pro starší prohlížeče.

  Pro lodash, použijte *babel-plugin-lodash*, který načte pouze moduly, které používáte ve zdrojáku. Změňte obecné lodash require a vyberte jen ty používané a zabraňte duplikaci kódu. Pomocí iniciujte včasné (vysoce prioritní) načítání modulů.

- **Identifikujte a přepište starý (legacy) kód s pomocí „incremental decoupling“.**

  Projděte závislosti na projektu a posuďte, kolik času by bylo zapotřebí k refaktorování nebo přepsání starého kódu. Nejprve nastavte metriky, které sledují, zda poměr volání legacy kódu zůstává stejný nebo klesá. Poměr nesmí stoupat. Nastavte si v týmu takové podmínky, které nebudou tolerovat používání dalších knihoven a ujistěte se, že CI vždy vývojáře upozorní při pull requestu.

- **Identifikujte a odstraňte nepoužívané CSS/JavaScript.**

  Pokrytí CSS a JavaScriptu v prohlížeči Chrome vám umožní zjistit, který kód byl spuštěn/použit a který ne. Jakmile zajistíte nepoužívaný kód, najděte tyto moduly a přidejte líné načtení pomocí *import()*. Po úpravě test pokrytí kódu opakujte a ověřte, že se nyní při prvním načtení odesílá méně kódu. Použijte Puppeteer k automatickému shromažďování výsledků pokrytí kódem.

- **Snižte velikost závislostí v JavaScriptu.**

  Je velká šance, že na svých projektech máte celé JavaScriptové knihovny, ale reálně používáte jen zlomek kódu. Zvažte využití automatického *webpack-libs-optimizations*, který odstraňuje nepoužívané metody a polyfilly během buildovacího procesu. Přidejte do svého běžného workflow kontrolu balíčků. *Bundlephobia* pomáhá vyčíslit dopad na performance přidáváním jednotlivých NPM balíčků do projektu. *Size-limit* rozšiřuje klasickou kontrolu velikosti balíčku o délku vykonávaného času v JavaScriptu. *Skypack* můžete použít jako zdroj balíčků zaměřených na kvalitu a výkon a spravovaných jednotlivými členy komunity.

- **Používáte prediktivní prefetch pro kusy JavaScriptového kódu?**

  Využijte své zkušenosti a cit k určení, kdy budete svůj JS přednačítat. *Guess.js* zahrnuje nástroje, které používají data z Google Analytics k určování, kterou stránku uživatel pravděpodobně navštíví jako další. Zvažte *Quicklink*, *Instant.page* a *DNStradamus*. Poznámka: Možná, že překládáte prohlížeči instrukce, aby stahoval nepotřebná data a předběžně načítal nepotřebné stránky. Rozhodně doporučujeme být opatrní, pokud jde o počet předem načtených požadavků.

- **Optimalizujte pro vaše cílové enginy JavaScriptu.**

  Využijte *script streaming* pro monolitické skripty, aby je bylo možné parsovat ihned po zahájení stahování v samostatném vlákně na pozadí. Zapojte také *code caching* enginu V8 pomocí oddělením knihoven od kódu, který je používá. Zvažte také optimalizační strategie JIT pro Baseline Intrepret prohlížeče Firefox.

- **Najděte způsob jak spojit vykreslování na straně klienta a na serveru.**

  Obvykle je cílem najít optimální rovnováhu mezi client-side a server-side vykreslování. Zvažte předrenderování, pokud se vaše stránky příliš často nemění a pokud je to možné, odložte bootování frameworků. Streamujte kousky (chunks) HTML pomocí server-side vykreslování a hydratujte až při zobrazení, interakci nebo během nečinnosti, abyste získali to nejlepší z obou světů. (*Streaming Server-Side Rendering With Progressive Hydration).*

- **Zvažte mikrooptimalizace a progresivní bootování.**

  Použijte server-side render pro rychlé „první smysluplné vykreslení“ (FMP), ale také zahrňte minimální JS, abyste „čas do interaktivity“ (TTI) udrželi blízko „prvnímu smysluplnému vykreslení“. Poté buď na vyžádání nebo jak čas dovolí, spusťte nepodstatné části aplikace. Vždy rozdělte provádění funkcí do asynchronních úkolů. Pokud je to možné použijte *requestIdleCallback.*

- **Zdroje třetích stran si hostujte sami.**

  Použití veřejné CDN není automaticky rychlejší. I když dva weby odkazují na přesně stejnou URL zdroje třetí strany, kód se stáhne znova pro každou doménu. Mezipaměť je pro pro doménu izolovaná. Vlastní zdroje zůstanou v mezipaměti s větší pravděpodobností než zdroje třetích stran. Vlastní hosting je spolehlivější, bezpečnější a výkonnější.

- **Omezte vliv skriptů třetích stran.**

  Příliš často jeden skript třetí strany udělá onen „dlouhý ocas“ a rychlost pokazí. Zvažte použití service workerů ke stažení zdrojů s časovým limitem. Nastavte Content Security Policy (CSP) abyste omezili vliv skriptů třetích stran, například zakázáním stahování audia nebo videa. Vložte skripty přes iframe, díky tomu nebudou mít skripty přístup k DOM. Pro zátěžové testy skriptů prozkoumejte shrnutí na záložce Perfomance (DevTools). Načtěte skripty třetích stran až je aplikace načtena. Zaměřte se na kousky kódu, které zamezí probliknutí obsahu před načtením skriptu.

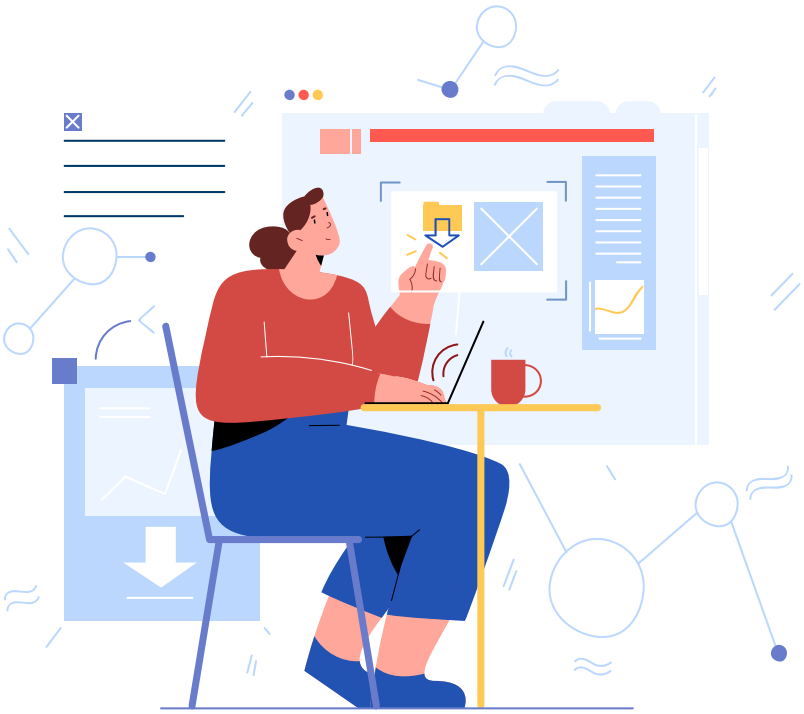
- **Nastavte správně HTTP cache hlavičky.**

  Překontrolujte, zda *expires, cache-control, max-age* a další HTTP cache hlavičky jsou správně nastaveny. Obecně by zdroje měly být kešovatelné buď na velmi krátkou dobu (pokud se pravděpodobně změní) nebo na neurčito (pokud jsou statické).

  Použijte *cache-control: immutable* abyste zabránili revalidaci. Zkontrolujte, že neposíláte zbytečné hlavičky (např. *x-powered-by*, *pragma*, *x- ua-compatible*, *expires*). Využijte nulové RTT pro opakované zobrazení prostřednictvím *stale-while-revalidate*.

# 6

# Optimalizace načítání



- **JavaScript načítejte asynchronně.**

  Využijte líné načtení pro všechny komponenty jako jsou velké JS soubory, videa, iframy, widgety a také obrázky. Využijte nativní lazy-loading (atributy loading a importance), nebo pomocí řešení využívající Intersection Observer. Druhé řešení můžete využít také pro výkonný „scrollytelling" - efekt typu parallax, nebo sledování reklam.

- **Odložte vykreslování a dekódování velkých obrázků.**

  S pomocí vlastnosti content-visibility: auto můžeme prohlížeč instruovat k přeskočení vykreslení potomků daného elementu, pokud je daný element mimo viewport. Ujistěte se, že používáte vlastnost contain-intrinsic-size s vhodně zvoleným placeholderem, abyste předešli zhoršení metriky Kumulativní posun layoutu (CLS). Použijte také `<img decoding="async">`. Prohlížeč pak dekóduje obrázek mimo hlavní vlákno a zkrátí se čas CPU potřebný pro operaci.

- **Kritické CSS načtěte co nejrychleji.**

  Z CSS vyjměte veškeré styly potřebné k vykreslení obsahu viditelného na stránce ve viewportu (tzv. kritické CSS nebo CSS nad foldem). Přidejte tyto styly jako inline do tagu na stránce. Zlaté pravidlo je, že byste neměli přesáhnout v celkové velikosti 14 kB. Zvažte podmíněné vkládání CSS na stránku. V některých případech se více vyplatí vložit kritické CSS do samostatného souboru v rootu domény. Díky cachování může být toto řešení lepší než vkládání inline stylů.

- **Experimentujte s přeskupováním CSS souborů/vlastností.**

  Při optimalizaci načtení můžete zvážit také rozdělení hlavního CSS podle jednotlivých media queries. Neumisťujte  před asynchronní soubory. Pokud skripty nezávisí na stylech, zvažte umístění blokujících skriptů nad blokující styly. Pokud ano, rozdělte tento JavaScript na dva a načtěte jej před i za CSS soubory. Ukládejte do mezipaměti vložené CSS pomocí service workerů a experimentujte s in-body CSS. Dynamické styly mohou být "drahé": zkontrolujte, zda CSS-in-JS optimalizuje vykonávání, pokud CSS nemá žádné závislosti na vzhledu nebo vlastnostech komponenty, nekomplikujte komponenty (styled-components).

- **Streamujte odpovědi.**

  Streamování poskytuje rozhraní pro čtení nebo zápis asynchronních bloků dat, jejichž podmnožina může být v daném okamžiku k dispozici v paměti. Místo toho, abyste servírovali prázdnou UI kostru a plnili obsah JavaScriptem, použijte service worker, kostru načtěte z mezipaměti a obsah ze sítě. HTML vykreslené během počátečního requestu pak může plně využít HTML parser prohlížeče.

- **Zvažte přizpůsobení vašich komponent připojení a paměti zařízení.**

  Použijte client-hint hlavičku *Save-Data* a přizpůsobte aplikaci omezením vašich uživatelů (cena, výkon). Můžete přepsat požadavky na obrázky s vysokým DPI na nízké DPI, odstranit webové fonty, parallax, vypnout automatické přehrávání videa nebo dokonce změnit jak doručujete kód. Použijte *Network Information API* pro doručení variant náročných komponent na základě připojení a *Device Memory API* k přizpůsobení zdrojů na základě paměti zařízení.

- **Udržujte stálé připojení ke zrychlení doručování.**

  Šetřete čas použitím *resource hints*; *dns-prefetch* (DNS lookup na pozadí), *preconnect* (spuštění handshake připojení (DNS, TCP, TLS)), *prefetch* (požadavek na zdroje), *preload* (přednačtení zdrojů bez jejich spuštění) a *prerender* (načtení zdrojů předem bez spuštění JS a bez vykreslení na stránce). Při použití *preload*, musí být definováno *as*, nebo se nic nenačte. Přednačtení fontů bez *crossorigin* atributu způsobí dvojí načtení. Při použití *preload* existuje řada priorit, takže zvažte vkládání *rel="preload"* prvků do DOM těsně před externí blokující skripty.

- **Použijte service workery pro ukládání do cache (mezipaměti) a jako fallback při výpadku sítě.**

  Pokud váš web běží na HTTPS, ukládejte statické prostředky do service worker cache (mezipaměti) a uložte offline fallback (nebo dokonce celé stránky) a načítejte je ze zařízení a nikoliv ze sítě. Ukládejte prostředí aplikace do service worker cache spolu s několika kritickými stránkami, např. offline stránkou nebo homepage. Ale, ujistěte se, že existuje správná CORS hlavička odpovědi, neukládejte neprůhledné odpovědi a přihlaste cross-origin obrázky do režimu CORS.

- **Použijte service worker na CDN/Edge (např. pro A/B testování).**

  U CDN implementující service workery na serveru zvažte ladění výkonu pomocí service workerů přímo „on the edge“. Např. v A/B testech, když HTML potřebuje měnit svůj obsah pro různé uživatele, použijte ke zpracování logiky service worker na CDN.

  Zrychlit weby, které používají písma Google můžete pomocí stream HTML rewriting.

- **Optimalizujte výkon vykreslování.**

  V případě potřeby, izolujte drahé komponenty pomocí CSS containment. Ujistěte se, že při posouvání stránky nebo animaci prvku nedochází k žádnému zpoždění a že trvale dosahujete rychlosti 60 snímků za sekundu. Pokud to není možné, snažte se držet rychlost snímků alespoň konstantní. Za minimální rychlost překreslení se obecně považuje 15 snímků za sekundu. Pomocí *CSS will-change* informujte prohlížeč o tom, které prvky se změní.

- **Máte optimalizovaný zážitek z vykreslování?**

  Nepodceňujte roli vnímané rychlosti. Pokud načítáte soubory, zkuste být vždy o krok před zákazníkem, aby byl zážitek plynulý, i když se na pozadí děje spousta věcí. Chcete-li zákazníka zaujmout, používejte „skeletony“ místo načítacích ozubených koleček a přidejte přechody a animace.

- **Zamezte posunům layoutu (reflow) a překreslení (repaint).**

  Posuny layoutu jsou běžně vyvolány změnou velikostí komponent jako jsou videa a obrázky, dále donačítání web fontů, vloženými reklamami nebo donačítání reálného obsahu do komponent. Nastavujte width a height u obrázků, aby mohly moderní prohlížeče rezervovat potřebné místo. Používejte zástupné SVG pro rezervaci dostatečného prostoru, ve kterém se obrázky nebo videa objeví. Použijte JavaScriptový lazy-loading, pokud není dostupná pro komponentu nativní verze. JS lazy-loading stahujte opravdu až v případě, že jej využijete. Identifikujte posuny layoutu způsobené webovými fonty a porovnejte line-height a mezery s pomocí *font-style-matcher*. Sledujte stabilitu layoutu s pomocí Layout Instability API a metriky Cumulative Layout Shift (CLS).

# 7

# Síť a HTTP/2

- **Máte zapnuté OCSP stapling?**

  OCSP stapling může zrychlit proces TLS autentizace, protože prohlížeč nemusí trávit čas hledáním a stahováním informací o certifikátu vaší domény.

- **Snížili jste dopad odvolání SSL certifikátu?**

  Extended Validation (EV) certifikáty jsou drahé a časové náročné, neboť je musí vždy ověřit člověk. Obyčejné Domain Validation (DV) certifikáty jsou dostupné zdarma (Let's Encrypt) a jejich získání a obnovu lze snadno automatizovat. EV certifikáty navíc nepodporují OCSP stapling, proto mějte vždy DV certifikát se zapnutým staplingem.

- **Používáte IPv6?**

  Studie ukazují, že webové stránky jsou až o 15 % rychlejší díky NDP (Neighbor Discovery Protocol) a optimalizaci trasy. Aktualizujte své DNS záznamy, aby byl váš web dostupný i skrze IPv6. Nezapomeňte však na stávající IPv4, neboť obě verze nejsou vzájemně kompatibilní.

- **Používá se TCP BBR?**

  BBR je relativně nový algoritmus řízení toku protokolu TCP. Reaguje spíše na skutečné přetížení než na ztrátu paketů, jako to dělá TCP. Jako takové je výrazně rychlejší, s vyšší propustností a nižší latencí. Povolte BBR a nastavte tcp_notsent_lowat na 16 kB, aby prioritizace na HTTP/2 fungovala spolehlivě na jádrech Linux 4.9 a novějších.

- **Vždy upřednostněte HTTP/2.**

  HTTP/2 je velmi dobře podporován a nabízí zvýšení rychlosti. V závislosti na tom, jak velká je vaše mobilní uživatelská základna, možná budete muset uživatelům posílat různé buildy (sestavení aplikace) a můžete profitovat z přizpůsobení různých buildů pro různá zařízení. (HTTP/2 je totiž často pomalejší v sítích, které mají znatelnou míru ztráty paketů.)

- **Správně nasaďte HTTP/2.**

  Musíte najít jemnou rovnováhu mezi spojením modulů do jednoho a paralelním načítáním mnoha malých modulů. Rozdělte celé své rozhraní na mnoho malých modulů; poté je seskupte, zkomprimujte a udělejte z nich bundly. Rozdělte je na úrovni balíčků nebo sledováním, které bloky CSS/JS se nepoužívají. Oddělte kód třetích stran od svého vlastního a oddělte závislosti, které se mění jen zřídka a často. Posílání přibližně 6–10 balíčků frontendových prvků se jeví jako slušný kompromis (a není to špatné pro starší prohlížeče). Experimentujte a měřte, abyste našli správnou rovnováhu. Snažte se poslat co nejvíce prvků přes jediné připojení HTTP/2.

- **Podporují vaše servery a CDN HTTP/2?**

  Různé servery a CDN podporují HTTP/2 odlišně. Pomocí nástrojů, které porovnávají CDN (např. CDN Comparison), můžete zkontrolovat své možnosti nebo rychle vyhledat úroveň podpory různých funkcí. Povolte BBR, nastavte tcp_notsent_lowat na 16 kB pro prioritizaci na HTTP/2.

- **Používá se komprese HPACK?**

  Pokud používáte HTTP/2, zkontrolujte, zda vaše servery implementují kompresi HPACK pro záhlaví odpovědí HTTP, abyste snížili zbytečnou režii. Servery HTTP/2 jsou relativně nové a tak nemusí plně podporovat specifikaci, příkladem je právě HPACK. H2spec je skvělý (i když technicky až moc podrobný) nástroj, který to zkontroluje.

- **Připravte se na HTTP/3.**

  Na HTTP/2 sdílí připojení více požadavků. Na HTTP/3 také požadavky sdílejí připojení, ale streamují nezávisle, takže zahozený paket již neovlivní všechny požadavky, pouze jeden stream. S technologií QUIC na HTTP/3 jsou TCP a TLS kombinovány a provedeny v jediném „round tripu“. Od druhého připojení již můžeme odesílat a přijímat data aplikační vrstvy v prvním „round tripu“ (0-RTT). Na balení frontendových prvků stále záleží, takže místo posílání monolitického JS odesílejte paralelně více souborů JS, jako na HTTP/2. Očekávejte dopad HTTP/3 na časy načítání v mobilních zařízeních.

- **Podporují vaše servery a CDN HTTP/3?**

  QUIC a HTTP/3 jsou lepší a neprůstřelnější: s rychlejšími „handshaky“, lepším šifrováním, spolehlivějšími nezávislými streamy, více šifrované a s 0-RTT, pokud měl klient připojení k serveru už dříve. Je to však docela náročné na CPU (2–3násobné využití CPU pro stejnou šířku pásma). Zkontrolujte, zda vaše servery nebo CDN podporují protokol HTTP over QUIC (také známý jako HTTP/3) a pokud můžete, povolte to.

- **Zkontrolujte, zda je zabezpečení na vašem serveru neprůstřelné.**

  Ověřte, zda jsou bezpečnostní hlavičky nastaveny správně, odstraňte známé chyby zabezpečení a zkontrolujte nastavení HTTPS. Ujistěte se, že jsou všechny externí pluginy a sledovací skripty načteny přes HTTPS, že skriptování mezi weby není možné a zda jsou správně nastaveny hlavičky HTTP Strict Transport Security i hlavičky Content Security Policy.

# 8

# Testování a monitorování

- **Monitorujte varování na smíšený obsah (mixed-content).**

  Pokud jste nedávno migrovali z HTTP na HTTPS, nezapomeňte sledovat aktivní i pasivní upozornění na smíšený obsah pomocí nástrojů, jako je Report-URI.io. Můžete také použít Mixed Content Scan ke skenování smíšeného obsahu na vašem webu s podporou HTTPS.

- **Optimalizovali jste pracovní postup auditu a ladění?**

  Investujte čas do studia technik ladění a auditu ve svém debuggeru, WebPageTest, Lighthouse a rozšiřte svůj textový editor. Můžete například spouštět WebPageTest z tabulky Google a skóre přístupnosti, výkonu a SEO začlenit do svého nastavení Travisu pomocí Lighthouse CI nebo přímo do Webpacku. Pro rychlé kontroly použijte CSS Perf Diagnostic.

- **Testovali jste v proxy prohlížečích a starších prohlížečích?**

  Testování v Chrome a Firefoxu nestačí. Podívejte se, jak váš web funguje v proxy prohlížečích a starších prohlížečích (včetně UC Browser a Opera Mini). Změřte průměrnou rychlost internetu mezi vaší uživatelskou základnou, abyste předešli velkým překvapením. Testujte s omezováním sítě a emulujte zařízení s vysokým DPI. BrowserStack je fantastický, ale testujte také na skutečných zařízeních.

- **Testovali jste rychlost svých 404 stránek?**

  Pokaždé, když klient požaduje prvek webu, který neexistuje, obdrží odpověď 404 – a tato odpověď je často obrovská. Nezapomeňte prozkoumat a optimalizovat strategii ukládání do mezipaměti také pro své 404 stránky. Ujistěte se, že prohlížeč HTML zobrazuje pouze v případě, že očekává odpověď HTML, a u všech ostatních odpovědí vraťte chybovou odpověď.

- **Testovali jste výkon svých výzev k souhlasu s GDPR a cookie lišt?**

  Za normálních okolností by výzvy k souhlasu se soubory cookie neměly mít dopad na CLS, ale někdy se to může stát, proto zvažte použití volných a open source zdrojů jako Osano nebo cookie-consent-box. Souhlas pravděpodobně změní dopad skriptů na celkový výkon, proto nastavte a prostudujte několik různých profilů testování výkonu webu pro různé typy souhlasu.

- **Testovali jste dopad na přístupnost?**

  Velké stránky a manipulace DOMem s JavaScriptem způsobí zpoždění oznámení odečítačů obrazovky. Rychlý Time to Interactive znamená, kolik času uplyne, než čtečka obrazovky může oznámit navigaci na dané stránce a uživatel čtečky obrazovky může skutečně stisknout klávesnici pro interakci.

- **Je nastaveno nepřetržité monitorování?**

  Dobré metriky rychlosti vznikají kombinací pasivních a aktivních monitorovacích nástrojů. Soukromá instance WebPagetestu a používání Lighthouse je vždy výhodné pro rychlé testy, ale také nastavení nepřetržitého monitorování pomocí nástrojů RUM, jako jsou SpeedTracker, SpeedCurve a další. Nastavte své vlastní značky časování uživatelů (user-timing marks) pro měření a monitorování metrik specifických pro váš web.

# 9

# Rychlé výhry

Checklist rychlosti webu je poměrně komplexní a dokončení všech optimalizací může chvíli trvat. Pokud byste tedy měli jen 1 hodinu na to, abyste získali výrazná vylepšení, co byste udělali? Zkrátíme to všechno na 18 nízko visících plodů. Samozřejmě platí, že než začnete a jakmile skončíte, změřte výsledky, včetně metrik Largest Contentful Paint a Time To Interactive na 3G a také kabelovém připojení.

- Změřte rychlost mezi skutečnými uživateli a stanovte vhodné cíle. Snažte se být nejméně o 20 % rychlejší než váš nejrychlejší konkurent. Udržte Largest Contentful Paint menší než 2,5 s, First Input Delay pod 100 ms, Time to Interactive pod 5 vteřin na pomalé 3G, pro opakované návštěvy pod 2 s. Optimalizujte alespoň pro First Contentful Paint a Time To Interactive.
- Optimalizujte obrázky pomocí Squoosh, mozjpeg, guetzli, pingo a SVGOMG a servírujte AVIF/WebP s obrázkovým CDN.
- Připravte kritické CSS pro své hlavní šablony a vložte je inline do každé z nich. U CSS a JS mějte rozpočet pro velikost kritických souborů max. 170 kB po gzip (0,7 MB rozbaleno).
- Ořezávejte, optimalizujte, odložte a líně načtěte skripty. Investujte do konfigurace vašeho bundleru, abyste odstranili nadbytečnost a dívejte se po méně datově objemných alternativách.
- Statické frontendové soubory vždy hostujte na vlastní doméně. Preferujte také hostování komponent třetích stran na vlastní doméně. Omezte jejich dopad. Používejte fasády a placeholdery, načítejte widgety na interakci a dejte si pozor na problik původní verze.
- Při výběru frameworku buďte nároční. U jednostránkových aplikací (SPA) identifikujte kritické stránky a renderujte je staticky, nebo je alespoň předběžně vykreslete (prerender). Používejte progresivní hydrataci na úrovni komponent a importujte moduly při interakci.
- Vykreslování jen na straně klienta není dobrou volbou pro výkon. Prerenderujte, pokud se vaše stránky příliš nezmění, a odložte bootování frameworků. Použijte streamované renderování na straně serveru (SSR).
- Poskytujte starší kód pouze starším prohlížečům pomocí návrhového vzoru module/nomodule.
- Experimentujte s přeskupením pravidel CSS a testujte CSS servírované přímo v HTML.
- Přidejte tipy pro zdroje (resource hints) ke zrychlení servírování pomocí dns-lookup, preconnect, prefetch, preload, prerender.
- Subsetujte webfonty a načítejte je asynchronně, využijte font-display v CSS pro rychlé první vykreslování.
- Zkontrolujte, zda jsou správně nastaveny záhlaví mezipaměti HTTP a bezpečnostní hlavičky.
- Povolte Brotli kompresi na serveru. (Pokud to není možné, nezapomeňte povolit kompresi Gzip.)
- Povolte řízení toku TCP pomocí BBR, pokud váš server běží na linuxovém jádře verze 4.9+.
- Pokud je to možné, povolte OCSP stapling a IPv6. Vždy servírujte DV certifikát s OCSP.
- Povolte kompresi HPACK pro HTTP/2 a migrujte na HTTP/3, pokud je k dispozici.
- Kešujte prvky jako jsou písma, styly, JavaScript a obrázky v mezipaměti Service Workeru.

- Prozkoumejte možnosti, jak se vyhnout rehydrataci, použijte pro svou jednostránkovou aplikaci (SPA) progresivní hydrataci a streamování na straně serveru.

*Velké díky za revizi původního článku na Smashing Magazine jdou k Guy Podjarny, Yoav Weiss, Addy Osmani, Artem Denysov, Denys Mishunov, Ilya Pukhalski, Jeremy Wagner, Colin Bendell, Mark Zeman, Patrick Meenan, Leonardo Losoviz, Andy Davies, Rachel Andrew, Anselm Hannemann, Barry Pollard, Patrick Hamann, Gideon Pyzer, Andy Davies, Maria Prosvernina, Tim Kadlec, Rey Bango, Matthias Ott, Peter Bowyer, Phil Walton, Mariana Peralta, Pepijn Senders, Mark Nottingham, Jean Pierre Vincent, Philipp Tellis, Ryan Townsend, Ingrid Bergman, Mohamed Hussain S. H., Jacob Groß, Tim Swalling, Bob Visser, Kev Adamson, Adir Amsalem, Aleksey Kulikov a Rodney Rehm.*

*Český překlad: Tomáš Hejč, Martin Brychta, Zuzana Šumlanská, Martin Michálek.*
*Tým PageSpeed.cz*


VZHŮRU
DOLŮ