Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

****

Vliv renderingu webových aplikací na web vitals metrics a SEO

diplomová PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Znalostní a webové technologie

Autor: Bc. Lukáš Březina

Vedoucí diplomové práce: Ing. et Ing. Stanislav Vojíř, Ph.D.

Praha, květen 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci [Název práce] [vypracoval/vypracovala] samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne [datum vč. roku (měsíc napsat slovy)]

[jméno a příjmení autora]

Poděkování

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo.

Abstrakt

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo.

Nullam justo enim, consectetuer nec, ullamcorper ac, vestibulum in, elit. Integer vulputate sem a nibh rutrum consequat. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Ut tempus purus at lorem. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam.

Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Vivamus ac leo pretium faucibus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nullam dapibus fermentum ipsum. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit.

Klíčová slova

amet, consectetuer, dolor, Lorem ipsum, sit.

JEL klasifikace

amet, consectetuer, dolor, Lorem ipsum, sit.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo.

Nullam justo enim, consectetuer nec, ullamcorper ac, vestibulum in, elit. Integer vulputate sem a nibh rutrum consequat. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Ut tempus purus at lorem. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam.

Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Vivamus ac leo pretium faucibus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nullam dapibus fermentum ipsum. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit.

Keywords

amet, consectetuer, dolor, Lorem ipsum, sit.

JEL Classification

amet, consectetuer, dolor, Lorem ipsum, sit.

Obsah

[Úvod 14](#_Toc69471784)

[1 Druhy renderingu 15](#_Toc69471785)

[1.1 Server rendering 15](#_Toc69471786)

[1.2 Client-side rendering 16](#_Toc69471787)

[1.3 Client-side rendering with prerendering 19](#_Toc69471788)

[1.4 Server-side rendering with (re)hydration 19](#_Toc69471789)

[1.4.1 Rehydration 20](#_Toc69471790)

[1.4.2 Progressive hydration 20](#_Toc69471791)

[1.4.3 Partial rehydration 20](#_Toc69471792)

[1.4.4 Trisomorphic rendering 21](#_Toc69471793)

[1.5 Static server-side rendering 21](#_Toc69471794)

[1.6 Dynamic server rendering (Turbolinks style) 22](#_Toc69471795)

[1.7 Dynamic rendering 22](#_Toc69471796)

[1.8 Shrnutí 23](#_Toc69471797)

[2 Měření výkonnosti webu 24](#_Toc69471798)

[2.1 Metody měření 24](#_Toc69471799)

[2.1.1 Syntetická měření 24](#_Toc69471800)

[2.1.2 Měření reálných uživatelů (RUM = Real user monitoring) 24](#_Toc69471801)

[2.2 Metriky měření 25](#_Toc69471802)

[2.2.1 Time To First Byte (TTFB) 25](#_Toc69471803)

[2.2.2 First Paint (FP) 26](#_Toc69471804)

[2.2.3 First Contentful Paint (FCP) 26](#_Toc69471805)

[2.2.4 DOM Content Loaded (DCL) 26](#_Toc69471806)

[2.2.5 First Meaningful Paint (FMP) 26](#_Toc69471807)

[2.2.6 Largest Contentful Paint (LCP) 26](#_Toc69471808)

[2.2.7 First Input Delay (FID) 27](#_Toc69471809)

[2.2.8 Time to interactive (TTI) 28](#_Toc69471810)

[2.2.9 Total blocking time (TBT) 28](#_Toc69471811)

[2.2.10 Cumulative Layout Shift (CLS) 28](#_Toc69471812)

[2.2.11 Load 28](#_Toc69471813)

[2.2.12 Speed Index 29](#_Toc69471814)

[2.2.13 Lighthouse Performance Score 29](#_Toc69471815)

[3 Vyhledávače, SEO a rendering 30](#_Toc69471816)

[3.1 Definice SEO 30](#_Toc69471817)

[3.2 SEO on-page optimalizace 30](#_Toc69471818)

[3.2.1 Title tag 30](#_Toc69471819)

[3.2.2 Meta tag description 31](#_Toc69471820)

[3.2.3 Obsah stránky 31](#_Toc69471821)

[3.2.4 Strukturovaná data 31](#_Toc69471822)

[3.2.5 Přizpůsobení stránky pro mobilní zařízení 32](#_Toc69471823)

[3.2.6 Sitemap 32](#_Toc69471824)

[3.3 Metody měření SEO 32](#_Toc69471825)

[3.3.1 Indexovatelnost 32](#_Toc69471826)

[3.3.2 Měření počtu přístupů 32](#_Toc69471827)

[3.3.3 Pořadí stránky ve výsledcích pro zadaný dotaz 33](#_Toc69471828)

[3.3.4 Google Lighthouse SEO audit 33](#_Toc69471829)

[3.4 Specifika jednotlivých vyhledávačů 33](#_Toc69471830)

[3.4.1 Google 33](#_Toc69471831)

[3.4.2 Bing 34](#_Toc69471832)

[3.4.3 Seznam vyhledávač 34](#_Toc69471833)

[3.4.4 Facebook, Twitter 34](#_Toc69471834)

[4 Jednostránková aplikace 35](#_Toc69471835)

[4.1 Volba aplikace 35](#_Toc69471836)

[4.2 Popis aplikace 35](#_Toc69471837)

[4.3 Specifikace aplikace 35](#_Toc69471838)

[4.3.1 Hlavička 35](#_Toc69471839)

[4.3.2 Informace o sbírkách 36](#_Toc69471840)

[4.3.3 Formulář pro vytvoření platby 36](#_Toc69471841)

[4.3.4 Výpis akcí ve farnosti 38](#_Toc69471842)

[5 Specifikace e-shop aplikace 40](#_Toc69471843)

[5.1 Volba aplikace 40](#_Toc69471844)

[5.2 Popis aplikace 40](#_Toc69471845)

[5.3 Specifikace komponent aplikace 40](#_Toc69471846)

[5.3.1 Záhlaví 40](#_Toc69471847)

[5.3.2 Hlavička 40](#_Toc69471848)

[5.3.3 Banner 41](#_Toc69471849)

[5.3.4 Karta produktu 41](#_Toc69471850)

[5.3.5 Stránkovač 42](#_Toc69471851)

[5.3.6 Karta vlastnost produktu 42](#_Toc69471852)

[5.3.7 Box přidat do košíku 43](#_Toc69471853)

[5.3.8 Box objednávka 43](#_Toc69471854)

[5.3.9 List produktů v košíku 44](#_Toc69471855)

[5.3.10 Patička 44](#_Toc69471856)

[5.4 Specifikace jednotlivých stránek aplikace 45](#_Toc69471857)

[5.4.1 Domovská stránka 45](#_Toc69471858)

[5.4.2 Detail kategorie 45](#_Toc69471859)

[5.4.3 Detail produktu 45](#_Toc69471860)

[5.4.4 Košík 46](#_Toc69471861)

[5.4.5 Statická stránka 46](#_Toc69471862)

[6 Metodologie testování a vyhodnocování 47](#_Toc69471863)

[6.1 Testovací aplikace 47](#_Toc69471864)

[6.2 Metody renderingu 47](#_Toc69471865)

[6.3 Výkonnostní metriky 47](#_Toc69471866)

[6.3.1 Syntetické měření pomocí Lighthouse 48](#_Toc69471867)

[6.3.2 Syntetické měření pomocí Pagespeed.cz 48](#_Toc69471868)

[6.3.3 Měření reálných uživatelů 48](#_Toc69471869)

[6.4 Měření SEO 48](#_Toc69471870)

[6.4.1 Pozice ve vyhledávačích pro dosud neexistující produkty 49](#_Toc69471871)

[6.4.1.1 Výsledek vyhledávání 49](#_Toc69471872)

[6.4.1.2 Google product snippet 49](#_Toc69471873)

[6.4.1.3 Facebook snippet 49](#_Toc69471874)

[7 Implementace 50](#_Toc69471875)

[7.1 Technologie pro implementaci 50](#_Toc69471876)

[7.2 Jednostránková aplikace 50](#_Toc69471877)

[7.2.1 A/B proxy 51](#_Toc69471878)

[7.2.2 Sbírání metrik reálných uživatelů 51](#_Toc69471879)

[7.2.3 Farní události 51](#_Toc69471880)

[7.2.4 FE aplikace 51](#_Toc69471881)

[7.3 Sbírání metrik 51](#_Toc69471882)

[7.4 Jednostránková aplikace 51](#_Toc69471883)

[7.5 E-shop 52](#_Toc69471884)

[7.5.1 Backend 52](#_Toc69471885)

[7.5.2 FE aplikace 52](#_Toc69471886)

[8 Vyhodnocení 53](#_Toc69471887)

[9 TODOs 54](#_Toc69471888)

[10 Otázky ke konzultaci 55](#_Toc69471889)

[Závěr 56](#_Toc69471890)

[Použitá literatura 57](#_Toc69471891)

[Přílohy I](#_Toc69471892)

[Příloha A: Wireframe aplikace farní sbírky I](#_Toc69471893)

[Příloha B: Wireframe e-shop zahradnictví I](#_Toc69471894)

Seznam obrázků

[Obr. 1.1 Četnosti různých velikostí bot v populaci žen, mužů a dětí (vlastní zpracování) 13](#_Toc529974559)

Poznámka: Seznam obrázků je vhodný použít, pokud počet obrázků v textu práce je větší než 20. Seznam grafů je vhodný použít pouze v případě, že autor práce rozlišuje mezi obrázkem a grafem. Seznam grafů je utvářen, pokud je počet grafů větší než 20. V této šabloně závěrečné práce se grafy a obrázky nerozlišují.

Seznam tabulek

[Tab. 1.1 Záznam výsledků závodu robotů (vlastní zpracování) 14](#_Toc529974579)

Poznámka: Seznam tabulek je vhodný použít, pokud počet tabulek v textu práce je větší než 20.

Seznam výpisů programového kódu

[Výpis 1.1 Ukázka formátování krátkého výpisu programového kódu (vlastní zpracování) 14](#_Toc529974597)

Poznámka: Seznam výpisů programového kódu je vhodný použít, pokud počet vložených objektů tohoto typu je větší než 20.

Seznam zkratek

SSR server-side rendering

SPA single page application

SEO

API

DOM

HTML

CDN

URL

LPS

FCP

SI

LCP

TTI

TBT

CLS

Poznámka: Seznam zkratek je vhodný použít, pokud počet zkratek v textu práce je větší než 20 a nejedná se o zkratky běžné.

Úvod

V diplomové je rozebrán vliv různých druhů renderingu webových aplikací na jejich výkonnost. Kromě toho je snaha popsat, jak si s jednotlivými druhy renderingu umí poradit vyhledávací a jiní roboti a jaký vliv mají tyto druhy renderingu na SEO.

Hypotézy, které chceme ověřit:

* Jednotlivé renderingy vzhledem k hodnotám TTFB, CLS, atd… - viz google obrázek
* Google umí indexovat CSR, ostatní vyhledávače, soc sítě ne, what’s the state pro jednotlivé renderingy?
* Google dává ve vyhledávání stále prioritu SSR/server render aplikacím oproti CSR

Napsat aspoň cíl teď…

Začít obecnou motivací proč tuto diplomku vlastně řeším, pár vět o tom že existuje více druhů renderingů – nejen to jak vykresluje prohlížeč, ale i to jak se stránka vytváří. Pak nějaký náš nápad, který konkretizujeme do cílle – předpoklad – výkonnostní rozdíly + cíl práce – je prozkoumat…

U cíle metodika řešení – návod jak dosáhneme toho cíle, kouknout na wiki k článku vědecké metody a najít správné termíny které se týkají mě. Najít článek o výkonnostním měření (diplomová práce o měření z budějc?).

Předpoklady a omezení – předpoklad = nechceme psát pro laiky, ale pro člověka který má zkušeností s programováním (webů?). Omezení – věci co máme dané (technologie kterou chceme měřit, máme to už na nějakém serveru…)

Odstavec, kde shrneme strukturu práce.

Cekem 1 – 2 strany.

# Druhy renderingu

Ze začátku je potřeba vyjasnit, o jakém renderingu webových aplikací tato práce pojednává. Renderingem webové aplikace v kontexu této práce je myšleno především převedení surových dat do HTML a CSS. Explicitně se tedy nejedná o rendering, který řeší vykreslovací jádra[[1]](#footnote-2) jednotlivých prohlížečů – tj. převod HTML a CSS na jednotlivé pixely. V práci je tedy řešeno, v jaký čas a jakými výpočetními prostředky převedeme surová data do formy, kterou již umí vykreslovací jádra vykreslit (HTML, CSS).

V následujících podkapitolách jsou popsány jednotlivé přístupy k renderingu. Je vycházeno především ze článku (Miller a Osmani, 2019), kde jsou jednotlivé druhy velice dobře popsány a shrnuty. U každého druhu je rozebrána typická architektura aplikace rendering využívající a princip fungování. Je naznačeno, jak se u jednotlivých druhů renderingu pracuje s daty a jak vypadá odpověď na HTTP požadavek pro jednoduchou aplikaci úkolníčku. Také je u každého druhu dán za příklad framework, knihovna, či aplikace, tento druh renderingu používající.

## Server rendering

Jedná se o nejstarší, nejklasičtější a také nejrozšířenější podobu renderingu webových aplikací. Prohlížeč na HTTP požadavek obdrží v odpovědi HTML dokument již obsahující data, která chceme vykreslit, také ve formátu HTML (viz Výpis 1.1). V rámci tohoto dokumentu jsou také v hlavičkách obvykle připojeny kaskádové styly upravující vzhled stránky. Javascript se při tomto druhu renderingu obvykle používá hlavně pro „zinteraktivnění“ aplikace – např. validování uživatelských vstupů ve formuláři, pokročilé animace, *drag&drop* operace atd. Serverově renderované aplikace jsou také specifické tím, že jejich stav (např. položky v košíku) je obvykle uchováván kompletně na serveru. Změna tohoto stavu probíhá obvykle posláním příslušného HTTP požadavku, který opět vrátí kompletní HTML dokument zobrazující již aktualizovaný stav.

Z hlediska softwarové architektury jsou serverově renderované aplikace obvykle monolitického typu.[[2]](#footnote-3) Jedná se tedy o jedinou aplikaci řešící vše od zpracování HTTP požadavku, přístupu k databázi, až po vyrenderování samotného HTML.

Na tvorbu serverově renderované aplikace existuje nespočet frameworků a jazyků, ve kterých je možno je psát. Nejčastěji používaným je jazyk PHP (W3Techs.com, 2020). Zástupcem serverově renderované aplikace může být například InSIS, který je napsaný v Perlu[[3]](#footnote-4).

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<link rel="stylesheet" href="style.css">

<title>Úkolníček</title>

</head>

<body>

<h1>Výpis úkolů</h1>

<ul>

<li><a href="/ukol/1">Vyvenčit psa</a></li>

<li><a href="/ukol/2">Vynést koše</a></li>

</ul>

<form method="POST" action="/pridat\_ukol">

<label for="ukol">Úkol</label>

<input id="ukol" type="text">

<button>Přidat</button>

</form>

<script src="script.js" type="text/javascript"></script>

</body>

</html>

Výpis . Nástin HTTP odpovědi pro server-rendered stránku úkolníčku

## Client-side rendering

Na rozdíl od serverově renderovaných aplikací obvykle veškeré HTTP požadavky na danou aplikaci vrací stejný obsah. Jedná se o HTML dokument obsahující odkazy na kaskádové styly a javascripty, který obvykle v těle obsahuje pouze text, popř. animaci, vizualizující načítání aplikace (viz Výpis 1.2). Jak a odkud se vezmou potřebná data pro daný požadavek, či jak se tyto data vykreslí, řeší již javascriptová aplikace odkazovaná v HTML dokumentu, u klienta v prohlížeči. Obvykle tedy javascript po načtení odešle několik HTTP požadavků na API, ze kterých získá potřebná data. Tato data poté vykreslí do DOMu (viz Výpis 1.3). Dost často tyto aplikace také drží velkou část stavu aplikace. Změna stavu obvykle probíhá tak, že se na pozadí pošle HTTP požadavek o změnu stavu a v případě úspěchu se překreslí pouze části aplikace ovlivněné změnou tohoto stavu (např. změna čísla s počtem položek v košíku v hlavičce po odebrání z košíku). Hlavní výhodou aplikací renderovaných na klientské straně je tedy absence kompletního přenačítávání celé stránky s každou změnou stavu. Zásadní nevýhodou je pomalé první načtení stránky.

Z architektonického hlediska se obvykle jedná o client-server architekturu[[4]](#footnote-5). Webová aplikace má obvykle dvě oddělené části – tzv. backend (server) a frontend (client), které mezi sebou komunikují pomocí API. Backendová část obvykle řeší ukládání a zpřístupnění dat, náročnější business logiku, komunikaci s jinými třetími stranami, či výpočetně náročné operace. Frontendová část tvoří uživatelské rozhraní aplikace a má na starosti rendering. Uživatel celou dobu interaguje pouze s touto částí, která jeho akce převádí na jednotlivá API volání. Velkou výhodou této architektury webových aplikací je to, že v tomto případě mohou být podle potřeby použito a libovolně kombinováno více backendových částí (např. aplikace sdružující více informačních systémů), více frontendových částí (web, mobilní aplikace, televizní aplikace) či různé technologie/architektury pro API (REST[[5]](#footnote-6), GraphQL[[6]](#footnote-7), Websocket[[7]](#footnote-8)). V případě frontend části se stala standardem *component-oriented* architektura (dshaps, 2016), která se také velmi liší od architektury používané v monolitických serverově renderovaných aplikací. Základem jsou znovu použitelné malé celky skládající se z šablony, stylů a logiky – komponenty.

Nejznámější frameworky/knihovny využívající client-side rendering jsou React[[8]](#footnote-9), Angular[[9]](#footnote-10) a Vue[[10]](#footnote-11). Typickou client-side rendered aplikací je např. Gmail[[11]](#footnote-12).

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<link rel="stylesheet" href="style.css">

<script src="app.js" type="text/javascript"></script>

<title>Úkolníček</title>

</head>

<body>

<app-root>

<img src=“nacitani.gif“ alt=“Načítám“>

</body>

</html>

Výpis . Nástin HTTP odpovědi pro client-rendered stránku úkolníčku

…

<body>

<app-root>

<app-vypis-ukolu>

<h1>Výpis úkolů</h1>

<ul>

<!-- data loaded from API, redirect on link click

prevented and handled by JS -->

<li><a href="/ukol/1" routerLink="ukol/1">Vyvenčit psa</a></li>

<li><a href="/ukol/2" routerLink="ukol/1">Vynést koše</a></li>

</ul>

</app-vypis-ukolu>

<app-novy-ukol>

<!-- submit and sending data handled by JS -->

<form>

<label for="ukol">Úkol</label>

<input id="ukol" type="text">

<button>Přidat</button>

</form>

</app-novy-ukol>

</app-root>

</body>

…

Výpis . client-side rendered stránka úkolníčku

## Client-side rendering with prerendering

Jedná se o jeden z přístupů, jak zlepšit rychlost prvního načtení stránky. Aplikace je psána stejným způsobem, jako client-side rendered aplikace. Building proces je ale upraven tak, že negeneruje prázdný HTML dokument, ale již předgenerovanou HTML strukturu (viz Výpis 1.4), do které se po načtení javascriptu pouze načtou dynamická data a stránka vypadá stejně, jako by byla vyrenderovaná na klientské straně (viz Výpis 1.3).

…

<body>

<app-root>

<app-vypis-ukolu>

<h1>Výpis úkolů</h1>

<ul>

</ul>

</app-vypis-ukolu>

<app-novy-ukol>

<!— submit nic neudělá dokud nenastartuje js aplikace -->

<form>

<label for="ukol">Úkol</label>

<input id="ukol" type="text">

<button>Přidat</button>

</form>

</app-novy-ukol>

</app-root>

</body>

…

Výpis . Nástin HTTP odpovědi pro CSR with prerendering stránku úkolníčku

## Server-side rendering with (re)hydration

Javascript může být kromě prohlížeče spouštěn také na serveru díky NodeJS[[12]](#footnote-13). Tohoto faktu využívá právě server-side rendering. Server-side renderingem rozumíme spuštění kódu client-side rendered aplikace pro daný požadavek na serveru. Výstup tohoto spuštění je tedy HTML dokument včetně stylů, který odpovídá tomu, co vyrenderuje klientská aplikace po načtení. Problém nastává, když výstup tohoto HTML dokumentu zobrazí prohlížeč. Stránka sice vypadá až na výjimky identicky jako client-side vyrenderovaná, ale není interaktivní – javascriptový kód registrující event listeners byl již spuštěn na serveru. „Zinteraktivnění“ takto vygenerovaného dokumentu se nazývá *hydration*. (Miller a Osmani, 2019)

Výpis . Server-side rendering with (re)hydration

V následujících odstavcích bude popsáno několik způsobů, jak lze tuto hydration provést.

### Rehydration

První možností je tzv. *rehydration*. Součástí dokumentu vráceným serverem je kromě onoho statického HTML také skript klientské aplikace a serializovaná data, která byla použita pro její rendering. Prohlížeč tedy HTML dokument vykreslí a přibalený skript se spustí jako při klasicky klientsky renderované aplikaci. Protože už má dostupná všechna data, která by obvykle získával voláním např. XmlHttpRcequestu v serializované podobě, může klient renderovat aplikaci velmi rychle. Po dokončení se buď serverem renderovaný DOM kompletně zahodí a nahradí DOMem vyrenderovaný klientem. Druhá možnost je, že se DOM nezahazuje, ale na již existující DOM se napojí všechny potřebné event listenery atd.

Když je někde popsán server-side rendering (SSR) ve SPA aplikacích, je tím většinou myšlen právě tento způsob renderingu s rehydration.

### Progressive hydration

Problémem rehydratace je velká výpočetní zátěž při načítání stránky. Toto se zejména na výpočetně slabších zařízeních, jako jsou chytré telefony, projevuje nejvíce. Hlavním problémem je, že stránka na takovémto telefonu vypadá funkčně, ale není interaktivní třeba i po několik sekund.

Jedním z přístupů, jak tento problém vyřešit, je rozdělit tento velký úkol hydratace na mnoho menších. Způsob, jakým to lze udělat, je staticky zanalyzovat celou aplikaci, a pro každou komponentu zjistit, jaké má závislosti a jaké události jdoucí z komponenty ovlivní jaké jiné komponenty v aplikaci. Díky tomu jsme aplikaci schopni rozdělit na mnoho malých částí, které lze dynamicky načítat a spouštět až v případě potřeby. Toto je pro frameworky či knihovny zajišťující rendering implementačně velice náročný úkol. S *progressive hydration* se již experimetuje v ReactJS (Markbåge, 2019) a diskutuje již několik let v Angularu (Cross, 2016). Velmi pěkně shrnuje problémy a nástrahy implementace progressive hydration v Angularu ve svém komentáři k návrhu implementace Minko Gechev (Gechev, 2020).

### Partial rehydration

*Partial rehydration* je *progressive hydration* posunutá o úroveň dál. Již během buildu jsou identifikovány komponenty, které jsou statické (pro jejich funkčnost javascript není potřeba). Tyto komponenty můžeme vyrenderovat na serveru a odstranit veškerý javascriptový kód, který by tyto komponenty jinak obsluhoval. Zbylé nestatické komponenty jsou po načtení hydratovány na klientské straně obdobně jako při *rehydration*, či *progressive hydration*.

Správně naimplementovat tento přístup je velice obtížné – nastávají zde problémy při přechodu mezi stránkami řízenými klientskou částí kódu – zde se již nevolá SSR, které by tyto statické komponenty vyrenderoval. Pěkný článek popisující pokus o implementaci tohoto přístupu v Reactu, resp. frameworku nad ním postaveným Next.js je k nalezení v (Bombach, 2019).

### Trisomorphic rendering

V tomto případě rendering může probíhat na třech místech: serveru, klientovi a service workeru v prohlížeči. První načtení aplikace vyrenderuje server. Všechny další navigace v aplikaci již jdou přes service worker, který si přednačte a uloží další stránky/komponenty v aplikaci. To umožňuje přechody mezi stránkami v aplikaci stejně rychle jako by aplikace byla renderována pouze na klientovi. Tento přístup může fungovat pouze když může být použit stejný kód pro vykreslení a navigaci napříč serverem, service workerem a klientskou aplikací. Více informací o tomto druhu renderingu lze nalézt v diagramu Obrázek 1.

Jedná se o velice experimentální druh renderingu, konkrétní příklad knihovny či fungující aplikace využívající tento druh renderingu se nepodařilo nalézt.

Diagram of Trisomorphic rendering, showing a browser and service worker
communicating with the server

Obrázek 1 Trisomorphic rendering diagram (Miller a Osmani, 2019)

## Static server-side rendering

Aplikace je psaná stejně jako client-side renderovaná aplikace (používáme komponenty, fetch API, …). Při buildu aplikace vygenerujeme kompletní HTML soubory pro každou stránku aplikace. Na rozdíl od prerenderingu je z těchto souborů kompletně odstraněn javascript, takže aplikace je kompletně statická.

Tento druh renderingu je použitelný pouze pro velmi specifické případy. Pro každou změnu obsahu stránky musí být aplikace znovu zbuilděna. Oproti klasickému HTML je výhoda v tom, že kód může být strukturován do znovupoužitelných komponent, a tedy mohou jednotlivé stránky sdílet společné části. Na rozdíl od server renderingu např. v PHP, není potřeba vlastního hostingu s běžícím PHP, který je hůře škálovatelný. Výstupem jsou totiž statické HTML soubory, které můžeme umístit třeba na CDN, které aplikaci rozdistribuuje velice levně po celém světě.

Příkladem frameworku využívající tohoto druhu renderingu je Docosaurus (Facebook Inc., 2020). Jejich stránka je zároveň napsána v tomto frameworku, takže se jedná i o příklad aplikace s tímto druhem renderingu.

Odpověď serveru na HTTP požadavek je v tomto případě nerozlišitelná od Výpis 1.1.

## Dynamic server rendering (Turbolinks style)

Jedná se o vylepšení klasického server renderingu. Dynamic server rendering eliminuje nutnost kompletního znovunačtení stránky s každou navigací / akcí v aplikaci. Toho dociluje tím, že zachytí veškeré navigační požadavky, provede je pomocí XHR API z javascriptu a výslednou odpovědí nahradí aktuální DOM. Kromě toho se již navštívené stránky cachují, takže přechod na již navštívenou stránku je proveden okamžitě. V případě, že pro celou aplikaci existuje pouze jediné CSS a jediný JS, šetří se s každou navigací na stránce čas s jejich parsováním a spouštěním. Nevýhodou je, že callback document.ready, či window.load je zavolán pouze při prvním načtení. S každou další navigací je již zavolán knihovnou zajišťující tento rendering jiný callback. V knihovně Turbolinks se například jedná o turbolinks:load (Basecamp, LLC, 2020). Tato skutečnost může rozbít již existující javascriptové knihovny spoléhající na skutečnost, že s každou navigací bude onen document.ready callback zavolán. Jak již z povahy tohoto řešení vyplývá, odpověď na HTTP požadavek je totožná s Výpis 1.1.

Příkladem knihovny implementující tento druh renderingu je již výše zmíněný Turbolinks (Basecamp, LLC, 2020). Příkladem aplikace tento rendering používající může být například (Wallmine, 2020).

## Dynamic rendering

Dynamic rendering je Googlem doporučený workaround (Google LLC, 2020a), jak umožnit crawlerům vyhledávačů a jiným robotům číst client rendered aplikace. Principiálně funguje tak, že uživatelům s prohlížeči posíláme klasickou client rendered aplikaci. Requesty crawlerů jsou přesměrovány na renderer, který již vrací plně vyrenderované HTML. Tento renderer obvykle funguje tak, že stránku načte v headless Chrome, které aplikaci vyrenderuje (GoogleChrome, 2020), (prerender, 2020). Tento výsledek pak vrátí.

Příkladem rendererů, které lze použít jsou Rendetron (GoogleChrome, 2020) či Prerender.io (prerender, 2020).

A diagram that shows how dynamic rendering works. The diagram shows the server serving
              initial HTML and JavaScript content directly to the browser. In contrast, the diagram
              shows the server serving initial HTML and JavaScript to a renderer, which converts the
              initial HTML and JavaScript to static HTML. Once the content is converted, the
              renderer serves static HTML to the crawler.

Obrázek 2 Dynamic rendering (Google LLC, 2020a)

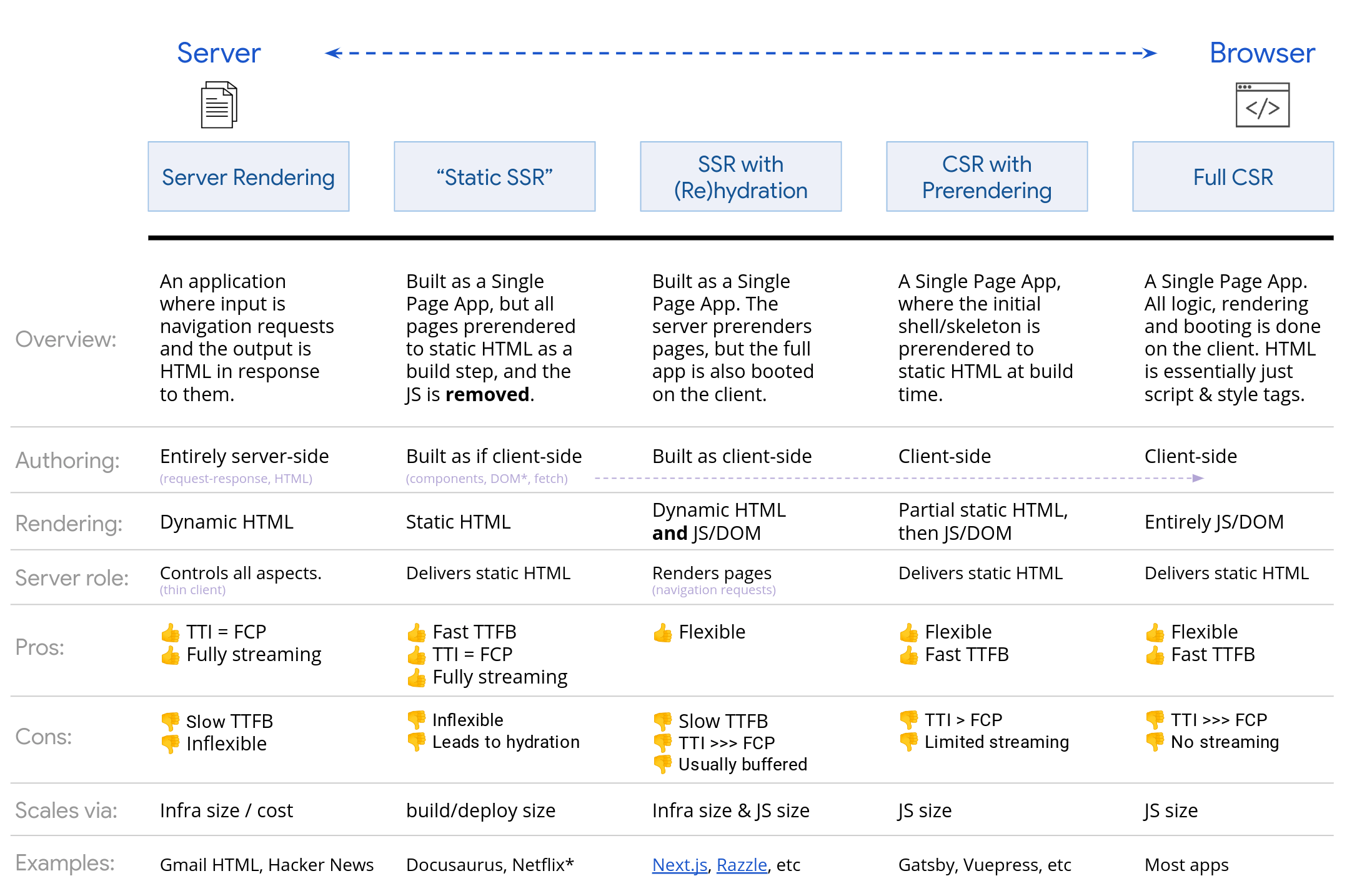
Velkou výhodou tohoto druhu renderingu oproti SSR je jednoduchost implementace.

Zajímavým problémem tohoto druhu renderingu je, že již z principu dodává jinou verzi stránek vyhledávacím robotům než uživatelům. Této technice se říká *maskování* a je považováno za porušení podmínek Google (Google LLC, 2020c). V tomto případě ale Google vysloveně uvádí, že „Pokud dynamic rendering produkuje podobný obsah, Googlebot nebude pohlížet na dynamic rendering jako na maskování.“ (Google LLC, 2020a). Jak je ale správně poznamenáno ve článku (Lavall, 2020), je otázka co už Google bot nemusí považovat za „podobný obsah“ a ve výsledku takto renderovaný obsah penalizovat.

Anthony Lavall ve výše uvedeném článku také zmiňuje další nevýhodu tohoto přístupu a tou je aktuálnost dat. Služby jako výše zmíněný prerender.io cachují již vygenerované odpovědi, a to minimálně po dobu několika dnů. Absence cachování řešení ale také není. Měla by za následek velkou výpočetní zátěž na straně serveru a pomalé odpovědi crawlerům.

## Shrnutí

Článek, který byl primárním zdrojem pro tuto kapitolu (Miller a Osmani, 2019) uvádí toto shrnutí jednotlivých přístupů k renderingu a uvádí je do kontextu jednotlivých výkonnostních metrik, které budou popsány v následující kapitole:



Obrázek 3 Shrnutí jednotlivých metod renderingu

Autoři článku také uvádí, že je při výběru renderingu nutno rozumět tomu, jaká úzká hrdla jednotlivé druhy renderingu mají. Výběr konkrétního druhu renderingu by měl záviset na konkrétní aplikaci a jaká očekávání po stránce výkonu a použitelnosti od ní zadavatel požaduje. V neposlední řadě výběr renderingu také ovlivňuje nejen použité technologie, ale celkovou architekturu aplikace a prostředí/počítače, na kterých poběží.

# Měření výkonnosti webu

V této kapitole jsou shrnuty a teoreticky popsány metody měření a metriky používány k měření výkonnosti webů. Výběr konkrétních metrik a metod, pomocí kterých budou měřeny aplikace vytvořené v rámci této práce, budou popsány v kapitole Metodologie testování a vyhodnocování.

Změřit výkonnost webové aplikace není zdaleka tak jednoduchý úkol, jak se na první pohled může zdát. Pro demonstraci lze ukázat například naivní přístup k měření výkonnosti. Nechť je metrikou čas, za jak dlouho budeme mít kompletní odpověď na HTTP požadavek. Taková metrika může být vypovídající pro aplikace renderované na serveru. V momentě, kdy bude aplikace napsaná jako client rendered SPA, odpověď přijde vždy téměř okamžitě. V tomto případě se hlavní část práce začne dít až při spuštění javascriptu, který začne přes API stahovat potřebná data, což může trvat dalších několik sekund. Z uživatelského pohledu tedy může serverově renderovaná aplikace působit mnohem výkonněji než SPA aplikace, i přesto že čas odpovědi na HTTP požadavek může být násobně delší.

Jak lze vidět z výše popsaného příkladu, vymyslet metriku, která by věrohodně měřila výkon webové aplikace z uživatelského pohledu, není jednoduchý úkol. Za poslední léta vývoje a měření výkonnosti webových aplikací bylo vymyšleno několik různých metrik, které jsou v této kapitole popsány.

## Metody měření

Měřit výkonnost webu lze dvěma způsoby – synteticky a na reálných zařízeních návštěvníků. Každý ze způsobů má své výhody a nevýhody, které jsou shrnuty v této podkapitole.

### Syntetická měření

Syntetické měření je nejčastěji používaná technika. Na web se obvykle pošle robot, který simuluje reálného uživatele, jeho konkrétní prohlížeč, rozlišení, rychlost internetu atd. Mezi nástroje/stránky pracující na principu syntetických měření patří Lighthouse, PageSpeed Insights, WebpageTest.org, Pagespeed.cz a další. (Michálek, 2019e) Syntetická měření mají výhodu v reprodukovatelnosti měření a v okamžité odezvě na nové změny ve stránce. Nevýhodou může být nepřesná emulace pomalejších zařízení a nutnost specifikace zařízení, na kterých bude aplikace testována.

### Měření reálných uživatelů (RUM = Real user monitoring)

Měření reálných uživatelů je daleko lepší metodika. Do stránky se obvykle vloží skript, který měří reálné uživatele aplikace. Některé analytické nástroje to už dnes umí, jen jsou nastavené spíše na velké weby a firmy. Obvykle jsou také velmi nákladné (Michálek, 2019e).

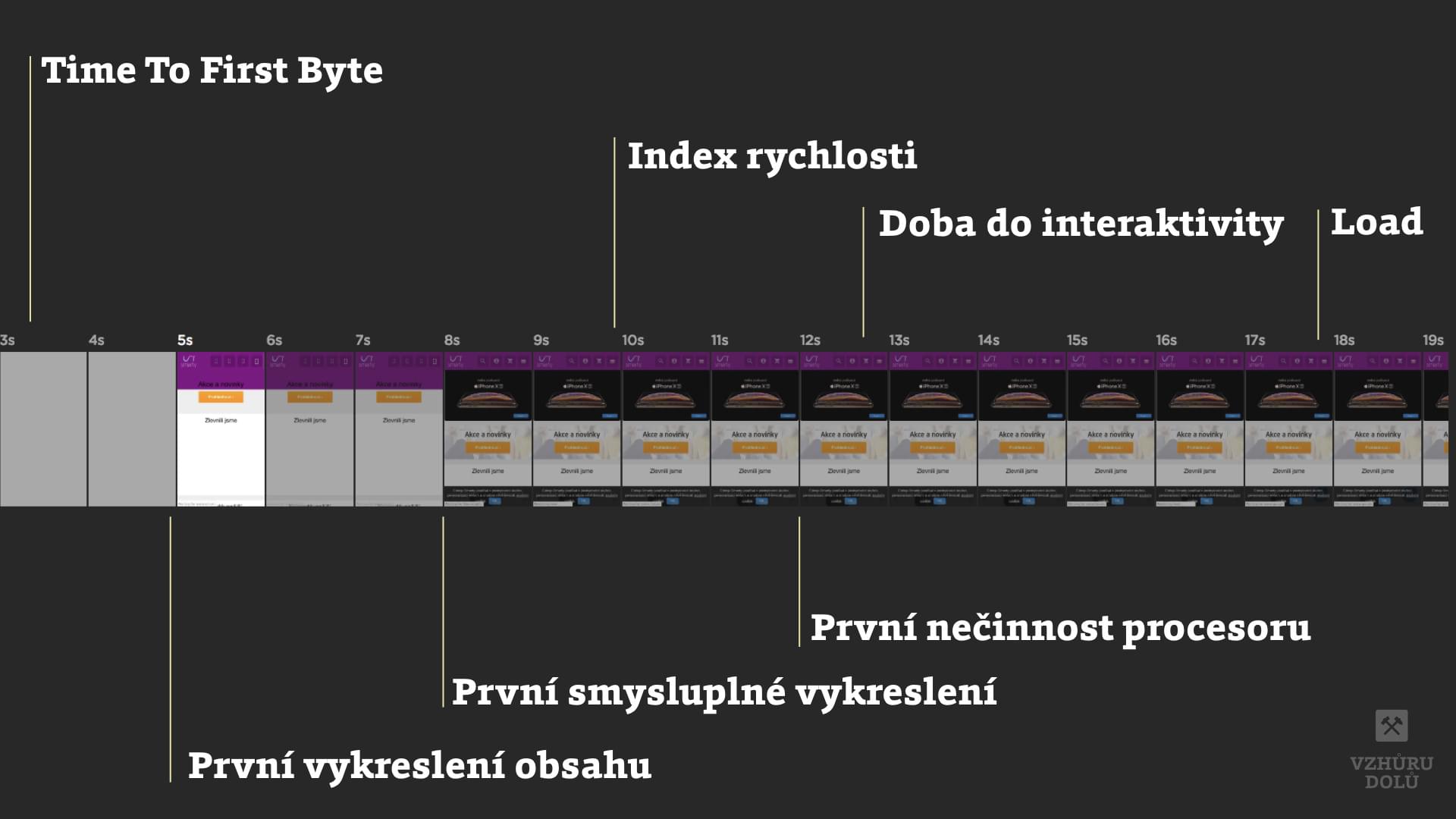
U většiny webů RUM metriky zobrazí i PageSpeed Insights – z veřejné databáze Chrome UX Report (CrUX). Sbírat tyto data od uživatelů umí také SpeedCurve, jedná se ale o nákladnou službu. Další možností je použít knihovnu javascriptovou knihovnu web-vitals (Google LLC, 2021b), která umožní zaznamenat metriky v prohlížečích. Jejich sbírání, ukládání a vyhodnocování je již ale potřeba vyřešit svépomocí.

## Metriky měření

Seznam metrik zahrnutých do této teoretické analýzy je čerpán hlavně z (Michálek, 2019e).

Nejdůležitějšími metrikami, podle kterých budou primárně měřeny i aplikace v této práci, budou metriky tzv. Core web vitals, kterými jsou LCP, FID a CLS (Google LLC, 2020e) a popř. také Lighthouse performance score, které by mělo být syntetickou obdobou těchto core web vitals. Tato kombinace metrik by měla co nejvěrohodněji reprezentovat výkonnost webu z uživatelského hlediska. (Google LLC, 2020d) Kromě toho také bude Google od příštího roku ve svém vyhledávacím algoritmu tyto metriky zohledňovat a zvýhodňovat weby, které dosahují v těchto metrikách nižších hodnot. (Subramanian‎, 2020)

Přehled jednotlivých momentů během vykreslování stránky lze vidět na Obrázek 3.



Obrázek 4 *Postupný vznik událostí pro vykreslování stránky* (Michálek, 2020a)

### Time To First Byte (TTFB)

„Time to First Byte (TTFB) refers to the time between the browser requesting a page and when it receives the first byte of information from the server. This time includes DNS lookup and establishing the connection using a TCP handshake and SSL handshake if the request is made over HTTPS.“ (Mozilla Contributors, 2019)

Metrika TTFB tedy měří hlavně rychlost odpovědi serveru na požadavek klienta. Při čistě serverově renderovaných webových aplikacích lze z této metriky vyčíst slabá místa aplikace.

### First Paint (FP)

„First Paint, part of the Paint Timing API, is the time between navigation and when the browser renders the first pixels to the screen, rendering anything that is visually different from what was on the screen prior to navigation. It answers the question »Is it happening?«“(Mozilla Contributors, 2021b)

Jak vyplývá z uvedené definice, tato metrika určuje, kdy se něco začalo vykreslovat. Tento údaj sám o sobě není příliš vypovídající. Uživatele většinou zajímá, kdy se vykreslil obsah stránky. Pro měření vykreslení se dnes tedy dává spíše přednost metrikám FCP a FMP. (Michálek, 2019h)

### First Contentful Paint (FCP)

„First Contentful Paint (FCP) is when the browser renders the first bit of content from the DOM, providing the first feedback to the user that the page is actually loading. … The First Contentful Paint time stamp is when the browser first rendered any text, image (including background images), non-white canvas or SVG. This excludes any content of iframes, but includes text with pending webfonts. This is the first time users could start consuming page content.“ (Mozilla Contributors, 2020a)

Touto metrikou lze tedy změřit, v jakém bodě se uživateli vykreslí nějaký obsah (klidně i text „Načítám“). Jedná se o jednu z důležitých metrik, která se používá pro vyhodnocování výkonnosti webu, je také součástí web vitals metrik (Google LLC, 2019a).

### DOM Content Loaded (DCL)

„The DOMContentLoaded event fires when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading.“ (Mozilla Contributors, 2020c).

Jak již definice napovídá, jedná se o jednu z událostí životního cyklu stránky, na který můžeme přidat *eventListener* v javascriptu. Jedná se ale spíše o technickou metriku, která sice koreluje s nižší *bounce rate*, je to ovšem tím, že DCL koreluje s FCP a FMP. (Michálek, 2019f)

### First Meaningful Paint (FMP)

„First Meaningful Paint (FMP) is the paint after which the biggest above-the-fold layout change has happened and web fonts have loaded. It is when the answer to "Is it useful?" becomes "yes", upon first meaningful paint completion.“ (Mozilla Contributors, 2020b)

Tato metrika tedy informuje, kdy začne být viditelný primární obsah stránky (např. nadpis stránky). Metrika FMP se již dnes považuje za nespolehlivou, nahrazuje se metrikou LCP (Michálek, 2019d).

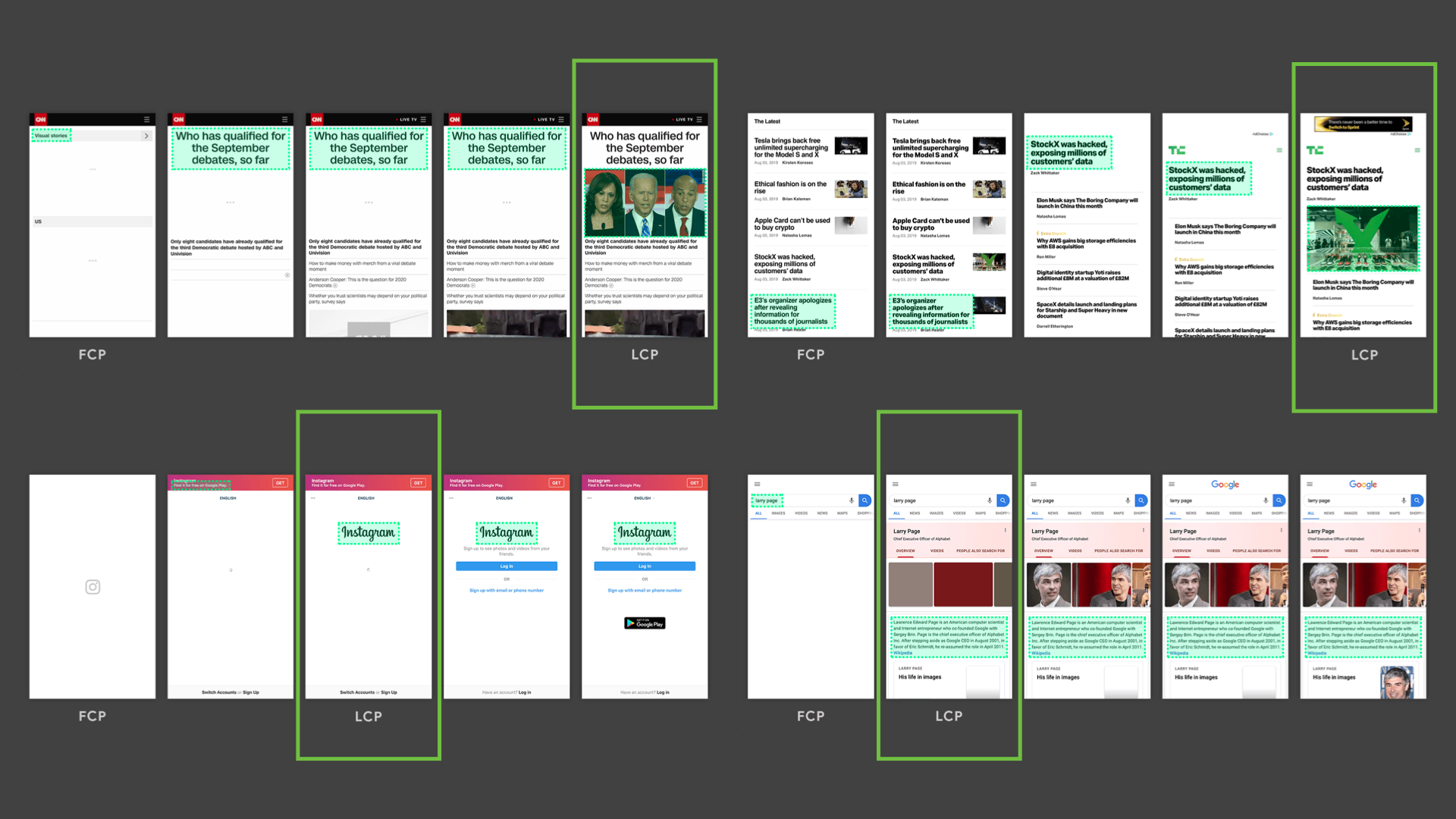
### Largest Contentful Paint (LCP)

„The Largest Contentful Paint (LCP) metric reports the render time of the largest image or text block visible within the viewport.“ (Google LLC, 2020b)

Tato metrika je aktuálně doporučována jako hlavní metrika, kterou měřit, kdy je stránka pro uživatele užitečná. Nahrazuje původně doporučované FMP a SI, které měly problém se spolehlivostí měření, či měřily nerelevantní elementy (Michálek, 2020c).

Měření momentu LCP probíhá tak, že je v každém momentu renderování vybrán tzv. *LCP candidate* – prvek, který byl do dané chvíli největší. *LCP candidate* může být v průběhu renderingu nahrazen jiným větším prvkem, který se stává novým LCP kandidátem. Hledání nových kandidátů LCP končí v momentě, kdy uživatel provede interakci se stránkou (*tapnutí*, *kliknutí*, vstupem z klávesnice nebo *scrollováním* stránky) nebo odejde ze stránky pryč, např. vložením nového URL. Poslední LCP kandidát vyhrává a jako LCP je reportován čas jeho renderování (Michálek, 2020c).

Pro ilustraci lze na Obrázek 3 vidět, kdy nastává FCP a kdy LCP. Také je ilustrován průběh algoritmu pro vyhodnocení největšího prvku na stránce – LCP kandidát je reprezentován zeleným vyplněným obdélníkem.



Obrázek 5 LCP vs FCP (Michálek, 2020c)

### First Input Delay (FID)

„First input delay (FID) measures the time from when a user first interacts with your site (i.e. when they click a link, tap on a button, or use a custom, JavaScript-powered control) to the time when the browser is actually able to respond to that interaction. It is the length of time, in milliseconds, between the first user interaction on a web page and the browser’s response to that interaction. Scrolling and zooming are not included in this metric“. (Mozilla Contributors, 2021a)

Jedná se o událost, kterou ovlivňuje hlavně JavaScript. Stránka je zobrazená a uživatel se snaží kliknout na tlačítko, rolovat s ní… jenže stránka nereaguje, protože hlavní vlákno prohlížeče je zaměstnané zpracováváním kódu. Tuto metriku lze již z definice měřit pouze měřením reálných uživatelů. Obdobou této metriky v syntetickém měření je Time to Interactive (TTI). Použít FID, pokud je to možné, je ale lepší, protože měří odezvu stránky pro reálné uživatele (Michálek, 2019c).

### Time to interactive (TTI)

„TTI measures how long it takes a page to become fully interactive. A page is considered fully interactive when:

* The page displays useful content, which is measured by the First Contentful Paint,
* Event handlers are registered for most visible page elements, and
* The page responds to user interactions within 50 milliseconds.“

(Google LLC, 2019b).

Jedná se tedy o metriku, která říká, kdy je stránka vyrenderována a zároveň schopna spolehlivě reagovat na uživatelský vstup. (Michálek, 2019a)

### Total blocking time (TBT)

„TBT measures the total amount of time that a page is blocked from responding to user input, such as mouse clicks, screen taps, or keyboard presses. The sum is calculated by adding the blocking portion of all long tasks between First Contentful Paint and Time to Interactive. Any task that executes for more than 50 ms is a long task. The amount of time after 50 ms is the blocking portion. For example, if Lighthouse detects a 70 ms long task, the blocking portion would be 20 ms.“ (Google LLC, 2019c)

Tato metrika odpovídá na otázku „Jak moc špatně napsaný JavaScript na stránce je?“. Na tuto otázku již z výše zmíněných metrik snaží odpovídat TTI a FID. Na rozdíl od FID lze ale tuto metriku měřit synteticky. A oproti TTI se do této metriky nepromítá čas odpovědi serveru (TTFB), a jiných vlivů. (Michálek, 2020d)

### Cumulative Layout Shift (CLS)

„CLS measures the sum total of all individual layout shift scores for every unexpected layout shift that occurs during the entire lifespan of the page. A layout shift occurs any time a visible element changes its position from one rendered frame to the next.“ (Walton a Mihajlija, 2020)

Tato metrika se snaží měřit vizuální stabilitu během vykreslování stránky. Problém, který se tato metrika snaží změřit popisuje (Michálek, 2020b): „Stránka vypadá, že je vykreslená… už už se chystáme kliknout… ale v tom se spustí externí skript, posune nám rozvržení a my klikáme na reklamu.“.

### Load

„The load event is fired when the whole page has loaded, including all dependent resources such as stylesheets and images. This is in contrast to DOMContentLoaded, which is fired as soon as the page DOM has been loaded, without waiting for resources to finish loading.“ (Mozilla Contributors, 2020d)

Jedná se o javascriptovou událost, kterou můžeme zachytit *eventListenerem* *„load“*. Jedná se o tradičně nejpoužívanější metriku rychlosti webu. Tato metrika ale neříká zhola nic o uživatelském požitku ze stránky. Když totiž bude stránka zobrazená, interaktivní (a už dávno konzumovaná uživatelem) a na pozadí ještě stahuje velké obrázky do patičky, uživatel o tom vůbec neví. Přitom událost *„load“* se vykoná až po tomto načtení na pozadí, což může v této metrice vycházet dost nehezky (Michálek, 2019g).

### Speed Index

„Speed Index (SI) is a page load performance metric that shows you how quickly the contents of a page are visibly populated. It is the average time at which visible parts of the page are displayed. Expressed in milliseconds, and dependent on the size of the viewport, the lower the score, the better.“ (Mozilla Contributors, 2021c)

Speed Index je navázán na konkrétní technologický kontext – prohlížeč, šířku okna nebo typ připojení. Speed index se měří ze záznamu obrazovky při načítání stránky. Tento způsob sběru není realizovatelný pro RUM, lze jej měřit jen pomocí syntetických testů. Další nevýhodou je, že když jsou na stránce animované prvky, například automatický karusel, speed index narůstá, protože se na stránce stále něco hýbe (Michálek, 2019b).

### Lighthouse Performance Score

Jednoduše řečeno jde o číslo, kterým rychlost vaší stránky zhodnotí testery vycházející z nástroje Lighthouse. Lighthouse Performance Score (LPS) lze považovat za hlavní ze syntetických metrik. Nástroj Lighthouse změří každou z důležitých rychlostních metrik a vytvoří z nich jedno číslo. Metriky obsažené v LPS se snaží co nejlépe pokrýt celou rychlostní část uživatelského zážitku. Metriky a jejich váhy, které LPS sdružuje, se v historii měnily, aby lépe měřily výsledný uživatelský dojem (Michálek, 2020a). Ve své aktuální verzi 6 se LPS skládá takto:

|  |  |  |
| --- | --- | --- |
| Metrika | Váha | Ideální hodnota |
| First Contentful Paint (FCP) | 15 % | ≤ 2 s |
| Largest Contentful Paint (LCP) | 25 % | ≤ 2,5 s |
| Time To Interactive (TTI) | 15 % | ≤ 3,8 s |
| Speed Index (SI) | 15 % | ≤ 4,3 s |
| Total Blocking Time (TBT) | 25 % | ≤ 0,3 s |
| Cumulative Layout Shift (CLS) | 5 % | ≤ 0,1 |

Tabulka 1 Metriky, jejich váhy a ideální hodnoty pro získání 100 % hodnocení v Lighthouse Performance Score od Lighthouse verze 6 (Michálek, 2020a)

# Vyhledávače, SEO a rendering

V první části této kapitoly je definováno, co SEO je a jakou jeho částí se tato práce zabývá. V druhé podkapitole jsou shrnuty teoretické základy on-page SEO optimalizací. Ve třetí podkapitole jsou zanalyzovány různé přístupy k měření SEO. Poslední podkapitola shrnuje současný stav jednotlivých vyhledávačů a robotů, a jak tyto vyhledávače umí, či neumí zpracovávat javascript, na jehož spouštění závisí CSR.

## Definice SEO

Ve článku (Veglis a Giomelakis, 2019) definují autoři SEO takto: „Search engine optimization (SEO) is a collection of strategies that improves a website’s presence and visibility on a search engine’s results page (SERP).“ V článku také identifikují 4 hlavní kategorie SEO optimalizací: „The methods that SEO includes can be divided into four major categories: keyword research/selection, search engine indexing, on-page optimization, and off-page optimization. On-page optimization includes the management of all factors associated directly with someone’s website (e.g., keywords, appropriate content, and internal link structure), while off-page optimization includes all the efforts made away from the website such as link building or social signal strategy.“

Vzhledem k tomu, že se tato práce zabývá vlivem druhu renderingu na SEO, ze všech těchto kategorií jsou relevantní pouze on-page optimalizace týkající se kódu stránky, které budou aplikace v této práci implementovat.

## SEO on-page optimalizace

Shrnout všechny on-page optimalizace by vystačilo na samostatnou diplomovou práci. Každý z vyhledávačů vyhodnocuje u stránky kromě jejího obsahu také různé množství jiných, se stránkou spojených metrik. Například vyhledávač Google bude od nového roku zvýhodňovat stránky s lepším Core web vitals score (Subramanian‎, 2020). V dokumentaci vyhledávačů Seznam či Bing se o ničem takovém zatím nezmiňují. V následující podkapitole jsou popsány pouze některé základní on-page optimalizace, které mají vyhledávače společně. Při rešerši bylo čerpáno především z (Google LLC, 2021d), (Microsoft Corporation, 2021) a (Seznam.cz, a.s., 2021).

### Title tag

Název stránky definujeme pomocí tagu *<title>* uvnitř hlavičky stránky. Ve výsledcích vyhledávání zobrazen jako první a největší prvek pro danou stránku, kliknutím na něj se z vyhledávače dostaneme na požadovanou stránku. Tvoří jej zpravidla několik slov či frází. Měl by stručně a výstižně uživatele informovat o obsahu stránky. Pro každou stránku by měl být jedinečný. (Google LLC, 2021d) (Seznam.cz, a.s., 2021)

### Meta tag description

Meta tag description by měl obsahovat shrnutí v několika větách, o čem daná stránka je. Popis by měl smysluplně rozšiřovat to, co je uvedeno v title stránky. Pro každou stránku by tento obsah měl být odlišný. Stejně jako title tag je umístěn uvnitř HTML hlavičky. Ve výsledcích vyhledávání je z tohoto tagu je vytvořen krátký snippet pod nadpisem stránky (Google LLC, 2021d) (Seznam.cz, a.s., 2021).

### Obsah stránky

#### Outline

Stránka by měla být hierarchicky strukturována pomocí nadpisů a podnadpisů <*h1>* – <*h6>*. Nadpisy by měly postupovat vždy pouze po jedné úrovni níž (*h1*>*h2*>*h3*). Kromě toho by nadpisy neměly být příliš dlouhé a nemělo by jich být na stránce příliš hodně. Mnohdy je potřeba zdůraznit určité části textu v odstavcích, k tomu ale slouží tagy *<strong>*, či *<em>*. Použití nadpisů je v tomto případě nevhodné (Microsoft Corporation, 2021) (Google LLC, 2021d).

#### Obrázky

Všechny obrázky by měly být zobrazeny pomocí HTML tagů *<image>* či *<picture>*. Obrázky by na sobě měly mít také atribut *alt* textově popisující, co se na obrázku vyskytuje. Také názvy souborů obrázků by měly být deskriptivní, a ne příliš dlouhé. Obrázky by ideálně měly být ve formátu JPEG, GIF, PNG, BMP, či WebP. Obrázky, které je potřeba indexovat, je vhodné zařadit do obrázkové sitemap (Microsoft Corporation, 2021) (Google LLC, 2021d).

#### Sémantické HTML elementy

Ve specifikaci HTML5 byly zavedeny nové tagy, které dávají sémantický význam obsahu do nich vnořenému. Jak Google, tak Microsoft doporučují v dokumentacích vyhledávačů jejich používání, (Microsoft Corporation, 2021) uvádí konkrétně tyto tagy: *<article>*, *<aside>*, *<details>*, *<figcaption>*, *<figure>*, *<footer>*, *<header>*, *<main>,* *<mark>*, *<nav>*, *<section>*, *<summary>*, *<time>*.

### Strukturovaná data

Vyhledávače dávají možnost rozšířit zobrazované informace u výsledků pomocí tzv. strukturovaných dat. Pomocí těchto dat můžeme dát vyhledávačům vědět, že daná stránka je například stránka produktu na e-shopu a poskytnout dostupnost a cenu tohoto produktu. (Microsoft Corporation, 2021) uvádí následující formáty dat, v jakých lze strukturovaná data vyhledávačům zveřejňovat:

* HTML Microdata
* JSON-LD
* Microformats
* RDFa
* Open Graph

V dokumentaci pro vyhledávání od Google lze vyčíst již konkrétní entity, které Google vyhledávač umí vyčíst, a jak přesně je pomocí strukturovaných dat popsat (Google LLC, 2021a).

### Přizpůsobení stránky pro mobilní zařízení

Od roku 2016 Google ve svém vyhledávacím algoritmu zohledňuje, jak je daná stránka přizpůsobená pro mobilní zařízení (Google LLC, 2021d). Přizpůsobením se myslí mimo jiné to, že je stránka pohodlně čitelná na zařízeních s menší obrazovkou a ovládací prvky jsou přizpůsobeny na ovládání pomocí dotyku. Google připravil veřejně přístupný vyhodnocovací nástroj (Google LLC, 2021e), pomocí kterého lze zjistit, jestli vyhledávač vnímá stránku jako přizpůsobenou pro mobilní zařízení, popřípadě zjistit konkrétní problémy, které vyhledávač se stránkou má.

### Sitemap

Sitemap je XML soubor, ve kterém jsou specifikovány informace o všech stránkách, či jiných souborech (obrázky, videa), které se na webu vyskytují. Pomocí tohoto souboru lze dát vyhledávači najevo, jaké všechny stránky má indexovat, ba dokonce i jak často je má navštěvovat, aby zaindexoval čerstvý obsah stránky. Přesná specifikaci tohoto soboru je popsána v (sitemaps.org, 2016).

## Metody měření SEO

Hned z počátku je potřeba zmínit, že SEO výkonnost webu nelze měřit exaktně – už jen z prostého důvodu, že přesné algoritmy, pomocí kterých vyhledávací enginy hodnotí jednotlivé weby, jsou neveřejné. Další problém je, že se tyto algoritmy často mění, takže spoustu zdrojů je již nerelevantní. Z provedené rešerše jak akademických článků, tak i článků komerčních firem, byly vybrány následující přístupy pro měření SEO.

### Indexovatelnost

Jedná se o prosté zjištění, zdali vůbec, popřípadě co, je daný robot / vyhledávač z dané stránky schopen vyčíst. Na první pohled může tato metrika působit hloupě, pokud ale daný robot nespouští na stránce javascript, který nastavuje stránce např. *title* tag, měřený titulek ze stránky ani nebude schopen vyčíst.

### Měření počtu přístupů

Pro webovou aplikaci je měřen počet návštěvníků přicházející na dané stránky z vyhledávačů. Jedná se o metriku z praxe, která měří přesně to, proč se vůbec webové stránky pro vyhledávače optimalizují. Tato metrika je doporučována už v historických knihách zaobírajíce se SEO – např. kapitola 21 (Ledford, 2008). Šikovným zkombinováním této metriky s metodologií A/B testů využili inženýři ze společnosti Pinterest, která se až dodnes objevuje velice často mezi prvními výsledky vyhledávání. O tom, jak tuto metriku implementovali, se lze dočíst v  článku (Pinterest Engineering, 2015). Nevýhodou této metriky je, že web již musí mít již určitou návštěvnost z vyhledávačů, aby výsledky získané z tohoto měření byly statisticky relevantní.

### Pořadí stránky ve výsledcích pro zadaný dotaz

Pro daný vyhledávací dotaz je vyhodnocena pozice, na jaké se hodnocená stránka ve vyhledávači vyskytuje. Tato metrika je velmi užitečná v případech, kdy je potřeba porovnání přizpůsobení stránky pro vyhledávače vzhledem ke konkurenci. Jedná se o další z metrik popsanou a doporučovanou v kapitole 21 knihy (Ledford, 2008)

### Google Lighthouse SEO audit

Google v rámci svého nástroje Lighthouse poskytuje SEO audit, který vyhodnocuje danou stránku vzhledem k základním SEO best practices a vypíše nedostatky dané stránky, které lze zlepšit. Tato metrika věrohodně shrnuje jednotlivé on-page optimalizace, které můžeme na stránce jako vývojáři udělat. Byla to také jedna z metrik, kterou použili autoři ve článku (Giannakoulopoulos et al., 2019) při vyhodnocování a porovnání kvality SEO stránek nejprestižnějších univerzit.

## Specifika jednotlivých vyhledávačů

V této podkapitole je snaha shrnout, co o sobě tvrdí jednotlivé vyhledávače a roboti, kteří budou použiti pro vyhodnocování jednotlivých druhů renderingu. Z vyhledávačů byly vybrány Google, Seznam a Bing, z důvodu jejich zastoupení na trhu vyhledávačů v ČR (StatCounter, 2021). Důraz je kladen především na to zda, a do jaké hloubky jsou schopny zpracovávat CSR aplikace, popř. jak ze stránky získávají potřebná data.

### Google

V případě Google je vycházeno především z dokumentace k SEO a javascriptu od Google (Google LLC, 2021f). Google indexuje všechny stránky pomocí aktuálního Chromium-based prohlížeče. Crawler by měl umět přečíst kompletní javascriptem vygenerovaný DOM včetně titulku stránky, či meta hlaviček. Ve své dokumentaci Google explicitně varuje před tzv. soft 404 stránkami – not found stránka by měla vracet správný *HTTP status code* 404, popř. nastavit do hlaviček *meta noindex*. Je také potřeba dbát na korektně vyřešené lazy-loadování ať už obrázků, tak stránek pomocí tzv. *infinite scroll*. Obojí by ale v případě správné implementace měl být Google schopen indexovat. Další z věcí, na které je potřeba při navrhování CSR aplikaci myslet, je správné používání *<a href=“…“>* linků pro navigaci mezi jednotlivými stránkami. Dost často se v CSR totiž stává, že je navigace vyvolána na *click event* na tlačítku, či *<a>* tagu s prázdným *href*. V tomto případě crawler není schopen tuto informaci přečíst.

Teoreticky by tedy Google vyhledávač neměl mít s CSR webovými stránkami problém, stále se ale na internetu objevují různé články tvrdící, že čisté CSR je i v dnešní době pro vyhledávače včetně Google znevýhodňující (Bobbink, 2019). Těmto informacím napomáhá také sám Google, když lze stále na některých místech nalézt doporučení k používání *dynamic renderingu* (Google LLC, 2021c).

### Bing

Bing ve svém Webmaster Guidelines uvádí, že jeho crawler umí zpracovávat a indexovat stránky renderované pomocí javascriptu, má ale problém se škálováním tohoto indexování a doporučuje vývojářům používat dynamic rendering (Microsoft Corporation, 2021).

### Seznam vyhledávač

Seznam vyhledávač se ve své dokumentaci k vyhledávači o podpoře javascriptu / CSR webových stránek nevyjadřuje, lze tedy předpokládat, že jejich crawler takto renderované stránky neumí zpracovávat a indexovat. Tento předpoklad se podařilo potvrdit v (sic již starším) blogovém článku (Jahoda, 2016). Zajímavostí v tomto článku je, že Seznam vyhledávač již zjevně uměl CSR stránku zobrazit v rámci jeho jiného screenshot robota. Náhledy webových stránek již ale na Seznam vyhledávání nejsou v současnosti dostupné, nelze tedy tuto skutečnost ověřit.

### Facebook, Twitter

Facebook i Twitter používají pro generování náhledů stránky Open Graph protokol. Tento protokol specifikuje určité *<meta>* tagy přítomny v hlavičce HTML dokumentu, ze kterých crawlery sociálních sítí čtou potřebná data pro generování náhledu při sdílení URL. Obě sociální sítě neuvádí, že by při crawlování spouštěly také javascript, je tedy předpoklad, že CSR stránky nebudou zobrazovány korektně (Facebook Inc., 2021) (Twitter, Inc., 2021).

# Jednostránková aplikace

## Volba aplikace

Jako jednostránkovou aplikace byla zvolena stránka online farní sbírky pro Komunitní centrum Matky Terezy na Chodově. Diplomová práce je psána v době pandemie Covid-19, kdy spoustu neziskovkám a církvím vypadla velká část příjmů z důvodu zákazu shromažďování. Za těchto okolností zde byl potenciál vytvořit aplikaci pro fundrasing, na kterou by bylo v této době mnohem snadnější dostat reálné uživatele, protože většina aktivit se přesunula do online prostředí. Z reálných uživatelů pak lze pro tuto práci dostat mnohem přesnější a věrohodnější data pro porovnání jednotlivých metod renderingu. Stránka pro fundrasing je vhodná také proto, že obsahuje dobrý poměr statického textu a interaktivity reprezentovanou formulářem pro zadání částky a vygenerování platby. Volba na Komunitní centrum již padla hlavně proto, že se jedná o jeden z nejnavštěvovanějších církevních webů v ČR (NAVRCHOLU.cz, 2021), takže je potenciál přivést některé z návštěvníků na tuto aplikaci. Dalším z důvodů bylo, že se autor práce osobně zná s tamním farářem, takže je jednodušší vše zorganizovat a uvést do reálného provozu.

## Popis aplikace

Aplikace bude umět na základě vybrané částky a případných kontaktních údajů vygenerovat QR kód pro platbu. Při analýze bylo zvažováno zaintegrovat přímo platební bránu podporující také platební karty, nakonec ale z důvodu složitosti schvalovacích procesů pro neziskové organizace ze strany poskytovatelů platebních bran bylo vybráno toto řešení.

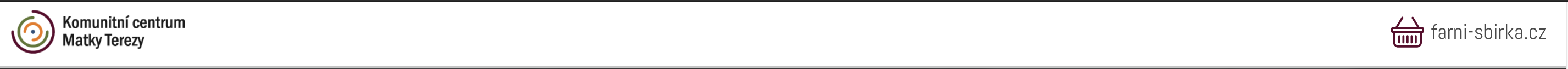
Farnost chtěla z účetních důvodů rozdělit sbírku na dva druhy – nedělní sbírku a dar farnosti. Tyto sbírky se liší variabilním symbolem a zprávou v posílané platbě. Pro dar farnosti je kromě částky potřeba také do zprávy platby přidat jméno a adresu dárce.

Aplikaci tvoří jedna stránka, na níž je krátký popis a informace o farní sbírce jako takové a popis jednotlivých druhů sbírky. Stránka dále obsahuje formulář pro zadání částky a případných kontaktních údajů. Kromě toho je vedle formuláře uveden seznam nadcházejících událostí ve farnosti. Tento seznam je na hlavním webu farnosti ne příliš dobře přístupný, byl jsem požádán, jestli by šlo z hlavního webu tyto události vyčíst a na stránce sbírky zobrazit.

## Specifikace aplikace

### Hlavička

Hlavička obsahuje pouze 2 loga. První je logo farnosti, které odkazuje na web farnosti. Druhé je logo webu *farni-sbirka.cz*, které vede na příslušný web.



Obrázek 6 Jednostránková aplikace: hlavička

### Informace o sbírkách

Tato část má za cíl informovat návštěvníky o důvodu vzniku webu a vysvětlit rozdíly mezi jednotlivými druhy sbírek. U každého druhu sbírky je také tlačítko přispět, které přejede o stránku níže na formulář pro zadání platby pro vybranou sbírku. Na pozadí této sekce bude fotografie farního sálu.

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 7 Jednostránková aplikace: informace o sbírkách

### Formulář pro vytvoření platby

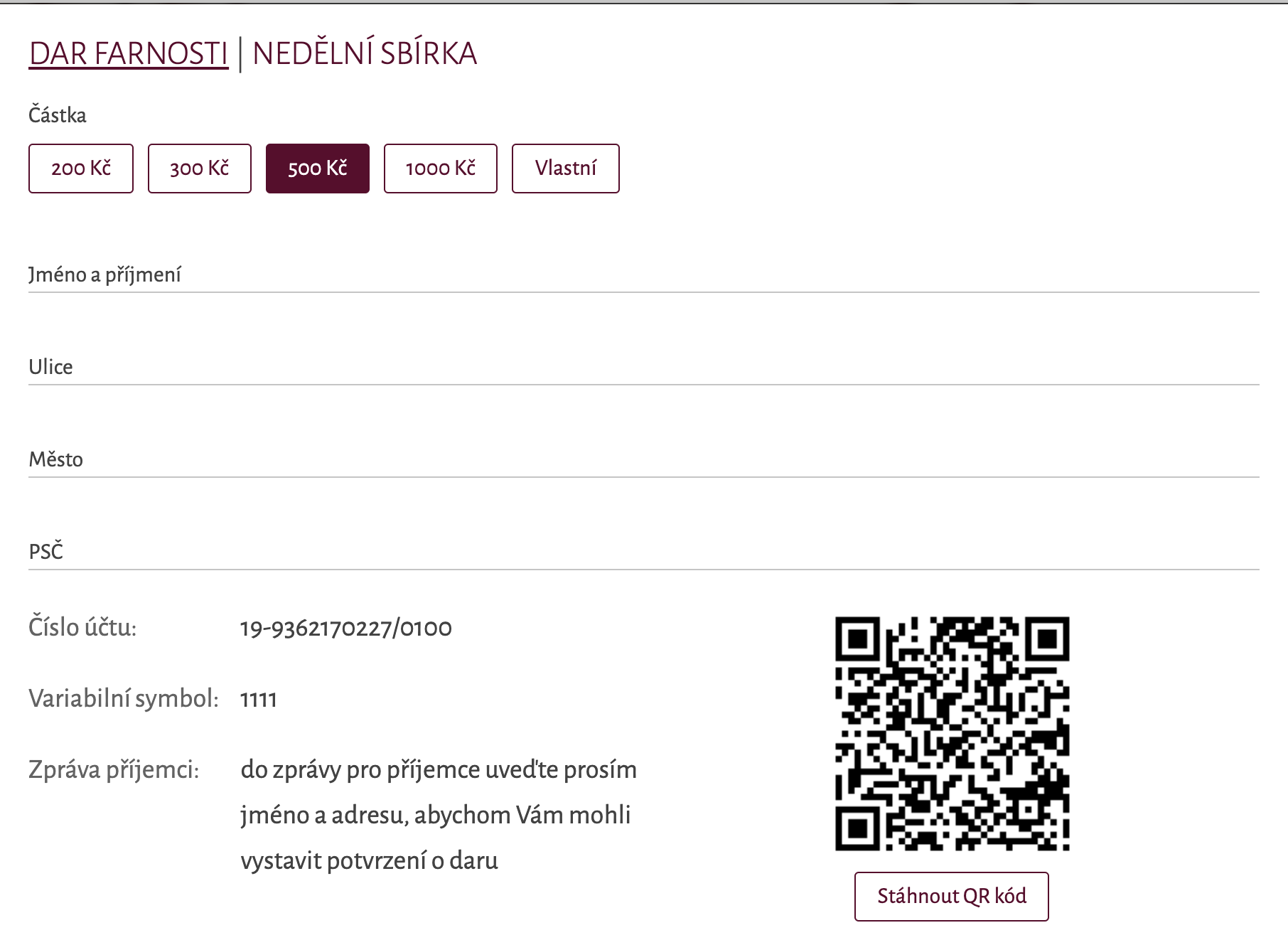
Jak již bylo napsáno výše, pomocí této aplikace lze přispět dvěma způsoby – jako dar farnosti, nebo nedělní sbírku. Pro každý z těchto způsobů bude lehce odlišný formulář pro vytvoření platby a také jiné parametry platby.

V horní části formuláře je přepínač mezi jednotlivými způsoby. Kromě samotného formuláře pro vyplnění částky a případných dalších informací je zobrazen vygenerovaný QR kód pro platbu. Pod QR kódem bude tlačítko umožňující stažení tohoto kódu na zařízení. Tohoto využijí především návštěvníci na chytrých telefonech, kteří chtějí zaplatit bankovní aplikací nainstalovanou na tom samém telefonu na kterém mají zobrazenou tuto stránku. Vedle QR kódu musí být také zobrazeny konkrétní platební údaje pro daný druh sbírky pro případ, že by uživatel nechtěl QR platbu využít.

#### ****Dar farnosti****

Pro dar farnosti jsou předvoleny k výběru částky 200, 300, 500 a 1000 Kč. Číslo účtu farnosti pro platbu je *19-9362170227/0100*, variabilní symbol *1111*.

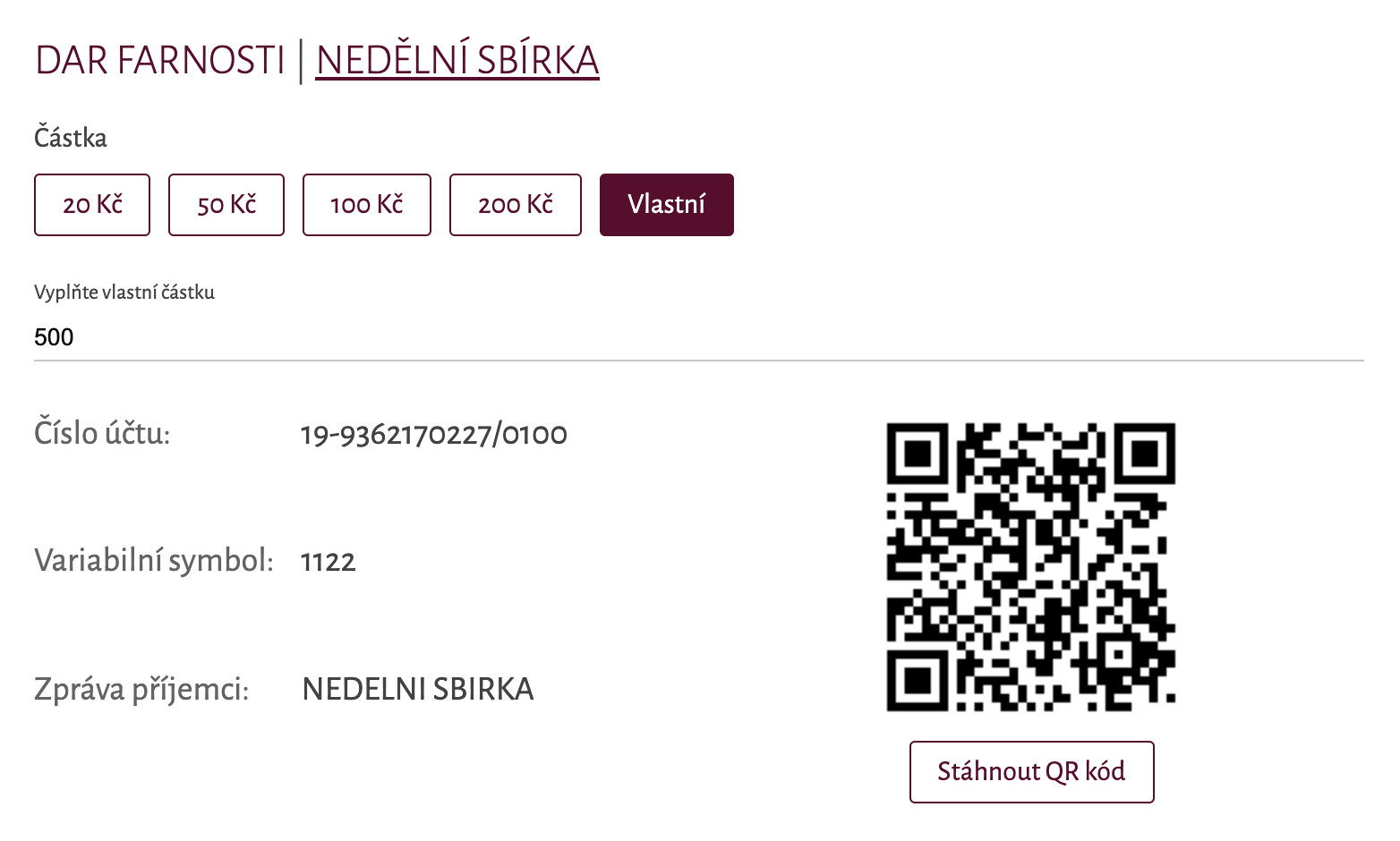
Pro dar farnosti je také potřeba do zprávy pro příjemce uvést jméno a adresu dárce, proto tento druh sbírky bude obsahovat také formulář pro vyplnění těchto údajů. Tyto údaje poté předáme do zprávy příjemci.



Obrázek 8 Jednostránková aplikace: formulář pro zadání platby – dar farnosti

#### Nedělní sbírka

Nedělní sbírka má předvolené částky v hodnotách 20, 50, 100 a 200 Kč. Číslo účtu je stejné, liší se pouze variabilní symbol *1122* a ve zprávě příjemci bude uvedeno *NEDELNI SBIRKA*.



Obrázek 9 Jednostránková aplikace: formulář pro zadání platby – nedělní sbírka

### Výpis akcí ve farnosti

Farnost na svém webu uveřejňuje různé události a akce, které pořádá (Komunitní centrum Matky Terezy, 2020). V rámci implementace této aplikace by měly být tyto události každý den z webu farnosti *vyscrapovány* a aplikace jako taková by měla všechny budoucí události vypisovat. Web farnosti bohužel nemá žádné API, obsah je potřeba vyčíst přímo z webové stránky.

Každá událost obsahuje název, který je zároveň odkazem na detail události na farním webu. Dále je u události uvedeno datum a místo konání události, fotografie a krátký útržek popisu události.

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 10 Jednostránková aplikace: výpis akcí ve farnosti

# Specifikace e-shop aplikace

## Volba aplikace

Jako druhá aplikace byl zvolen e-shop fiktivního zahradnictví. Na rozdíl od aplikace farní sbírky již tento e-shop není určen pro reálnou entitu. Volba zahradnictví nemá žádný hlubší význam, produkty a jejich popisy budou za účelem měření náhodně generovány z latinských slov – což v případě rostlin nebude až tak vadit.

## Popis aplikace

E-shop se všemi funkcionalitami je poměrně komplikovaná aplikace. Pro potřeby této práce není potřeba programovat všechny funkce, které by reálný e-shop musel mít. Explicitně tedy není potřeba implementovat části vytvoření, správa a zobrazení objednávek, správu uživatelů. Dále lze nahradit v jednotlivých statických stránkách jako jsou obchodní podmínky text těchto stránek za pouze náhodně generovaný text.

Obchod bude obsahovat zahrádkářské zboží zařazené do kategorií, které lze přidat do košíku. Aplikace jako taková tedy bude obsahovat pět druhů stránek: hlavní stránku, detail kategorie, detail zboží, košík a statickou stránku (např. obchodní podmínky).

V rámci specifikace byl vytvořen wireframe (Příloha B), který může lépe vizualizovat finální představu aplikace jako takové.

## Specifikace komponent aplikace

### Záhlaví

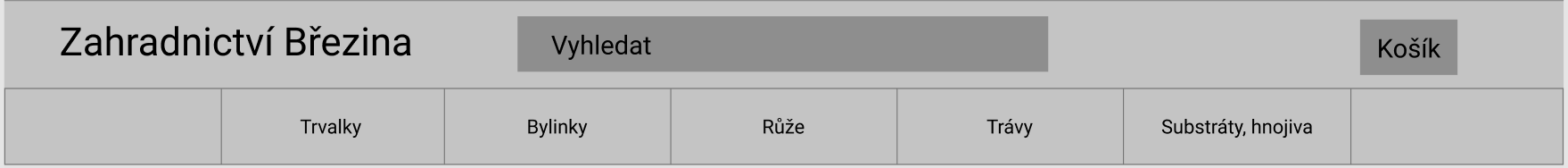
V záhlaví budou zobrazeny základní informace důležité pro zákazníka. Budou zde uvedeny odkazy na stránky typu statická stránka popisující přepravu rostlin, informace o zahradnictví, kontakty a obchodní podmínky. Kromě toho je v záhlaví také uvedeno telefonní číslo na zákaznickou podporu a otevírací doba.



Obrázek 11 E-shop: záhlaví

### Hlavička

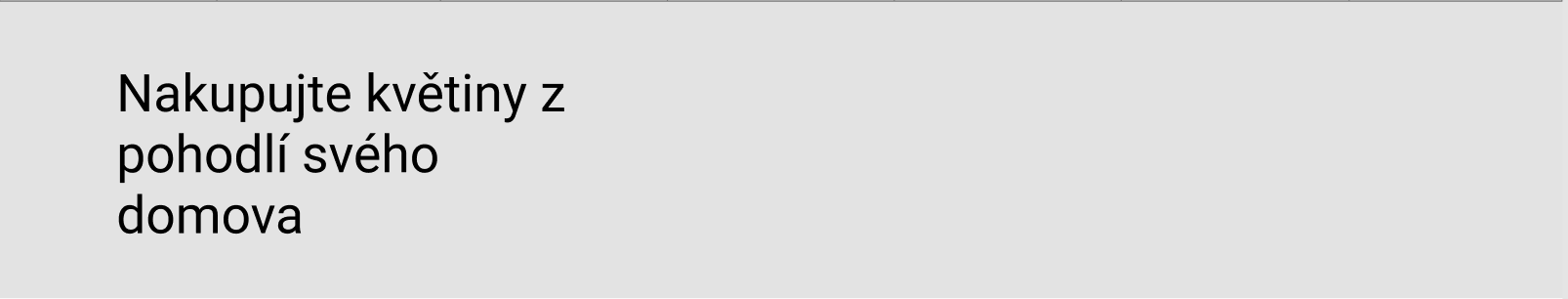
Hlavička obsahuje logo zahradnictví, vyhledávácí pole, položku košíku a menu odkazující na stránky detailů jednotlivých kategorií na e-shopu. Logo zahradnictví je zároveň odkazem na domovskou stránku. Vyhledávací pole bude pouze textový input, vyhledávání jako takové nebude implementováno. Kliknutím na košík se dostaneme na stránku košíku. Zároveň je na tomto odkazu zobrazen aktuální počet položek v košíku. Přidání / odebrání položek z košíku musí aktualizovat toto číslo.



Obrázek 12 E-shop: hlavička

### Banner

Banner je jednoduchá komponenta obsahující pouze nadpis na pozadí s fotografií. Tato fotografie bude přizpůsobena formátu metodou *cover*.



Obrázek 13 E-shop: testimonial

### Karta produktu

Karta produktu obsahuje fotografii, nadpis třetí úrovně s názvem, krátký popis, cenu, informaci o dostupnosti, input a tlačítka k navolení počtu kusů přidávaných do košíku a tlačítko „Přidat do košíku“.

Horní polovina kartičky (fotka, název, popis, cena, dostupnost) tvoří odkaz na detail produktu. Fotografie produktů jsou rozměrově přizpůsobeny požadovanému formátu ořezáváním metodou *cover*. V případě, že název produktu přetéká jeden řádek, nadpis je zkrácen třemi tečkami tak, aby nepřetékal. Krátký popis je tímtéž způsobem zkrácen na maximální délku dvou řádků. Informace o dostupnosti může nabývat hodnot „skladem“, „brzy na skladě“, „nedostupné“. K navolení počtu kusů přidávaných do košíku můžeme buď použít tlačítka +/- či napsat konkrétní počet kusů do inputu. Počet přidávaných kusů musí být číslo větší než 0. Tlačítkem „Přidat do košíku“ se přidá zvolený počet kusů produktu do košíku.



Obrázek 14 E-shop: Karta produktu

### Stránkovač

Stránkovač má na starosti zobrazit a přepínat se mezi stránkami při výpise produktů. Stránkovač tvoří jednotlivá čísla stránek, vždy maximálně šest. Kliknutím na číslo stránky aplikace zobrazí požadovanou stránku. Pokud je počet stran větší než šest, skryjeme čísla za tři tečkami – viz Obrázek 7.

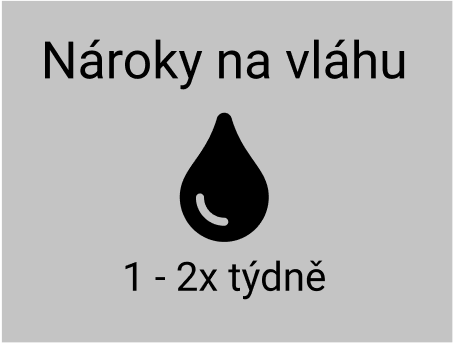
V případě mobilní verze stránky čísla nahradíme pouze tlačítky „Předchozí“ a „Další“, kterými se bude přepínat mezi stránkami.



Obrázek 15 E-shop: stránkovač

### Karta vlastnost produktu

Komponenta má za cíl zobrazit vlastnost produktu, jako jsou nároky na vláhu apod. Karta se skládá z nadpisu vlastnosti, textu s hodnotou pro danou vlastnost a volitelnou ikonou dané vlastnosti. V případě karty s ikonkou je text vlastnosti uveden pod ikonkou. V případě karty bez ikonky je text vlastnosti vycentrován na střed karty.



Obrázek 16 E-shop: Karta vlastnost produktu s ikonkou



Obrázek 17 E-shop: Karta vlastnost produktu bez ikonky

### Box přidat do košíku

Box slouží především k tomu, aby uživatel mohl přidat vybrané zboží do košíku. Kromě ceny, tlačítek k navolení počtu kusů a tlačítka pro přidání do košíku, které fungují úplně stejně jako v komponentě *Karta produktu*. Kromě toho box také obsahuje informaci o počtu kusů na skladě a předpokládanou dobu doručení.

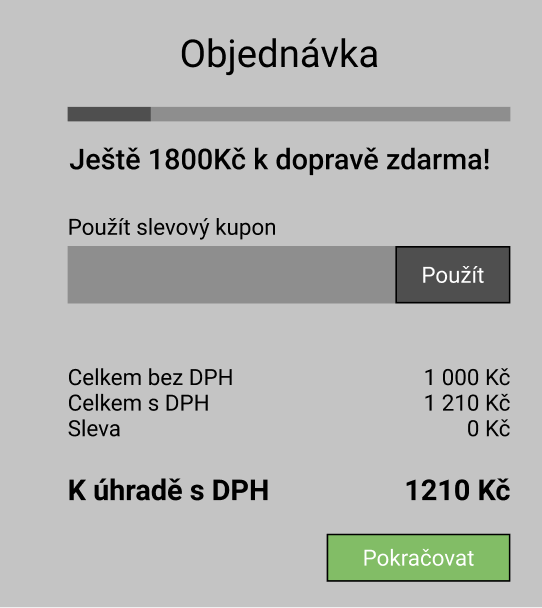
Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 18 E-shop: box přidat do košíku

### Box objednávka

Tato komponenta shrnuje stav košíku a umožňuje přejít k objednání zboží. Předpokládáme, že na e-shopu bude určitý limit pro poštovné zdarma. V horní části boxu je *progress bar* vizualizující tento limit vzhledem k aktuálnímu zboží v košíku. V prostřední části je textové pole s tlačítkem pro aplikování slevového kuponu na danou objednávku. V dolní části je již souhrn částky k úhradě a tlačítko pokračovat, které by v reálné aplikaci vedlo na zadávání adresy a vytvoření objednávky.



Obrázek 19 E-shop: box objednávka

### List produktů v košíku

Komponenta list produktů v košíku má za cíl zobrazit jednotlivé položky v košíku a upravit jejich objednávané množství. Zobrazeny jsou jednotlivé položky odděleny horizontální čárou. Každá položka obsahuje obrázek produktu, název, stav zboží na skladě, *input* a tlačítka k navolení počtu kusů, cenu za kus a tlačítko pro odebrání produktu z košíku.

Obsah obrázku text

Popis byl vytvořen automaticky

Obrázek 20 E-shop: list produktů v košíku

### Patička

Patička obsahuje tři sloupce s užitečnými informacemi týkajících se e-shopu. V prvním sloupci je sekce kontakty obsahující telefonní číslo a kontaktní email. Dále je zde uvedena adresa pobočky e-shopu, kde lze zboží vyzvednout osobně. V druhém sloupci je uvedena otevírací doba zahradnictví pro jednotlivé dny. V třetím sloupci je seznam odkazů na stránky typu Statická stránka obsahující užitečné informace jako obchodní a reklamační podmínky, informace o přepravě a podobně.

Ve mobilní verzi se patička přeskládá do jednoho sloupce.

Obsah obrázku stůl

Popis byl vytvořen automaticky

Obrázek 21 E-shop: patička

## Specifikace jednotlivých stránek aplikace

Všechny níže uvedené stránky obsahují záhlaví, hlavičku a patičku. Liší se pouze obsahem mezi hlavičkou a patičkou. V následující podkapitole je detailně popsáno, spolu s dalšími požadavky na stránky, jak.

### Domovská stránka

Na domovské stránce máme komponentu *Banner* obsahující slogan zahradnictví. Na pozadí banneru bude fotografie zahradnictví. Pod bannerem máme seznam obchodních kampaní. Každá taková kampaň je tvořena nadpisem a čtyřmi vybranými produkty zobrazenými pomocí komponenty *Karta produktu*.

*Title* stránky je „Zahradnictví Březina“.

### Detail kategorie

Detail kategorie má v horní části komponentu *Banner* obsahující v nadpisu název dané kategorie s pozadím vybraným pro danou kategorii. Pod bannerem je již výpis jednotlivých produktů stránkovaný pomocí komponenty Stránkovač.

*Title* stránky obsahuje „Název kategorie | Zahradnictví Březina“. Stránka pomocí vhodných og tagů sdílí fotografii dané kategorie. Stránkování správně nastavuje <link rel=“prev/next/canonical“>.

### Detail produktu

Detail produktu obsahuje fotografii produktu. Kromě toho obsahuje také v horní části v nadpisu název produktu a krátký útržek detailního popisu produktu. Pod popisem obsahuje komponenty *Karta vlastnost produktu* pro každou vlastnost zobrazovaného produktu. Na stránce detailu produktu bude také *Box přidat do košíku*. Ve střední části stránky se vyskytuje úplný popis produktu. Pod tímto popisem jsou čtyři podobné produkty k právě vybranému produktu zobrazené pomocí komponent *Karta produktu*.

Stránka má správně nastaveny vhodné mikroformáty (*JSONLD product*, *JSONLD breadcrumb*s nadřazenou stránkou detailu kategorie produktu, *og:image* s obrázkem produktu) a *title* jako „Název produktu | Zahradnictví Březina“.

### Košík

Stránka košíku má zrekapitulovat zboží, které si zákazník během návštěvy e-shopu vybral k nákupu. Kromě toho umožňuje upravit počet položek v košíku pro jednotlivé produkty, či tyto produkty z košíku úplně odstranit. Zákazník může také na svůj košík uplatnit slevový kupon. Všechny tyto změna se musí propisovat do další části stránky – shrnutí objednávky, kde je vypsána celková cena objednávky. Z této stránky by zákazník dále pokračoval k vytvoření objednávky. Vytváření objednávky v této aplikaci již implementováno nebude, tlačítko pro vytvoření objednávky tedy nebude nikam odkazovat. Stránka se tedy bude skládat z komponent *List produktů v košíku* a *Box objednávka*.

### Statická stránka

Hlavním účelem statické stránky je zobrazit textový obsah, který provozovatel e-shopu chce, či má povinnost na své stránce zveřejnit. Mezi takové stránky budou v této aplikaci patřit například stránky kontaktů, obchodních, reklamačních a dopravních podmínek apod. Protože se nejedná o e-shop reálné entity, bude na těchto stránkách pouze vygenerovaný text.

Statická stránka jako taková bude mít v záhlaví komponentu *Banner* obsahující název stránky. Pod tímto banerem se již vyskytuje html, které by definoval samotný provozovatel v e-shopu.

# Metodologie testování a vyhodnocování

## Testovací aplikace

Jednotlivé druhy renderingů jsou měřeny, porovnávány a vyhodnocovány na implementaci dvou aplikací specifikovaných v kapitolách 5 a 6.

Tyto dvě testovací aplikace jsou zvoleny zejména proto, že každý druh renderingu má svá určitá specifika. Pro různé druhy webových aplikací může být vhodné použít odlišné druhy renderingů. Za cíl bylo vybrat reprezentativní druhy webových aplikací, ve kterých záleží na SEO, a ze kterých by šlo odvodit obecnější závěry pro nějakou kategorii těchto aplikací. Jak pro e–shop, tak landing pages je SEO a výkon webových aplikací důležitý – zlepšení výkonnosti se přímo projevuje na ziscích generovaných touto aplikací (Wagner et al., 2020), (web.dev, 2020). Záměrně nebyly vybrány aplikace, jejichž hlavní část je přístupná pouze pro přihlášené uživatele (administrace, sociální sítě). I když se jedná o další signifikantní část používaných webových aplikací, ve kterých se může projevit síla client-side rendered aplikací, nezáleží u nich na SEO, kterým se tato práce také zabývá.

Při specifikování reprezentativních druhu webových aplikací byly po konzultacích vybrány tyto kategorie: e-shop a jednostránková webové aplikace s interaktivními prvky. Lze si představit např. landing page nějaké firmy, či produktu. Výběr právě těchto druhů byl konzultován s vedoucím práce Ing. et Ing. Stanislav Vojířem, Ph.D a frontend leaderem ve společnosti COEX s.r.o. Ing. Lukášem Bartákem.

## Metody renderingu

Z metod renderingu popsaných v kapitole 1 jsou pro testování a vyhodnocování vybrány:

* client-side rendering
* client-side rendering with prerendering
* server-side rendering s rehydratací
* server rendering

Právě tyto metody jsou zvoleny proto, že reprezentují všechny hlavní odnože možných druhů renderingů popsaných v článku (Miller a Osmani, 2019) vhodné pro zvolené druhy aplikací. Dalším důvodem je také to, že jsou právě tyto metody implementovány v hlavních javascriptových frameworcích – VueJS (VueJS, 2021), React (Next.js, 2021) a Angular (Angular/universal, 2021).

## Výkonnostní metriky

Pro porovnání výkonnostních kvalit jednotlivých metod renderingu jsou využita měření jak syntetická, tak měření na reálných uživatelích. Obě aplikace ve všech vybraných metodách renderingu je za cíl změřit a porovnat především pomocí metrik web vitals. V syntetických měřeních se jedná o FCP, LCP, SI, TTI, TBT, CLS a Lighthouse performance score. V měření reálných uživatelů jsou zaznamenávány FCP, FID, CLS, LCP a TTFB. Jak jsou tyto metriky definovány a co se snaží měřit je podrobněji rozepsáno v kapitole 2. Také jsou tam popsány odlišnosti a specifika měření syntetického a měření na reálných uživatelích.

### Syntetické měření pomocí Lighthouse

Měření probíhá lokálně na Macbooku Pro 2018 v prohlížeči Google Chrome verze 89.0.4389.114. Měření je prováděno pro každou aplikaci, druh renderingu a Lighthouse přepínač *Mobile*/*Desktop* 5x, výsledná hodnota pro každou takovou konfiguraci je zvolena jako medián těchto 5 hodnot měření. Tento postup měření je zvolen především proto, že se naměřené hodnoty při stejné konfiguraci z nejrůznějších důvodů liší. Tento způsob korekce a zpřesnění měření je doporučen přímo autory tohoto nástroje (Google LLC, 2019)

### Syntetické měření pomocí Pagespeed.cz

Pagespeed měří každý den jednotlivé aplikace a druhy renderingu pomocí PageSpeed Insights API přes Amazon Web Services. PageSpeed Insights API měří pomocí emulace zařízení Moto G4 a napodobuje mobilní připojení 3G. Výsledné skóre je 75. percentil pro sadu měření (PageSpeed.cz, 2021). Tento způsob měření byl zakomponován hlavně proto, že měření probíhá pomocí počítačů běžících v datacentrech Amazonu, kde je předpoklad pro mnohem stabilnější měřící prostředí – ať už z důvodu internetového připojení, tak např. absence procesů běžících na pozadí operačního systému osobního počítače. Dalším důvodem také je, že konfigurace tohoto měření je snadná.

### Měření reálných uživatelů

Měření reálných uživatelů probíhá pomocí knihovny *web-vitals* (Google LLC, 2021b), která je pro sběr těchto dat doporučena. Uživatelům je náhodně vybrán druh renderingu, pomocí kterého je mu stránka vyrenderována. Pro sbírání naměřených dat je v rámci této práce naimplementován vlastní endpoint, na který jsou z prohlížečů uživatelů odesílány jednotlivé naměřené hodnoty a který tyto data následně ukládá do Google tabulky. Implementační detaily tohoto řešení jsou podrobněji rozebrány v podkapitole 7.1.

Protože reálné uživatele má v rámci této práce pouze jednostránková aplikace farní sbírky, je měření reálných uživatelů provedeno pouze pro tuto aplikaci.

## Měření SEO

V rámci měření SEO není v této práci cílem měřit kvalitu přizpůsobení dané aplikace pro vyhledávače. Za cíl je spíše zjistit, jak si stojí jednotlivé druhy renderingu ve výsledcích vyhledávání proti sobě. Dalším výstupem měření je také zjistit, co všechno umí vyhledávače a sociálně z těchto různých druhů renderingu vyčíst. Tímto měřením je zejména snaha odpovědět na otázku, co konkrétně nezvládnou vyhledávače a sociální sítě ze stránky vyčíst, pokud je pouze client-side rendered.

Měření je prováděno pouze na e-shop aplikaci, protože na rozdíl od jednostránkové aplikace se jedná o fiktivní stránku, kde si lze dovolit indexovat každou verzi renderingu na zvláštní URL.

Měřit a testovat SEO se bude v jednotlivých rendering verzích detailu produktu. Každý druh renderingu bude testován ve vyhledávačích Google, Bing a Seznam. Kromě toho bude stránka detailu produktu otestována, jak se pro každý druh renderingu zobrazuje v Google product snippetu a snippetu pro nový příspěvek na Facebooku.

### Pozice ve vyhledávačích pro dosud neexistující produkty

V databázi se vytvoří produkt s názvem, na který má každý z vyhledávačů 0 výsledků. Každý druh renderingu má různou URL pro každý z produktů, po úspěšné indexaci by se tedy měly po zadání tohoto názvu zobrazit 4 výsledky. Výsledkem tohoto měření je pořadí jednotlivých metod ve výsledcích vyhledávání v anonymním okně. Aby byly výsledky tohoto měření přesnější, provedeme toto měření pro 5 takovýchto různých produktů.

### Výsledek vyhledávání

Jedná se pouze o vizuální porovnání toho, co se zobrazí pro jednotlivé druhy renderingu ve výsledcích vyhledávání. Testováno pro domovskou stránku, detail kategorie a detail produktu.

### Google product snippet

Zatím ani nevím jestli chci, možná že to bude úplně na nic – jedná se o strukturovaný data, který si google zvládne vyčíst když přečte JS… Možná bude stačit jen uvést u Výsledku vyhledávání, že Google umí vyčíst structured data ze všech renderingů

### Facebook snippet

Vizuální porovnání náhledů stránek jednotlivých renderingů při tvoření nového příspěvku na Facebooku. Testováno na domovské stránce, detailu kategorie a detailu produktu.

# Implementace

Pro zvolené druhy renderingů je potřeba každou aplikaci naimplementovat dvakrát – jednou pro server-rendering verzi, podruhé CSR, SSR a prerendering verzi. Kromě těchto dvou verzí eshopu a webu farní sbírky bylo naimplementováno ještě několik dalších pomocných aplikací, které byly k funkčnosti těchto aplikací, či pro potřeby měření třeba.

## Technologie pro implementaci

Všechny aplikace implementované v rámci této práce jsou napsány v jazyce Javascript, resp. Typescript. Stejný jazyk na serveru byl použit hlavně proto, aby si implementace na klientské a serverové straně byly po všech stránkách co nejbližší.

CSR, SSR a prerendering verze aplikací jsou implementovány ve frameworku Angular. Jednotlivé verze renderingu jsou vygenerovány odlišným buildingem stejné Angular aplikace. Pro implementaci by šly dobře použít také konkurenční ReactJS, či VueJS, které také podporují všechny tři druhy renderingu. Angular byl vybrán zejména proto, že s ním má autor práce zkušenosti, obzvláště se zprovozněním SSR a prerenderingu, které může být ve všech třech alternativách komplikované.

Server-rendered verze je naimplementována ve frameworku NestJS. Volba na tento Framework padla proto, že je architekturou kódu inspirován frameworkem Angular. Díky tomu bylo možno použít některé části zejména modelové části aplikace pouze s lehkými úpravami.

Pro hostování aplikací je použit Firebase od Google. CSR a prerendering verze jsou ve Firebase hostingu, což je CDN pro statické soubory. Serverové aplikace jsou hostovány pomocí Firebase functions. Velkou výhodou této volby je jednoduché nasazení aplikace, nízké náklady a řešení funguje v podstatě bezúdržbově. Nevýhodou je tzv *cold start* – při prvním zavolání existuje zpoždění, než se funkce nastartuje a odpoví.

Pro pomocné části aplikace, které by byly choulostivé na výše zmíněný *cold start* a byly využity Cloudflare functions, které fungují na jiném principu slibující spouštění bez *cold startu*. Důvod, proč toto řešení nebylo použito pro SSR a SR je, že Cloudflare functions implementují jen určitou podmnožinu javascriptu – např. nepodporují *XMLHttpRequest* API. Místo něj je nutné využívat *fetch* API, což kód vygenerovaný z Angularu, či různé knihovny použité v NestJS nesplňují.

## Instalace, spuštění, nasazování

Všechny aplikace implementovány v této práci fungují s *NodeJS v14.8.0* a *yarn v1.22.4*. Dost pravděpodobně fungují i v jiných verzích *Node*/*Yarn*. Postup pro instalaci, spuštění a nasazení byl sjednocen pro všechny aplikace v této práci. Ve složce každé konkrétní aplikace (obsahující *package.json*) fungují následující příkazy:

yarn # nainstaluje potřebné knihovny a závislosti aplikace

yarn start # spustí aplikaci pro vývoj

yarn deploy # nasadí projekt do firebase/cloudflare functions

Nasazení aplikace do konkrétních projektů úspěšně proběhne pouze pokud je správně nastaveno přihlášení do firebase/cloudflare CLI a uživatel má práva k nasazování v konkrétních projektech.

## Jednostránková aplikace

Pro funkčnost aplikace farní sbírky bylo kromě dvou verzí frontendové části aplikace potřeba naimplementovat pomocné aplikace pro měření výkonnostních metrik na reálných uživatelích. Konkrétně bylo potřeba vytvořit *A/B proxy* pro přidělení a vykreslení náhodného druhu renderingu nově příchozímu uživateli a endpoint, na který jsou nasbíraná data uživatelů odesílána a který tyto data ukládá. Dále bylo potřeba naprogramovat *scraper*, který bude pravidelně stahovat z stránek farnosti nadcházející události a uloží je do databáze. Poslední pomocnou aplikací je endpoint, který bude události z této databáze vracet FE aplikacím.

Aplikace v současné době funguje na <https://kcmt.farni-sbirka.cz/>.

### A/B proxy

A/B proxy je jednoduchá cloudflare funkce (HTTP endpoint), skrz kterou jdou všechny http požadavky na aplikaci farní sbírky. Z cookies HTTP požadavku vyčte cookie *rendering\_choice* obsahující požadovanou verzi renderingu. V případě, že cookie není přítomna, nebo obsahuje nevalidní hodnotu, náhodně se vygeneruje a do cookie uloží jedna z dostupných verzí renderingu. Funkce pak zduplikuje požadavek na konkrétní doménu obsahující požadovanou verzi renderingu a vrátí výsledek tohoto požadavku.

Implementaci proxy lze nalézt v Příloha C, konkrétně v souboru *donate\_app/ab\_proxy/src/handler.ts.*

### Sbírání metrik reálných uživatelů

Měření jednotlivých výkonnostních metrik je měřeno pomocí knihovny *web-vitals*. Do hlaviček každé z FE aplikací byl přidán následující skript (Výpis 7.1), odesílající tyto naměřené hodnoty na endpoint <https://metrics-endpoint.brezina.workers.dev/>. Za zmínku stojí, že k odesílání těchto metrik je použita metoda *navigator.sendBeacon()*, které je k odesílání těchto diagnostických dat určeno.

Výpis . Sběr a odeslání naměřených výkonnostních metrik

<script>

addEventListener('DOMContentLoaded', function () {

const endpoint = 'https://metrics-endpoint.brezina.workers.dev/';

function sendToAnalytics({ name, value }) {

const body = JSON.stringify({

name,

value,

renderingType: document.cookie.match(/rendering\_choice=([^;]\*)/)[1],

userAgent: navigator.userAgent

});

// Use `navigator.sendBeacon()` if available, falling back to `fetch()`.

(navigator.sendBeacon && navigator.sendBeacon(endpoint, body)) ||

fetch(endpoint, { body, method: 'POST', keepalive: true });

}

webVitals.getCLS(sendToAnalytics);

webVitals.getFID(sendToAnalytics);

webVitals.getLCP(sendToAnalytics);

webVitals.getFCP(sendToAnalytics);

webVitals.getTTFB(sendToAnalytics);

});

</script>

Endpoint pro sbírání metrik běží na Cloudflare functions. Tento endpoint po zavolání pomocí *Google Spreadsheets API* uloží do Google tabulky naměřené informace včetně informace, o jaký druh renderingu se jedná (rozpoznaný z cookie) a základních informacích o prohlížeči a operačním systému vyčtených z hlaviček požadavku.

Implementaci endpointu a ukládání dat lze nalézt v Příloha C, konkrétně v souboru *donate\_app/metrics-endpoint/src/handler.ts.*

Obsah obrázku stůl

Popis byl vytvořen automaticky

Obrázek 22 Ukázka Google tabulky se záznamy výkonnostních metrik

### Farní události

Výpis události ve farnosti je přístupný na adrese <https://www.kcmt.cz/kalendar-akci/?allInMonth=1&year=2021&day=1&month=4> . V rámci této práce byl naprogramován scraper, který každý den v určitou dobu navštíví výpis událostí pro 3 měsíce dopředu a uloží do databáze jejich názvy, obrázky, datum a místo konání, popis a odkaz na detail události.

Scraper byl naimplementován jako Cloudflare funkce. Výhodou použití tohoto poskytovatele bylo nastavení pravidelného spouštění každý den v rámci dvou řádku kódu (soubor *wrangler.toml*). Pro ukládání vytěžených dat byla použita integrovaná databáze *Cloudflare workers KV store*. Data jsou uložena uvnitř namespace *EVENTS\_KV*.

Implementace scraperu lze nalézt v Příloha C, konkrétně v souboru *donate\_app/metrics-endpoint/src/handler.ts*.Implementace endpointu poskytující posbírané události tamtéž, v souboru *donate\_app/events/events-api/src/handler.ts*.

### FE aplikace

V rámci implementace FE aplikací zde bude reprezentativně rozebrána implementace formuláře pro generování platebních informací (Obrázek 9) jak v Angular verzi (*donate\_app/client*), tak v NestJS verzi (*donate\_app/server*).

Hlavním rozdílem mezi těmito verzemi – jak z hlediska uživatelského, tak implementačního – je, na základě jaké události je vygenerován nový platební QR kód. V případě server-rendered NestJS verze se tak děje po odeslání formuláře (HTTP požadavku). V případě Angular verze se platební kód v reálném čase generuje vyplňováním formuláře.

#### NestJS

V případě NestJS verze musí být veškerá data potřebná k vykreslení stránky být inicializována a vrácena do *view* uvnitř akce obsluhujícího *controlleru*. Veškerá logika potřebná k rozhodnutí, jaká data budou vrácena je definována tamtéž. Ve Výpis 7.2 je naznačeno, jak je z odeslaného formuláře generován platební QR kód. Ve výpisu je také vidět, že kromě dat potřebných pro správné zobrazení formuláře či platebního kódu je v jednom místě vracen třeba také seznam událostí ve farnosti.

Výpis . NestJS: Logika generování platebního kódu

…

const isDonation = query.type === 'donation';

const amount = query.amountRadio === 'custom' ? query.customAmount : query.amountRadio;

const paymentImg = amount

? await this.getPaymentImg(amount, query, isDonation, paymentConfig)

: false;

…

…

return {

query,

amountOptions,

events,

isDonation,

showCustomAmountInput: query.amountRadio === 'custom',

paymentImg,

paymentConfig,

};

View vrstva již jen vykresluje předpřipravená data od controlleru s použitím pouze jednoduchých strukturálních operátorů jako jsou podmínky, cykly, či vykreslení podšablony. Každá taková podšablona je obalena elementem *div* s třídou shodnou s názvem podšablony. Pro definici stylů je využit jazyk *scss* umožňující mimo jiné zanořování stylů. Kombinací těchto dvou přístupů je zajištěna vzájemná nekonfliktnost stejných tříd v rámci podšablon. Ukázku z definice stylů a formuláře pro platbu lze vidět ve Výpis 7.3.

Výpis . Ukázka view vrstvy formuláře pro platby

*/\* \_\_payment.scss \*/*

.payment {

flex-grow: 1;

display: block;

padding: 20px;

box-sizing: border-box;

height: calc(100vh - 60px);

overflow: auto;

…

.contact-form .submit-btn {

margin: 20px 0;

display: flex;

justify-content: center;

}

…

}

*/\* payment.hbs \*/*

<div class="payment">

…

<form action="/#donate" method="GET">

…

{{#if isDonation}}

<div class="contact-form">

<div class="form-control">

<input

name="contact\_form\_\_name"

id="contact-form\_\_name"

type="text"   
 autocomplete="name"

placeholder="&nbsp;"

value="{{query.contact\_form\_\_name}}"  
 >

<label for="contact-form\_\_name">Jméno a příjmení</label>

</div>

…

#### Angular

V Angularu se jednotlivé části stránky strukturují do komponent. Každou komponentu tvoří trojice souborů – typescript *\*.component.ts* obsahující „logiku“ komponenty, HTML *\*.component.html* obsahující šablonu komponenty a SCSS *\*.component.scss* obsahující definice stylů pro danou komponentu – které na rozdíl od NestJS verze není již potřeba nijak prefixovat názvem komponenty pro eliminaci konfliktů, Angular již toto zajišťuje out-of-the-box. Na rozdíl od NestJS je veškerá logika pro zobrazení platebního formuláře a generování platebního QR kódu uvnitř typescript souboru této komponenty – úplně oddělena např. od událostí. Ve Výpis 7.4 lze vidět, jak probíhá generování platebního kódu na základě změny hodnot ve formuláři bez posílání HTTP requestů.

Výpis . Angular: Logika generování platebního QR kódu

combineLatest([

this.amountFC.valueChanges.pipe(startWith(this.amountFC.value)),

this.userInfo.valueChanges.pipe(startWith(this.userInfo.value))

]).subscribe(([am, userInfo]) => {

if (am) {

this.showQR = true;

const payment = {

acc: this.config.accounts[this.type].account,

am,

cc: 'CZK',

msg: this.type === 'donation' ? `${this.getUserInfo()} ` : 'NEDELNI SBIRKA',

xvs: this.config.accounts[this.type].vs

};

qrcode

.toDataURL(spayd(payment))

.then((url) => {

this.imgSrc = url;

})

.catch(console.error);

} else {

this.showQR = false;

}

});

Veškeré *public properties* na typescript třídě komponety jsou dostupné v šabloně komponenty. Jak lze vidět ve Výpis 7.5, na rozdíl od NestJS a klasického MVC přístupu má Angular definováno vlastní API pro práci s formuláři – používající pro komunikaci s typescript částí speciální HTML atributy (direktivy) jako *[formGroup]*, či *[formControlName]*. Všimnout si lze také speciálního atributu *\*ngIf*. Atributy začínající na \* jsou tzv. *strukturální direktivy* a umožňují v šablonách podmínky, cykly atd.

Výpis . Angular: šablona platebního formuláře

/\* payment.component.scss \*/

:host {

display: block;

padding: 20px;

box-sizing: border-box;

}

/\* payment.component.html \*/

…

<div class="contact-form" \*ngIf="type !== 'sunday'" [formGroup]="userInfo">

<div class="form-control">

<input formControlName="name"

id="contact-form\_\_name"

type="text"

autocomplete="name"

placeholder="&nbsp;" />

<label for="contact-form\_\_name">Jméno a příjmení</label>

</div>

…

## E-shop

Kromě FE aplikací musel být v rámci této práce naimplementován mockovaný backend, se kterým aplikace komunikují a ze kterého tyto aplikace čerpají data.

Jednotlivé verze jsou nasazeny na následujících adresách:

* SSR: <https://dp-eshop.web.app/>
* Server-rendered: <https://dp-eshop-sr.web.app/>
* CSR: <https://dp-eshop-csr.web.app/>
* Prerender: <https://dp-eshop-prerender.web.app/>

### Backend

Backend pro FE aplikace je naimplementován také v NestJS. Naimplementováno bylo jednoduché REST API, které má následující endpointy:

* */cart* – práce s košíkem
* */categories* – informace o kategoriích produktů
* */homepage* – data pro domovskou stránku
* */products* – detaily produktů a jejich stránkované zobrazování
* */site-info* – kontaktní informace, otevírací doba, …
* */static-page* – data pro jednotlivé statické stránky

Kód pro jednotlivé endpointy sídlí v odpovídajících složkách uvnitř *eshop/backend/src v* Příloha C. Kromě controllerů (*\*.controller.ts*) a modelů (*\*.service.ts*) obsahuje každá složka také soubor (*\*.data.ts*), který obsahuje mocková data pro každý endpoint.

Protože bylo potřeba vygenerovat větší množství produktů, aby výsledné stránky vypadaly reálně a zároveň mohlo být implementováno stránkování, byl vytvořen skript na generování produktů. Tento skript lze nalézt v Příloha C – *eshop/backend/scripts/generate\_products.ts*.

### FE aplikace

Na příkladu stránky detailu kategorie bude ukázáno, jak se v Angularu a NestJS definují jednotlivé stránky, získávají a vykreslují data z REST API, a jak se pracuje s router parametry a SEO. Popis struktury komponent a šablon podrobněji rozebírá sekce 7.3.4, tato část se tím tedy nezaobírá.

#### Definice jednotlivých stránek

Dobrou zvyklostí při definování nového typu stránky je jak v Angularu, tak NestJS oddělit veškerý kód potřebný k této stránce do tzv. modulů. Tento přístup má výhodu hlavně v tom, že výsledná aplikace pro zvolenou cestu načítá pouze závislosti, které daná stránka opravdu potřebuje. V případě Angular verze e-shopu lze vidět, že pouze stránka detailu kategorie využívá knihovnu pro stránkování, proto je také v *CategoriesModule* naimportován *NgbPaginationModule* tuto knihovnu obsahující. Ostatní stránky jako domovská stránka již stránkování nepoužívají. Výsledný kód, který se musí pro jejich vykreslení načíst a spustit již knihovnu pro stránkování neobsahují, což vede k větší výkonnosti těchto stránek. V Angularu se nové stránky definují pomocí tzv. *routing modules*, které mohou být hierarchicky zanořeny a definují, jaké komponenty pro jaké cesty vykreslit viz Výpis 7.6.

Výpis . Angular: definice routes pro detail kategorie

/\* eshop/clients/angular/src/app/app-routing.module.ts \*/

const routes: Routes = [

…

{ path: 'cat', loadChildren: () => import('./categories/categories.module').then(m => m.CategoriesModule) },

…

];

/\* eshop/clients/angular/src/app/categories/categories-routing.module.ts \*/

const routes: Routes = [{ path: ':slug', component: CategoriesComponent }];

V případě NestJS se definice routes specifikuje příslušnými dekorátory na odpovídajících controllerech viz Výpis 7.7.

Výpis . NestJS: definice routes pro detail kategorie

@Controller('cat')

export class CategoryController {

…

@Get(':slug')

@Render('category/show')

async show(

…

#### Získávání a vykreslování dat z REST API

Práce s datovou vrstvu obsluhují v obou frameworcích tzv. *Services*. Pro detail kategorie jsou využity dvě services – *CategoryService* pro získání dat k požadované kategorii (SEO, název, obrázek, …) a *ProductsService* pro získávání produktů patřící k dané kategorii. Implementace je v obou verzích v podstatě stejná, pro práci s HTTP požadavky se používají jiné nativní services (v Angularu *HttpClient* pod pokličkou využívající *XmlHttpRequest* API, v NestJS pak *HttpService* využívající *Axios*). Ukázku z *ProductsService* metody na získání požadované stránky výpisu produktů kategorie lze vidět ve Výpis 7.8.

Výpis . Services: získání stránkovaných produktů kategorie

public getAll(params: { page: string; pageSize: string; category: string }) {

return this.http.get<Paginated<Product>>(`${environment.api}/products`, {

params: params,

});

}

Tyto services poté využívá k poslání dat do šablony na straně NestJS *CategoryController*, v případě Angularu *CategoriesComponent*.

#### Práce s router parametry

V případě detailu kategorie je router parametrem slug kategorie. Kromě toho stránka používá query parametry, ze kterých se získává aktuální stránka výpisu produktů kategorie a počet produktů na stránku. URL výpisu kategorie tedy vypadá přibližně takto: */cat/:slug?page=X&pageSize=Y*.

V práci s routami se Angular a NestJS dosti liší. V případě NestJS stačí označit funkci obsluhující akci controlleru a její parametry správnými dekorátory a tyto parametry již framework sám naplní správnými hodnotami z HTTP požadavku viz Výpis 7.9.

Výpis . NestJS: práce s route parametry

/\* eshop/clients/server-rendered/src/category/category.controller.ts \*/

@Get(':slug')

@Render('category/show')

async show(

@Param('slug') slug,

@Query('page') page = '1',

@Query('pageSize') pageSize = '12',

): Promise<SharedViewData> {

…

V případě Angularu je v komponentách dostupná ActivatedRoute service, která poskytuje dvě *RxJS Observables* – jednu pro parametry, druhou query parametry. Už z tohoto rozhraní vyplývá více funkcionální přístup k programování a práci s těmito daty. Ve Výpis 7.10 lze vidět, jak je naplněna proměnná obsahující výpis produktů pro danou stránku a kategorii a jejíž obsah reaktivně reaguje na změnu route.

Výpis . Angular: práce s route parametry

/\* eshop/clients/angular/src/app/categories/categories.component.ts \*/

this.products$ = combineLatest([

this.route.paramMap,

this.route.queryParamMap,

]).pipe(

map((params) => ({

page: params[1].get('page') ?? this.page.toString(),

pageSize: params[1].get('pageSize') ?? this.pageSize.toString(),

category: params[0].get('slug') ?? '',

})),

tap((params) => {

this.page = +params.page;

this.pageSize = +params.pageSize;

}),

switchMap((p) => this.productsService.getAll(p))

);

#### Práce se SEO

Práce se SEO je v Angularu a NestJS dosti odlišná. Aplikace Angularu má statický *index.html*, kde se Angular samotný napojuje až dovnitř tagu *body* – hlavička HTML dokumentu tedy leží úplně mimo dosah frameworku a k její úpravě je potřeba použít javascript. V Angularu byla využita knihovna *ngx-seo* obsahující services ulehčující tuto manipulaci s elementy v hlavičce. V případě nastavení SEO pro detail kategorie tak stačilo zavolat na její *SeoSocialShareService* metodu *setData* s požadovanými daty.

V případě NestJS je situace jednodušší – šablony s každým požadavkem vykreslují také hlavičky, takže lze konkrétní požadované *meta* tagy a jiné napřímo vypsat v šabloně. Část šablony vykreslující SEO lze vidět v *eshop/clients/server-rendered/src/views/shared/seo.ejs*

# Objektivní vyhodnocení

Na naimplementovaných aplikacích byla měřena výkonnost jednotlivých druhů renderingů. U e-shopu bylo také měřeno, jak dobře zvládají vybrané vyhledávače indexovat jednotlivé druhy renderingu. Všechna syntetická měření byla prováděna najednou, není tedy předpoklad vlivu rychlosti internetu, či *cold startů* lambda funkcí na naměřené hodnoty. Podrobněji je metodika těchto měření a vyhodnocování popsána v kapitole 6.

Aby šlo lépe porovnat, jak si jednotlivé renderingy vedou mezi sebou, ve všech tabulkách v této kapitole jsou jednotlivé buňky obarveny škálou barev v pořadí zelená, žlutá, oranžová a červená. Tyto barvy jsou přiřazeny podle pořadí daného renderingu v dané metrice od nejlepšího po nejhorší.

Prerendering vyhrál ve všech SI – důvod?

## Jednostránková aplikace

U jednostránkové aplikace bylo kromě syntetického měření také měřena výkonnost jednotlivých renderingů na zařízeních skutečných návštěvníků. V rámci syntetických měření nejlepších výsledků dosahovala server rendered verze. V případě měření reálných uživatelů si výrazně nejlépe vedla prerendering verze.

Syntetická měření pomocí Lighthouse dosáhly ve většině metrik lepších výsledků než měření těch samých metrik pomocí PageSpeed.cz. Hlavní rozdíl mezi druhy měření byl především v metrikách FCP a SI – jaký je za tím ale důvod se nepodařilo zjistit.

Důležitým zjištěním bylo, že při měření reálných uživatelů dosáhly SSR a server rendered verze výrazně horších výsledků než CSR a prerendering. Horší výkonnost právě těchto verzí je nejspíše způsobena *cold startem* Firebase funkcí. Tento jev se bohužel už z definice nemůže projevit u syntetických měření, čímž se jen potvrzuje důležitost měřit výkonnost webů na reálných uživatelích – mohou se projevit skryté vlivy na jejich výkonnost.

### Výkonnostní syntetické měření pomocí Lighthouse

Jak lze vidět v Tabulka 2, při syntetickém měření v desktopovém módu dosáhly všechny druhy renderingů velmi dobrých výsledků, které se od sebe signifikantně nelišily. Zajímavostí na měřeních je výrazně horší hodnota CLS pro prerendered a server rendered verzi, která bude nejspíše způsobena pomalejším načítáním vlastních Google fontů. To, že se tento jev nevyskytuje při CSR a SSR verzích bude nejspíše způsobeno tím, že se fonty stihnout stáhnout a načíst dříve, než stihne javascript vykreslit měřený text.

Při testování v režimu mobilních zařízení se již začaly více projevovat rozdíly v jednotlivých druzích renderingů. Jediný server rendering v tomto případě dosáhl perfektního skóre LPS, prerendering verzi uteklo o 1 bod. Vliv na toto pořadí měla především metrika LCP, která v případě CSR vychází jako nevyhovující.

Tabulka 2 Jednostránková aplikace: synteticky naměřené výkonnostní metriky pomocí Lighthouse

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 98 | 0,4 | 1 | 1 | 0,9 | 50 | 0,07 |
| prerender | 95 | 0,4 | 0,4 | 0,9 | 0,8 | 10 | 0,39 |
| server render | 96 | 0,4 | 0,7 | 0,8 | 0,4 | 0 | 0,42 |
| CSR | 99 | 0,7 | 0,7 | 0,9 | 0,8 | 20 | 0,05 |
| SSR mobile | 86 | 1,5 | 2,3 | 3,5 | 4,6 | 200 | 0,04 |
| prerender mobile | 89 | 1,5 | 1,5 | 3,4 | 3,4 | 170 | 0,05 |
| server render mobile | 93 | 1,5 | 1,8 | 3,2 | 1,5 | 0 | 0,05 |
| CSR mobile | 82 | 2,5 | 2,5 | 4,1 | 3,9 | 170 | 0,03 |

### Výkonnostní syntetické měření pomocí Pagespeed.cz

Poslední výkonnostní metriky naměřené nástrojem Pagespeed.cz lze vidět v Tabulka 3. Všechny verze renderingů jsou tímto nástrojem monitorovány déle než měsíc. V případě LPS se jak pořadí, tak hodnoty metrik jednotlivých renderingů dosti měnily – v extrémních případech až o 22 bodů pro stejnou metriku. Bohužel nelze příliš určit, co tyto odchylky způsobuje. Příčinou rozdílů může být jak hosting aplikací, tak nestabilita měřícího prostředí nástroje Pagespeed až po změny v obsahu stránky – přece jen události ve farnosti se mění v průběhu času. Tak či tak je dosti diskutabilní, jak moc lze brát v úvahu hodnoty naměřené tímto nástrojem.

Kompletní historii měření lze nalézt na <https://pagespeed.cz/r/dfd49e8aff7b>.

Tabulka 3 Jednostránková aplikace: synteticky naměřené výkonnostní metriky pomocí Pagespeed.cz

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 82 | 2,64 | 5,14 | 3,26 | 3,57 | 210 | 0,06 |
| prerender | 76 | 2,97 | 3,01 | 4,65 | 3,9 | 210 | 0,06 |
| server render | 85 | 2,64 | 4,71 | 3,26 | 2,64 | 0 | 0,06 |
| CSR | 80 | 2,64 | 3,36 | 4,24 | 3,49 | 200 | 0,06 |

### Výkonnostní měření reálných uživatelů

Měření výkonu na zařízeních návštěvníků webu a sběr dat probíhalo od 6. února do 18. dubna 2021. Během této doby bylo nasbíráno 1172 záznamů. Výsledné hodnoty uvedené v Tabulka 4 jsou vždy 7. decilem ze všech naměřených hodnot pro danou metriku a rendering.

Výkonnostní zpomalení aplikací používajících pro rendering serverovou stranu bylo popsáno již v úvodu této podkapitoly. To, že jsou horší výsledky dány právě *cold startem* bylo usouzeno především na základě rozdílných hodnot TTFB pro jednotlivé druhy renderingů.

Za povšimnutí stojí lepší výsledky prerenderingu oproti CSR v metrikách FCP a LCP – předgenerované HTML v tomto případě relativně statické stránky opravdu zrychluje vykreslení stránky uživatelům. Další zajímavostí jsou velmi nízké hodnoty FID – za ideální se považují hodnoty do 100ms, což s přehledem splňují všechny druhy renderingů.

Tabulka 4 Jednostránková aplikace: naměřené výkonnostní metriky reálných uživatelů

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rendering | FCP | LCP | FID | CLS | TTFB |
| SSR | 3.87 | 4.59 | 13.42 | 0.07 | 3.12 |
| prerender | 1.63 | 2.6 | 8.85 | 0.06 | 0.83 |
| server render | 3.49 | 4.04 | 6.88 | 0.05 | 2.99 |
| CSR | 2.68 | 2.88 | 2.12 | 0.03 | 0.95 |

## E-shop

### Výkonnostní syntetické měření pomocí Lighthouse

Prerendering vyhrává SI, celkově válcuje server\_rendering, na desktopu nejsou moc rozdíly, hlavní je to na mobilu. Server-rendering to nejvíc dává v TTI a TBT. CSR to nedává na LCP – logicky, musí proběhnout kus logiky, než se zobrazí obrázky, banner, text…

Tabulka 5 E-shop domovská stránka: synteticky naměřené výkonnostní metriky pomocí Lighthouse

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 98 | 0,6 | 0,7 | 1,1 | 0,8 | 10 | 0,08 |
| prerender | 98 | 0,5 | 0,5 | 1 | 0,7 | 10 | 0,08 |
| server render | 97 | 0,5 | 1,1 | 1 | 0,5 | 0 | 0,04 |
| CSR | 98 | 0,5 | 0,6 | 1,1 | 0,6 | 0 | 0,09 |
| SSR mobile | 73 | 1,7 | 1,9 | 4,5 | 4,5 | 480 | 0,11 |
| prerender mobile | 70 | 1,7 | 1,7 | 4,7 | 4,5 | 470 | 0,16 |
| server render mobile | 84 | 1,6 | 2,7 | 4,3 | 2,2 | 30 | 0,1 |
| CSR mobile | 74 | 1,7 | 1,8 | 4,9 | 3,9 | 390 | 0,16 |

Tabulka 6 E-shop detail produktu: synteticky naměřené výkonnostní metriky pomocí Lighthouse

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 99 | 0,5 | 0,6 | 0,8 | 0,6 | 0 | 0,06 |
| prerender | 99 | 0,5 | 0,5 | 0,7 | 0,7 | 10 | 0,06 |
| server render | 100 | 0,5 | 0,8 | 0,7 | 0,5 | 0 | 0,02 |
| CSR | 98 | 0,5 | 0,6 | 1,1 | 0,6 | 0 | 0,04 |
| SSR mobile | 84 | 1,7 | 2,1 | 2,6 | 3,7 | 430 | 0,39 |
| prerender mobile | 85 | 1,7 | 1,7 | 2,6 | 3,8 | 440 | 0,44 |
| server render mobile | 93 | 1,7 | 2,1 | 2,6 | 2,4 | 30 | 0,34 |
| CSR mobile | 77 | 1,6 | 1,9 | 3,9 | 4,1 | 380 | 0,42 |

Tabulka 7 E-shop statická stránka: synteticky naměřené výkonnostní metriky pomocí Lighthouse

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 99 | 0,5 | 0,6 | 0,8 | 0,6 | 10 | 0,05 |
| prerender | 99 | 0,5 | 0,5 | 0,7 | 0,6 | 0 | 0,05 |
| server render | 99 | 0,6 | 0,8 | 0,8 | 0,6 | 0 | 0 |
| CSR | 99 | 0,5 | 0,6 | 0,8 | 0,6 | 0 | 0,05 |
| SSR mobile | 93 | 1,7 | 1,9 | 2,6 | 3,4 | 300 | 0,06 |
| prerender mobile | 90 | 1,6 | 1,6 | 2,6 | 4,1 | 350 | 0,06 |
| server render mobile | 95 | 2,1 | 2,4 | 2,7 | 2,2 | 20 | 0 |
| CSR mobile | 82 | 1,7 | 1,8 | 3,9 | 3,3 | 370 | 0,06 |

Tabulka 8 E-shop detail kategorie: synteticky naměřené výkonnostní metriky pomocí Lighthouse

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 94 | 0,5 | 0,9 | 1 | 1,3 | 120 | 0,14 |
| prerender | 97 | 0,5 | 0,5 | 1,1 | 1,3 | 40 | 0,11 |
| server render | 96 | 0,6 | 1,6 | 0,9 | 0,6 | 0 | 0,07 |
| CSR | 97 | 0,5 | 0,7 | 1,1 | 0,6 | 0 | 0,11 |
| SSR mobile | 61 | 1,7 | 2,8 | 4,2 | 5,1 | 960 | 0,3 |
| prerender mobile | 58 | 1,7 | 2,4 | 4,3 | 5,1 | 970 | 0,11 |
| server render mobile | 86 | 1,7 | 2,5 | 3,7 | 2,2 | 30 | 0,24 |
| CSR mobile | 72 | 1,6 | 2 | 4,5 | 4,1 | 390 | 0,88 |

### Výkonnostní syntetické měření pomocí Pagespeed.cz

HP: <https://pagespeed.cz/r/faaec312600d>

Product detail: <https://pagespeed.cz/r/e2b5f760c2af>

Category detail: <https://pagespeed.cz/r/082926336764>

Static site detail: <https://pagespeed.cz/r/f7651e1fd2a3>

Tabulka 9 E-shop domovská stránka: synteticky naměřené výkonnostní metriky pomocí Pagespeed.cz

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 66 | 2 | 5 | 4,6 | 4,5 | 490 | 0,06 |
| prerender | 73 | 2 | 2,5 | 4,6 | 4,4 | 440 | 0,06 |
| server render | 74 | 1,6 | 7,1 | 4,3 | 1,9 | 30 | 0,1 |
| CSR | 73 | 1,6 | 3,9 | 4,5 | 3,8 | 400 | 0,16 |

Tabulka 10 E-shop detail produktu: synteticky naměřené výkonnostní metriky pomocí Pagespeed.cz

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 86 | 2 | 2,5 | 6 | 3 | 260 | 0,06 |
| prerender | 79 | 1,6 | 2,7 | 7,8 | 1,6 | 0 | 0,39 |
| server render | 82 | 1,6 | 3,7 | 6,7 | 3,3 | 390 | 0,33 |
| CSR | 70 | 2,2 | 3,2 | 5,6 | 3,8 | 220 | 0,39 |

Tabulka 11 E-shop statická stránka: synteticky naměřené výkonnostní metriky pomocí Pagespeed.cz

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 82 | 2 | 5,4 | 2,9 | 2,8 | 390 | 0,06 |
| prerender | 76 | 2,2 | 4,1 | 3,6 | 3,8 | 470 | 0,06 |
| server render | 88 | 2 | 6,4 | 2,5 | 2 | 20 | 0 |
| CSR | 82 | 1,6 | 3,9 | 3,8 | 2,9 | 290 | 0,11 |

Tabulka 12 E-shop detail kategorie: synteticky naměřené výkonnostní metriky pomocí Pagespeed.cz

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rendering | LPS | FCP | SI | LCP | TTI | TBT | CLS |
| SSR | 68 | 1,6 | 5,9 | 3,5 | 4,1 | 470 | 0,7 |
| prerender | 73 | 2,2 | 5,3 | 3,8 | 4,5 | 280 | 0,46 |
| server render | 76 | 2 | 8,6 | 3,8 | 2,1 | 30 | 0 |
| CSR | 66 | 1,6 | 6,1 | 4,4 | 3,9 | 320 | 1,18 |

### SEO – Pozice ve vyhledávačích pro dosud neexistující produkty

Tabulka 13 Seznam.cz pozice jednotlivých renderingů ve výsledcích vyhledávání

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rendering | 1. | 2. | 3. | 4. |
| SSR | **2** | 0 | **3** | 0 |
| prerender | **1** | 0 | **2** | **2** |
| server render | **2** | 0 | 0 | **3** |
| CSR | 0 | **5** | 0 | 0 |

### SEO – Výsledek vyhledávání

### SEO – Google product snippet

### SEO – Facebook snippet

# Subjektivní vyhodnocení

Cílem této práce je kromě objektivního vyhodnocení jednotlivých druhů renderingů pomocí definovaných metrik a měření také subjektivně srovnat přívětivost a výhody implementace jednotlivých verzí renderingu. V rámci subjektivního vyhodnocení jsou také porovnány jednotlivé implementace z uživatelského hlediska.

## Implementační přívětivost

I když se přístupy v programování určitých částí v různých druzích renderingu dosti liší, výslednou aplikaci lze udělat perfektně fungující a výkonnou jak v Angularu, tak NestJS. V každém z těchto frameworků jsou ale různé části různě složité, a to je to hlavní, v čem se liší.

### Srovnání architektur a rolí programátorů

Angular – rozdělení na BE a FE => možnost specializace na db na BE, UX a výkonnost na FE => nezávislost týmů.

CSR – primitivní na nasazení, pokud API již je a není potřeba SEO, není nic jednoduššího

SSR, prerendering – opravuje věci, které CSR neumí a je to znát – správně rozchodit v existující CSR aplikaci dost problém

SSR – nutnost udržovat server i pro FE – potřeba tedy minimálně BE API + FE SR + CDN

Prerendering – zdlouhavé nasazování (klidně i půl hodiny), nevhodné pro aplikace s velkým počtem stránek (>5000) – každá se musí předrenderovat, a to nejen se změnou aplikace, ale i změnou dat!! Nevhodné tedy pro všechny případy

Server rendering – programátor musí být odborník na dost oblastí (db, programování na straně serveru, JS, HTML, CSS, testování, správa serveru…), v případě špatného používání může být mnohem větší problém škálovat než BE + FE

### Svůdnost k bad practices

Angular a jeho alternativy dost často svádějí k tomu, že navigace mezi stránkami není řešená odkazy, ale pouze reakce na akci *click* na navigační tlačítko. Celkově s navigací mají tato řešení obvykle problém – nepoužívají GET parametry, tam kde by byly vhodné, ale stav aplikace drží uvnitř, což znemožňuje poslat někomu odkaz. Dosti často také bývá celková implementace práce s routes špatně, což má za následek třeba nefunkční tlačítko zpět. Velice často také není čas či chuť ošetřit všechny stavy načítání dat, což vede k tomu, že se aplikace v případě nefunkčního API začne chovat divně, ale na rozdíl od serverově renderované aplikace nespadne, což vede k mnohem většímu zmatení uživatelů. Dalším častým jevem je absolutní absence jakékoliv implementace i základů SEO jako je titulek stránky, či náhledový obrázek – obvykle bývá pouze jeden globální pro celou aplikaci.

Serverově renderované aplikace mají ze zkušenosti autora problémy s organizací javascriptů potřebných k interaktivitě stránky. Obvykle existuje v horším případě jeden, v lepším případě několik javascriptových souborů s několika stovkami až tisíci řádky špagetového jQuery kódu, ve kterém se nikdo nevyzná. Další obvykle používanou bad practice je používání v lepším případě metody POST, v horším dokonce GET pro vytváření, mazání, či upravování objektů. Tento přístup v lepším případě znesnadňuje následné vyhledávání v logs aplikace na produkci při opravování chyb, v horším nechtěné mazání či přepisování dat refreshováním stránky.

## Uživatelská přívětivost

Z pohledu uživatele výsledných webových aplikací lze subjektivní pohled autora práce na jednotlivé druhy renderingů zjednodušeně shrnout následovně.

Pro aplikace typu farní sbírky implementované v rámci této práce je jednoznačným vítězem Angular. Okamžité překreslování platebního kódu během vyplňování formuláře působí skvěle a člověk má okamžitou odezvu, že vše funguje, jak má. Pro takto malé projekty je také skvělé, že lze využit existujících hostingů statických souborů a CDN, které fungují spolehlivě – na rozdíl od aplikací, které vyžadují server, kde se může ledacos pokazit (jak se také autor v rámci implementace serverových částí farních sbírek přesvědčil).

Pro aplikace typu e-shop, či blogy již volba není tak jednoznačná. I ze zkušenosti s různými e-shopy fungujícími na internetu většinou zůstává pocit, že klientsky renderované aplikace fungují v průměru hůř. Podobný pocit je také z e-shopu implementovaného v rámci této práce. Hlavním problémem jsou v tomto případě různé části stránky, které při stahování potřebných dat různě problikávají a poskakují. Tomuto chování se sice dá v klientsky renderovaných aplikacích zabránit, ale implementace tohoto chování již není tak přímočará a odladit ji může zabrat hodně času – na což v praxi dost často nezbudou peníze. Nehledě na strojovou čitelnost webu, která je v případě klientsky renderovaných aplikacích z důvodu náročné implementace velmi často absolutně zanedbána. V případě, že je ale na implementaci takové aplikace kvalitní tým s dostatkem času a peněz, je autor práce přesvědčen, že Angular a jemu podobné frameworky umí doručit lepší uživatelský zážitek z aplikace než serverově renderované aplikace. Pro většinu aplikací tohoto typu je ale většinou lepší použít PHP, Ruby on Rails, Django, NodeJS či jiné serverově renderované technologie – a ušetřený čas a peníze investovat do jiných částí businessu než do ladění uživatelských detailů klientsky renderované aplikace.

# TODOs

Zkontrolovat odkazy na zdroje v poznámkách pod čarou

Předělat sbírku na wires

Odstranit košík ze specky eshopu

Seznamy obrázků, popsat zkratky, …

# Otázky ke konzultaci

* NextJS by nyní byl nejspíš lepší alternativou – umožňuje různé renderingy pro různé stránky, přišel před měsícem s novým druhem renderingu – ISR, který ani nemám v teorii, protože prostě neexistoval. Co s tím? To samé react a jeho nové server components. To samé [www.hotwire.dev](http://www.hotwire.dev). –– dopsat na konci v závěru
* Z dat ohledně metrik mám pocit, že firebase functions pěkně podělávaj TTFB a tím i všechny ostatní metriky. Mám chuť naimplementovat e-shop mimo lambdy => pak ale projekt sbírek a projekt e-shopu nebude vzájemně porovnatelný co se týče metrik? Možná nechám tak, ale medián na TTFB dělá okolo 5s. Nejspíš to způsobuje pomalý cold start. Ve firmě se nám moc nedělo – asi navštěvovanější web. –– Dopsat ve vyhodnocení

5.4.:

- jak strukturovat výsledky měření – tabulka?

- testování na windows – na té nasazené verzi? Asi jo…

- google snippet – nejspíš bude jen že nefunguje/funguje

Závěr

V závěru shrnu: Cílem bylo to a to, s tím a tím předpokladem, v tý a tý kapitole jsem provedl to a to a cíl práce byl splněn…

Future work, kdo může využít atd…

Cíl nemusí být jedna věta, ale třeba jeden hlavní cíl s několika dílčími cíli – konkrétní věci co chci měřit. Dílčí cíle se většinou číslují s dc1, dc2 atd – ať to nejsou jen odrážky. Tu v závěru se na ně pak takto odkážu.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo. Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Phasellus et lorem id felis nonummy placerat.

Použitá literatura

ANGULAR/UNIVERSAL, 2021. *angular/universal*. TypeScript. B.m.: Angular [vid. 31. březen 2021]. Získáno z: https://github.com/angular/universal

BASECAMP, LLC, 2020. *turbolinks/turbolinks*. TypeScript. B.m.: Turbolinks [vid. 6. říjen 2020]. Získáno z: https://github.com/turbolinks/turbolinks

BOBBINK, Jan-Willem, 2019. Rendering on the Web - The SEO Version. *NotProvided.eu*. [vid. 19. březen 2021]. Získáno z: https://www.notprovided.eu/rendering-on-the-web-the-seo-version/

BOMBACH, Lukas, 2019. How we achieved the best Web Performance with Partial Hydration. *Medium* [vid. 6. říjen 2020]. Získáno z: https://medium.com/@luke\_schmuke/how-we-achieved-the-best-web-performance-with-partial-hydration-20fab9c808d5

CROSS, Jeff, 2016. feat(DomRenderer): allow partial DOM hydration from pre-rendered content · Issue #13446 · angular/angular. *GitHub* [vid. 20. září 2020]. Získáno z: https://github.com/angular/angular/issues/13446

DHH, 2016. The Majestic Monolith. *Signal v. Noise* [vid. 20. září 2020]. Získáno z: https://m.signalvnoise.com/the-majestic-monolith/

DSHAPS, 2016. Understanding Component-Based Architecture. *Medium* [vid. 20. září 2020]. Získáno z: https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238

FACEBOOK INC., 2020. *Docusaurus · Easy to Maintain Open Source Documentation Websites* [vid. 6. říjen 2020]. Získáno z: https://docusaurus.io/

FACEBOOK INC., 2021. Facebook Crawler - Sharing - Documentation. *Facebook for Developers* [vid. 20. březen 2021]. Získáno z: https://developers.facebook.com/docs/sharing/webmasters/crawler/

GECHEV, Minko, 2020. feat(DomRenderer): allow partial DOM hydration from pre-rendered content · Issue #13446 · angular/angular. *GitHub* [vid. 20. září 2020]. Získáno z: https://github.com/angular/angular/issues/13446

GIANNAKOULOPOULOS, Andreas, Nikos KONSTANTINOU, Dimitris KOUTSOMPOLIS, Minas PERGANTIS a Iraklis VARLAMIS, 2019. Academic Excellence, Website Quality, SEO Performance: Is there a Correlation? *Future Internet*. 18.11., roč. 11, č. 11, s. 242. ISSN 1999-5903. DOI: 10.3390/fi11110242

GOOGLE LLC, 2019a. First Contentful Paint. *web.dev* [vid. 26. únor 2021]. Získáno z: https://web.dev/first-contentful-paint/

GOOGLE LLC, 2019. Lighthouse Variability | Tools for Web Developers. *Google Developers* [vid. 5. duben 2021]. Získáno z: https://developers.google.com/web/tools/lighthouse/variability

GOOGLE LLC, 2019b. Time to Interactive. *web.dev* [vid. 13. březen 2021]. Získáno z: https://web.dev/interactive/

GOOGLE LLC, 2019c. Total Blocking Time. *web.dev* [vid. 13. březen 2021]. Získáno z: https://web.dev/lighthouse-total-blocking-time/

GOOGLE LLC, 2020a. Implement dynamic rendering | Search for Developers. *Google Developers* [vid. 21. říjen 2020]. Získáno z: https://developers.google.com/search/docs/guides/dynamic-rendering

GOOGLE LLC, 2020b. Largest Contentful Paint (LCP). *web.dev* [vid. 10. březen 2021]. Získáno z: https://web.dev/lcp/

GOOGLE LLC, 2020c. *Maskování - Nápověda Search Console* [vid. 21. říjen 2020]. Získáno z: https://support.google.com/webmasters/answer/66355

GOOGLE LLC, 2020d. The Science Behind Web Vitals. *Chromium Blog*. [vid. 2. listopad 2020]. Získáno z: https://blog.chromium.org/2020/05/the-science-behind-web-vitals.html

GOOGLE LLC, 2020e. *Web Vitals* [vid. 1. srpen 2020]. Získáno z: https://web.dev/vitals/

GOOGLE LLC, 2021a. Explore the Search Gallery & Rich Results | Search Central. *Google Developers* [vid. 19. březen 2021]. Získáno z: https://developers.google.com/search/docs/guides/search-gallery

GOOGLE LLC, 2021b. *GoogleChrome/web-vitals*. JavaScript. B.m.: GoogleChrome [vid. 14. březen 2021]. Získáno z: https://github.com/GoogleChrome/web-vitals

GOOGLE LLC, 2021c. Implement dynamic rendering | Search Central. *Google Developers* [vid. 19. březen 2021]. Získáno z: https://developers.google.com/search/docs/guides/dynamic-rendering

GOOGLE LLC, 2021d. SEO Starter Guide: The Basics | Google Search Central. *Google Developers* [vid. 17. březen 2021]. Získáno z: https://developers.google.com/search/docs/beginner/seo-starter-guide

GOOGLE LLC, 2021e. *Test použitelnosti v mobilech – Google Search Console* [vid. 19. březen 2021]. Získáno z: https://search.google.com/test/mobile-friendly

GOOGLE LLC, 2021f. Understand the JavaScript SEO basics | Search Central. *Google Developers* [vid. 19. březen 2021]. Získáno z: https://developers.google.com/search/docs/guides/javascript-seo-basics

GOOGLECHROME, 2020. *GoogleChrome/rendertron*. TypeScript. B.m.: GoogleChrome [vid. 21. říjen 2020]. Získáno z: https://github.com/GoogleChrome/rendertron

GRAPHQL FOUNDATION, 2020. *GraphQL: A query language for APIs.* [vid. 20. září 2020]. Získáno z: http://graphql.org/

JAHODA, Bohumil, 2016. *Indexování JavaScriptu* [vid. 20. březen 2021]. Získáno z: https://jecas.cz/seo-javascript

KOMUNITNÍ CENTRUM MATKY TEREZY, 2020. *Akce* [vid. 20. únor 2021]. Získáno z: https://www.kcmt.cz/kalendar-akci/

LAVALL, Anthony, 2020. JavaScript rendering and the problems for SEO in 2020. *Search Engine Watch* [vid. 21. říjen 2020]. Získáno z: https://www.searchenginewatch.com/2020/05/06/javascript-rendering-and-the-problems-for-seo-in-2020/

LEDFORD, Jerri L, 2008. *SEO Search Engine Optimization Bible* [vid. 19. březen 2021]. ISBN 978-0-470-26211-5. Získáno z: https://nbn-resolving.org/urn:nbn:de:101:1-201502085974

MALÝ, Martin, 2009. Web Sockets. *Zdroják*. [vid. 20. září 2020]. Získáno z: https://www.zdrojak.cz/clanky/web-sockets/

MARKBÅGE, Sebastian, 2019. Partial Hydration by sebmarkbage · Pull Request #14717 · facebook/react. *GitHub* [vid. 20. září 2020]. Získáno z: https://github.com/facebook/react/pull/14717

MICROSOFT CORPORATION, 2021. *Webmaster Guidelines - Bing Webmaster Tools* [vid. 17. březen 2021]. Získáno z: https://www.bing.com/webmasters/help/webmasters-guidelines-30fba23a

MICHÁLEK, Martin, 2018. Prohlížeče v Česku: Webdesignérův průvodce pro rok 2018. *Vzhůru dolů* [vid. 2. listopad 2020]. Získáno z: https://www.vzhurudolu.cz/prirucka/prohlizece

MICHÁLEK, Martin, 2019a. Metrika „Čas do interaktivity“ (Time To Interactive, TTI). *Vzhůru dolů* [vid. 3. srpen 2020]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-tti

MICHÁLEK, Martin, 2019b. Metrika „Index rychlosti“ (Speed Index, SI). *Vzhůru dolů* [vid. 13. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/speedindex

MICHÁLEK, Martin, 2019c. Metrika „První nečinnost procesoru“ (First Input Delay, FID). *Vzhůru dolů* [vid. 13. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-fid

MICHÁLEK, Martin, 2019d. Metrika „První smysluplné vykreslení“ (First Meaningful Paint, FMP). *Vzhůru dolů* [vid. 3. srpen 2020]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-fmp

MICHÁLEK, Martin, 2019e. Me­t­ri­ky rych­los­ti webů: Prů­vod­ce s de­tail­ním vy­svět­le­ním. *Vzhůru dolů* [vid. 1. srpen 2020]. Získáno z: https://www.vzhurudolu.cz/prirucka/metriky-rychlosti

MICHÁLEK, Martin, 2019f. Událost DOM Content Loaded (DCL). *Vzhůru dolů* [vid. 5. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/udalost-dcl

MICHÁLEK, Martin, 2019g. Událost Load. *Vzhůru dolů* [vid. 13. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/load

MICHÁLEK, Martin, 2019h. Událost „První vykreslení“ (First Paint, FP). *Vzhůru dolů* [vid. 3. srpen 2020]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-fp

MICHÁLEK, Martin, 2020a. Lighthouse Performance Score (LPS): krásy i pasti metriky všech metrik. *Vzhůru dolů* [vid. 13. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-lps

MICHÁLEK, Martin, 2020b. Metrika „Kumulativní posun layoutu“ (Cumulative Layout Shift, CLS). *Vzhůru dolů* [vid. 13. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-cls

MICHÁLEK, Martin, 2020c. Metrika „Největší vykreslení obsahu“ (Largest Contentful Paint, LCP): Kdy se vykreslí hlavní obsah stránky? *Vzhůru dolů* [vid. 10. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-lcp

MICHÁLEK, Martin, 2020d. Metrika Total Blocking Time (TBT), zaměřeno na pomalý JavaScript. *Vzhůru dolů* [vid. 13. březen 2021]. Získáno z: https://www.vzhurudolu.cz/prirucka/metrika-tbt

MILLER, Jason a Addy OSMANI, 2019. Rendering on the Web. *Google Developers* [vid. 18. srpen 2020]. Získáno z: https://developers.google.com/web/updates/2019/02/rendering-on-the-web

MOZILLA CONTRIBUTORS, 2019. Time to first byte. *MDN Web Docs* [vid. 1. srpen 2020]. Získáno z: https://developer.mozilla.org/en-US/docs/Glossary/time\_to\_first\_byte

MOZILLA CONTRIBUTORS, 2020a. *First contentful paint - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [vid. 26. únor 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Glossary/First\_contentful\_paint

MOZILLA CONTRIBUTORS, 2020b. *First Meaningful Paint - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [vid. 5. březen 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Glossary/first\_meaningful\_paint

MOZILLA CONTRIBUTORS, 2020c. *Window: DOMContentLoaded event - Web APIs | MDN* [vid. 5. březen 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded\_event

MOZILLA CONTRIBUTORS, 2020d. *Window: load event - Web APIs | MDN* [vid. 13. březen 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Web/API/Window/load\_event

MOZILLA CONTRIBUTORS, 2021a. *First input delay - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [vid. 10. březen 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Glossary/First\_input\_delay

MOZILLA CONTRIBUTORS, 2021b. *First paint - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [vid. 26. únor 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Glossary/First\_paint

MOZILLA CONTRIBUTORS, 2021c. *Speed index - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [vid. 13. březen 2021]. Získáno z: https://developer.mozilla.org/en-US/docs/Glossary/Speed\_index

NAVRCHOLU.CZ, 2021. NAVRCHOLU.cz: Žebříček Náboženství a církve (9. 2. 2021). *NAVRCHOLU.cz* [vid. 20. únor 2021]. Získáno z: https://navrcholu.cz/Zebricek/Instituce-urady/Nabozenstvi-cirkve/2021/02/09/

NEXT.JS, 2021. *Basic Features: Pages | Next.js* [vid. 31. březen 2021]. Získáno z: https://nextjs.org/docs/basic-features/pages

PAGESPEED.CZ, 2021. PageSpeed.cz - Na rychlosti záleží. *PageSpeed.cz* [vid. 5. duben 2021]. Získáno z: https://pagespeed.cz/

PICHLÍK, Roman, 2017. A REST. *Medium* [vid. 20. září 2020]. Získáno z: https://dagblog.cz/a-rest-c5156313d79e

PINTEREST ENGINEERING, 2015. *Demystifying SEO with experiments | by Pinterest Engineering | Pinterest Engineering Blog | Medium* [vid. 19. březen 2021]. Získáno z: https://medium.com/pinterest-engineering/demystifying-seo-with-experiments-a183b325cf4c

PRERENDER, 2020. *prerender/prerender*. JavaScript. B.m.: Prerender [vid. 21. říjen 2020]. Získáno z: https://github.com/prerender/prerender

SEZNAM.CZ, A.S., 2021. *Optimalizace webu | Seznam Nápověda* [vid. 17. březen 2021]. Získáno z: https://napoveda.seznam.cz/cz/fulltext-hledani-v-internetu/optimalizace-webu/

SITEMAPS.ORG, 2016. *sitemaps.org - Protocol* [vid. 19. březen 2021]. Získáno z: https://www.sitemaps.org/protocol.html

STATCOUNTER, 2021. Search Engine Market Share Czech Republic. *StatCounter Global Stats* [vid. 20. březen 2021]. Získáno z: https://gs.statcounter.com/search-engine-market-share/all/czech-republic

SUBRAMANIAN‎, Sowmya, 2020. Evaluating page experience for a better web. *Official Google Webmaster Central Blog*. [vid. 18. srpen 2020]. Získáno z: https://webmasters.googleblog.com/2020/05/evaluating-page-experience.html

THE OPEN UNIVERSITY, 2020. An introduction to web applications architecture. *OpenLearn* [vid. 20. září 2020]. Získáno z: https://www.open.edu/openlearn/science-maths-technology/introduction-web-applications-architecture/content-section-0

TWITTER, INC., 2021. *Getting started with Cards* [vid. 20. březen 2021]. Získáno z: https://developer.twitter.com/en/docs/twitter-for-websites/cards/guides/getting-started

VEGLIS a GIOMELAKIS, 2019. Search Engine Optimization. *Future Internet*. 31.12., roč. 12, č. 1, s. 6. ISSN 1999-5903. DOI: 10.3390/fi12010006

VUEJS, 2021. *Vue.js Server-Side Rendering Guide | Vue SSR Guide* [vid. 31. březen 2021]. Získáno z: https://ssr.vuejs.org/

W3TECHS.COM, 2020. *Usage Statistics and Market Share of Server-side Programming Languages for Websites, October 2020* [vid. 17. říjen 2020]. Získáno z: https://w3techs.com/technologies/overview/programming\_language

WAGNER, Jeremy, Chris ANSTEY a Bojan PAVIC, 2020. Why does speed matter? *web.dev* [vid. 31. březen 2021]. Získáno z: https://web.dev/why-speed-matters/

WALLMINE, 2020. *wallmine – Finance & Investing: Stocks, Crypto, Forex, ETFs* [vid. 6. říjen 2020]. Získáno z: https://wallmine.com/

WALTON, Philip a Milica MIHAJLIJA, 2020. Cumulative Layout Shift (CLS). *web.dev* [vid. 13. březen 2021]. Získáno z: https://web.dev/cls/

WEB.DEV, 2020. How focusing on web performance improved Tokopedia’s click-through rate by 35%. *web.dev* [vid. 31. březen 2021]. Získáno z: https://web.dev/tokopedia/

Přílohy

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo. Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Phasellus et lorem id felis nonummy placerat.

1. Wireframe aplikace farní sbírky
2. Wireframe e-shop zahradnictví
3. Zdrojové kódy aplikací

1. Blink, Gecko, Webkit; více v (Michálek, 2018) [↑](#footnote-ref-2)
2. Více o specificích monolitických webových aplikací v (DHH, 2016) [↑](#footnote-ref-3)
3. <https://insis.vse.cz/> [↑](#footnote-ref-4)
4. Více do detailu rozebraná specifika client-server architektury lze nalézt např. v (The Open University, 2020) [↑](#footnote-ref-5)
5. Více o REST konceptu v (Pichlík, 2017) [↑](#footnote-ref-6)
6. Více o GraphQL v (GraphQL Foundation, 2020) [↑](#footnote-ref-7)
7. Více o Websocketech např. v (Malý, 2009) [↑](#footnote-ref-8)
8. <https://reactjs.org/> [↑](#footnote-ref-9)
9. <https://angular.io/> [↑](#footnote-ref-10)
10. <https://vuejs.org/> [↑](#footnote-ref-11)
11. <https://mail.google.com/> [↑](#footnote-ref-12)
12. <https://nodejs.org/en/> [↑](#footnote-ref-13)