Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Identifikace nejvhodnějšího použití pro frameworky Gatsby a Next.js

Bekzot Adylov

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Bekzot Adylov

Informatika

Název práce

Identifikace nejvhodnějšího použití pro frameworky Gatsby a Next.js

Název anglicky

Identifying the Best Use Cases for Gatsby and Next.js Frameworks

---

Cíle práce

Hlavní cíl práce je identifikovat nejvhodnější použití pro frameworky Gatsby a Next.js.
Dílčí cíle práce jsou:
- vytvořit ukázkovou aplikaci s oběma frameworky
- na základě zvolených kritérií zhodnotit výkon, funkčnost a další faktory aplikací

Metodika

Teoretická část práce bude založena na základě studia odborných a

internetových zdrojů.

Praktická část práce bude založena na vytvoření a testování vzorových aplikací s využitím obou frameworků.

Na základě zvolených kritérií bude provedeno vyhodnocení aplikací.

Na základě teoretických a praktických znalostí budou formulovány závěry práce.

Doporučený rozsah práce

40

Klíčová slova

Gatsby, Next.js, JavaScript, výkon, framework, aplikace

Doporučené zdroje informací

CAMDEN, R., RINALDI, B., MATHIAS, Biilmann C. The Jamstack book : beyond static sites with JavaScript, APIs, and Markup. Shelter Island: Manning Publications, 2022. ISBN 9781617298882.

HAVERBEKE, M. Eloquent JavaScript : a modern introduction to programming. 3. vyd. San Francisco: No Starch Press, 2018. ISBN 9781593279509

RIVA, M. Real-World Next.js. Birmingham: Packt Publishing, 2022. ISBN 9781801073493

SO, P. Gatsby: The Definitive Guide. Cambridge: O'Reilly, 2021. ISBN 9781492087519

WAGNER, J. Web Performance in Action. Shelter Island: Manning Publications, 2016. ISBN 9781617293771

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

RNDr. Alexander Galba

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 4. 7. 2023

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 13. 03. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Identifikace nejvhodnějšího použití pro frameworky Gatsby a Next.js" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2024 _____

Poděkování

Rád bych poděkoval svému vedoucímu RNDr. Alexandru Galbovi za jeho cenné konzultace během psaní této práce.

Identifikace nejvhodnějšího použití pro frameworky Gatsby a Next.js

Abstrakt

Bakalářská práce zkoumá výkonnostní charakteristiky a nejvhodnější případy použití dvou populárních frameworků založených na Reactu, Next.js a Gatsby, v kontextu moderního vývoje webových aplikací. Výzkum zahrnuje teoretické i praktické zkoumání , jehož cílem je poskytnout komplexní pohled na možnosti a vhodnost těchto frameworků.

Prostřednictvím řady srovnávacích testů práce hodnotí faktory, jako je rychlost tvorby balíčků aplikací, velikost balíčků aplikací, optimalizace obrázků, doba spuštění a ekosystém pluginů.

Výsledky ukazují rozdíly mezi oběma frameworky v jejich metodách vykreslování a osvětlují nejlepší případy jejich využití při vývoji webových stránek v reálném světě.

Klíčová slova: Gatsby, Next.js, JavaScript, výkon, framework, aplikace

Identifying the Best Use Cases for Gatsby and Next.js Frameworks

Abstract

This bachelor thesis examines the performance characteristics and best use cases of two popular React-based frameworks, Next.js and Gatsby, in the context of modern web application development. The research includes both theoretical and practical assessments, aiming to provide a comprehensive view of the capabilities and suitability of these frameworks.

Through a series of benchmarks, the thesis evaluates factors such as application package creation speed, application package size, image optimization, execution time, and plugin ecosystem.

The results show the differences between the two frameworks in their rendering methods and on their best use cases for real-world web development.

Keywords: Gatsby, Next.js, JavaScript, performance, framework, application

Obsah

1 Úvod

Výběr vhodného frameworku má velký význam pro funkčnost, efektivitu a celkový výkon moderních webových aplikací. Vzhledem k tomu, že se vývojáři musí orientovat ve velkém množství různých frameworků, je důležité vybrat nejvhodnější software pro konkrétní případ použití. Mezi dostupnými možnostmi se v oblasti webových frameworků založených na React objevily Next.js a Gatsby jako hlavní konkurenti.

Next.js společnosti Vercel získal uznání v komunitě vývojářů díky své schopnosti efektivních metod načítání dat a dobře strukturovanému systému routingu, díky němuž se stal frameworkem pro komplexní a rozšiřitelné aplikace. Naopak Gatsby se dříve zaměřoval především na statické generování stránek (SSG) a poskytoval vysokorychlostní výkon poháněný integrací s GraphQL. Zavedení vykreslování na straně serveru (SSR) s verzí Gatsby 4 však srovnalo podmínky, díky čemuž jsou oba frameworky srovnatelnější než kdy dříve a mají nyní společné metody vykreslování.

Tato práce je věnována zkoumání a porovnání obou frameworků, přičemž hlavním cílem je určit jejich nejvhodnější případy použití. Zaměřuje se zejména na jejich možnosti vykreslování na straně serveru (SSR) a generování statických stránek (SSG). Prostřednictvím tohoto komplexního zkoumání si tato práce klade také za úkol přispět k rozšíření znalostí o vývoji webových aplikací a plně využít potenciál frameworků Next.js a Gatsby.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavní cíl práce je iden fikovat nejvhodnější použi pro frameworky
Gatsby a Next.js.
Dílčí cíle práce jsou:

- vytvořit ukázkovou aplikaci s oběma frameworky;
- na základě zvolených kritérií zhodno t výkon, funkčnost a další faktory aplikací.

2.2 Metodika

Teoretická část práce bude založena na studiu odborné literatury a internetových zdrojů. Bude zahrnovat důkladnou analýzu existujících výzkumů, teorií a postupů souvisejících s porovnáváním frameworků pro vývoj webových aplikací se zaměřením na Next.js a Gatsby.

Praktická část práce se bude soustředit na tvorbu a testování ukázkových aplikací s využitím frameworků Next.js a Gatsby. Tento praktický přístup je navržen tak, aby uvedl teoretické koncepty do reálného prostředí a umožnil komplexní pochopení funkčnosti a výkonnostních možností obou frameworků.

Na základě předem stanovených kritérií bude provedeno hodnocení těchto aplikací. Toto hodnocení poskytne konkrétní výsledky pro analýzu toho, jak jednotlivé frameworky fungují v podobných situacích.

Konečné závěry tohoto výzkumu budou formulovány na základě analýzy získaných teoretických a praktických poznatků. Tím se vytvoří celkový pohled na nejefektivnější využití jednotlivých frameworků, což usnadní jasné pochopení toho, v jakých oblastech jsou jednotlivé frameworky efektivnější. To potenciálně povede vývojáře k informované volbě, budou-li muset zvolit mezi frameworky Next.js a Gatsby.

3 Teoretická východiska

3.1 Webové aplikace

Podle organizace StackPath (2022) webová aplikace je počítačový program, který využívá různé webové prohlížeče a webové technologie k provádění úkolů prostřednictvím internetu. Podnikům a uživatelům poskytuje nákladově efektivní a účinný způsob výměny informací.

Olsson a Ockelberg (2020) uvádějí, že na rozdíl od tradičních aplikací jsou webové aplikace vyvinuty jednou a uživatelé k nim mohou přistupovat prostřednictvím webových prohlížečů, takže není nutné vytvářet samostatné verze pro různé operační systémy. Díky této univerzální přístupnosti je používání webových aplikací snadné, protože k jejich používání stačí pouze webový prohlížeč.

Jak uvádí Xu (2021), vývoj webových aplikací se opírá především o webové technologie, jako jsou HTML (Hypertext Markup Language) a JavaScript. HTML5, nejnovější verze jazyka HTML, sehrála významnou roli při utváření moderních webových rozhraní a interakcí. Přispěla k pokroku ve vývoji front-endu a vývoji webových prohlížečů.

Podle organizace StackPath (2022) webové aplikace se skládají ze dvou hlavních součástí: na straně klienta a na straně serveru. Klientská část se spouští ve webovém prohlížeči uživatele a interpretuje a zobrazuje obsah HTML přijatý ze serveru. Na straně serveru, který je spuštěn na webovém serveru, se provádějí různé procesy pro generování dokumentu HTML a jeho doručení klientovi.

Jak uvádí Kaluža et al. (2019) při vývoji webové aplikace je zásadní zvolit správnou technologii. Frameworky se staly cennými nástroji, které zvyšují kvalitu a standardizaci vývoje webových aplikací. Vzhledem k velkému množství dostupných frameworků je nezbytné je analyzovat a porovnávat na základě různých kritérií a určit jejich vhodnost pro konkrétní potřeby vývoje webových aplikací.

3.2 Technologie na straně klienta

3.2.1 Co jsou technologie na straně klienta

Shenoy a Prabhu (2018) uvádějí, že technologie na straně klienta hrají při vývoji webových aplikací zásadní roli, protože umožňují interaktivitu, dynamický obsah a uživatelsky prostředí. V této části se zaměříme na základní technologie na straně klienta a koncepty, které z nich vyplývají.

3.2.2 HTML

Podle MDN Web Docs (2018) HTML, zkratka pro Hyper Text Markup Language, je technologie pro tvorbu webových stránek, která se používá k vytváření a strukturování webových stránek. Skládá se z řady standardizovaných značek, které jsou uspořádány ve formátu textového souboru. Jazyk HTML slouží jako značkovací jazyk pro tvorbu webových stránek a umožňuje vývojářům vytvářet hypertextové dokumenty.

3.2.3 CSS

Podle Frain (2015) CSS neboli kaskádové styly je technologie používaná k ovládání rozvržení, barev, písem a celkové prezentace webové stránky. Funguje společně s jazykem HTML a slouží jako jazyk pro stylování webových stránek. Nejnovější verzí CSS je CSS3, která poskytuje lepší kontrolu nad zobrazením a vzhledem webových stránek. Zavádí různé moduly, včetně box modelu, modulu seznamu a modulu vícesloupcového rozvržení.

3.2.4 JavaScript

Podle mozilla (2019) JavaScript je lehký, za běhu kompilovaný programovací jazyk, který se běžně používá ve webových aplikacích. Umožňuje vývojářům vytvářet interaktivní a dynamická zobrazení pomocí manipulace s obsahem HTML, úprav stylů a reakcí na akce uživatele, jako jsou kliknutí myší a stisknutí kláves.

JavaScript je nejpopulárnějším programovacím jazykem na základě průzkumu Stack Overflow provedeného v roce 2023. (Stack Overflow, 2023),

3.2.5 JSX

Podle Meta Open Source (2024) JSX, zkratka pro JavaScript XML, je syntaktické rozšíření jazyka JavaScript, které umožňuje začlenit kód podobný XML/HTML do souborů jazyka JavaScript. To umožňuje vývojářům psát HTML/XML a využívat plný výkon jazyka JavaScript v jednom souboru. Kód JSX je převeden do objektů JavaScript, které mohou být zpracovány JavaScriptovými procesory, což podporuje integraci HTML/XML a JavaScriptu.

3.2.6 Kompilace

Podle mozilla (2024a) vývojáři píší kód v jazycích, jako je TypeScript, JSX a nové verze JavaScriptu. Tyto jazyky pomáhají vývojářům být efektivnější, ale je nutné je zkompilovat do jazyka JavaScript, aby jim prohlížeče rozuměly.
Kompilace znamená převzetí kódu z jednoho jazyka a jeho převedení do jiné verze tohoto jazyka nebo do jiného jazyka.

3.2.7 Objektový model dokumentu (DOM)

Podle MDN Web Docs (2019) objektový model dokumentu (DOM) slouží jako programovací rozhraní pro dokumenty HTML, XML a XHTML. Poskytuje reprezentaci webové stránky v podobě uzlů a objektů a umožňuje programům dynamicky přistupovat ke struktuře, stylu a obsahu dokumentu, upravovat je a manipulovat s nimi. Frameworky JavaScriptu se při interakci s webovými stránkami ve velké míře spoléhají na DOM a umožňují plynulou aktualizaci a vykreslování obsahu.

3.2.8 VDOM

Podle Sven Casteleyn et al. (2016) virtuální DOM (VDOM) je koncept používaný při vývoji webových stránek, který byl zpopularizován technologií React. Zahrnuje vytvoření virtuální reprezentace uživatelského rozhraní (UI) v paměti, která je synchronizována se skutečným objektovým modelem dokumentu (DOM) webové stránky. Díky porovnávání rozdílů mezi virtuálním a skutečným DOM se aktualizují pouze potřebné části uživatelského rozhraní, což urychluje a zefektivňuje manipulaci s DOM. Virtuální DOM automatizuje proces výpočtu a aplikace změn uživatelského rozhraní, čímž snižuje potřebu ručních aktualizací DOM ze strany vývojářů.

3.2.9 React.js

Podle facebook (2019) React.js je komponentový knihovna jazyka JavaScriptu pro vytváření uživatelských rozhraní. K popisu stavu uživatelského rozhraní používá deklarativní styl programování.[1] Pomocí React je možné vytvářet uživatelská rozhraní s využitím JSX. Komponenty React jsou funkce nebo třídy JavaScriptu, které vracejí kód JSX, který se transformuje do konečných objektů JavaScriptu zobrazených v aplikaci. React považuje komponenty a stavy za neměnné entity. To zajišťuje předvídatelné vykreslování a umožňuje efektivní manipulaci s DOM.
Podle Artemij Fedosejev a Bush (2015) Vnitřní systém řízení stavu React optimalizuje renderování tím, že při změně stavu aktualizuje pouze potřebné části uživatelského rozhraní, což vede k rychlejší manipulaci s DOM a zjednodušení logiky kódu.

[1] deklarativní programování je programovací paradigma - styl tvorby struktury a prvků počítačových programů - které vyjadřuje logiku výpočtu bez popisu jeho řídicího postupu (Sebesta, 2016)

3.2.10 Balíčkování

Balíčkování je proces řešení webových závislostí a slučování souborů do optimalizovaných balíčků pro webový prohlížeč, jehož cílem je snížit počet požadavků na soubory při návštěvě webové aplikace uživatelem. (Yasul 2023)

3.3 Technologie na straně serveru

3.3.1 Co jsou technologie na straně serveru

Technologie na straně serveru tvoří základ vývoje webových stránek a umožňují vytvářet dynamické webové stránky a webové aplikace. Tyto technologie jsou zodpovědné za obsluhu skrytých operací webových stránek, jako je zpracování dat, ověřování uživatelů a správa jejich obsahu.

3.3.2 Application programming interface (API)

API (Application programming interface) slouží k interakci mezi klientskou a serverovou stranou a usnadňuje doručování dat do libovolného zařízení.

3.3.3 SQLlite

SQLite je databáze[2] která je integrovaná přímo do aplikací, takže je bezserverová a vysoce přístupná. Funguje tak, že všechna data jsou uložena v jediném diskovém souboru v rámci aplikace. Samostatná povaha SQLite umožňuje integrovat celý databázový systém do aplikace pomocí jediné knihovny.

3.3.4 GraphQL

GraphQL je dotazovací jazyk používaný k interakci s aplikačními službami. Poskytuje jednotné rozhraní pro zadávání požadavků na služby vystavením dat jako grafu prostřednictvím schématu. Pomocí jazyka GraphQL mohou uživatelé odesílat dotazy do jediného koncového bodu a zadávat přesná data, která potřebují. Nabízí jasný popis

[2] Databáze je strukturovaný soubor dat nebo obsahu. (Oracle, 2022)

rozhraní API, podporuje jeho vývoj a umožňuje vývoj výkonných nástrojů.[3] (The GraphQL Foundation 2012)

3.3.5 CMS

Podle Davise (2020) CMS (Content Management System) je softwarový systém, který slouží k přidávání a úpravám informací na webových stránkách. Headless CMS jsou systémy pouze na straně serveru, které oddělují serverovou a klientskou stranu webových stránek. Existují dva typy Headless CMS: Řízené pomocí API a založené na systému Git. Systém CMS řízený pomocí API ukládá obsah do databáze a může být open-source nebo cloudový. Naproti tomu systém CMS založený na systému Git ukládá obsah do souborů v systému Git.[4]

3.3.6 Origin Server

Podle CDNetworks (2021) Origin server označuje hlavní počítač, na kterém je uložena a spuštěna původní verze kódu webové aplikace. Termín origin server se používá k odlišení tohoto hlavního serveru od ostatních míst, kam může být kód aplikace distribuován, jako jsou servery CDN a Edge servery.
Když origin server obdrží požadavek, provede před odesláním odpovědi určitý výpočet. Výsledek této práce zpracování dat může být následně přesunut do sítě CDN (Content Delivery Network).

3.3.7 CDN

Podle Cloudflare (2023) sítě CDN ukládají statický obsah (například obrázky a soubory HTML) na různých adresách po celém světě a jsou umístěny mezi origin serverem a klientem. Když je odeslán nový požadavek, nejbližší umístění sítě CDN uživateli odpoví výsledkem z cache[5] Tím se snižuje celkové zatížení origin serveru, protože výpočet nemusí probíhat při každém požadavku. Také je to rychlejší pro uživatele, protože odpověď přichází z umístění, které je mu geograficky nejbližší.

---

[3] Dotazovací jazyk je programovací jazyk pro vyhledávání a změnu obsahu databáze. (Risch, 2009)
[4] Git je systém pro správu verzí, který monitoruje změny v kódové bázi. (Atlassian, 2019)
[5] Cachování je způsob zrychlení aplikací ukládáním dat pro budoucí použití po jejich načtení. (Grudl 2024)

3.3.8 Edge

Podle Jamnadass (2024) Edge je pojem pro okrajovou část sítě, která je nejbližší k uživateli. Sítě CDN jsou součástí „Edge“, protože ukládají statický obsah na okrajích sítě. Stejně jako sítě CDN jsou i servery Edge umístěny na více místech po celém světě. Ale na rozdíl od sítí CDN, které ukládají pouze statický obsah, jsou některé servery Edge schopny spouštět malé části kódu. To znamená, že cachování může být prováděno na serverech Edge blíže k uživateli.

Pokud se část práce, která se dříve prováděla v prohlížeči klienta, přesune do Edge, zlepší se funkčnost aplikace a sníží se její latence.[6]

3.4 Volba renderování

3.4.1 Renderování webových stránek

Thakkar (2020) uvádí, že volba renderování definuje fázi, ve které se generuje HTML pro uživatele stránky. Může to být během požadavku HTTP (SSR), v době sestavení (SSG) nebo lokálně v prohlížeči.

3.4.2 Důležitost metody vykreslování

Thakkar (2020) píše, že renderování je klíčovým aspektem vývoje webových stránek, protože určuje, jak se uživatelské prostředí zobrazí uživatelům. Způsob vykreslování zvolený pro webové stránky nebo aplikace ovlivňuje jejich výkon, uživatelský zážitek a pracovní postupy vývojářů.

Podle Netlify (2023) výběr vhodné metody vykreslování je důležitý pro dosažení optimálního výkonu, zlepšení počáteční doby načítání, zlepšení SEO (optimalizace pro vyhledávače) a zajištění bezchybného zážitku pro uživatele.

3.4.3 Statické generování stránek (SSG)

Monus (2022) uvádí, že SSG zahrnuje generování HTML webových stránek předem, obvykle během procesu sestavování. Soubory JavaScript , CSS a HTML jsou předem renderovány a uloženy v Content Delivery Network (CDN) pro rychlé a efektivní

[6] Latence - je doba, která trvá od okamžiku odeslání požadavku do doby, kdy se vrátí odpověď. (mozilla 2024b)

doručení uživatelům.
SSG zajišťuje rychlé načítání, protože obsah je připraven k zobrazení, jakmile uživatel navštíví web. Je vhodný pro webové stránky s relativně statickým obsahem nebo pro ty, které nevyžadují aktualizace dat v reálném čase. S rostoucím počtem stránek však může SSG vést k delší době sestavení.

3.4.4 Vykreslování na straně serveru (SSR)

Podle společnosti heavy.ai (2024) Server-Side Rendering zahrnuje generování HTML pro každou stránku na serveru za běhu reakci na požadavky uživatele. Server zpracuje požadavek, načte všechna potřebná data a vygeneruje HTML, které je poté odesláno klientovi.
SSR umožňuje dynamický obsah a aktualizace dat v reálném čase. Poskytuje dobrou rovnováhu mezi výkonem a interaktivitou. SSR však může zvýšit zatížení serveru a dobu odezvy, zejména u vysoce dynamických webových stránek.

3.4.5 Vykreslování na straně klienta (CSR)

Osmani a Miller (2024) uvádějí, že CSR je metoda renderingu, která zahrnuje odeslání kódu JavaScriptu a HTML klientovi. JavaScript je zodpovědný za zobrazování uživatelského prostředí a načítání dat z API. Rendering na straně klienta probíhá v zařízení klienta, typicky v prohlížeči.
CSR poskytuje dynamické a vysoce interaktivní uživatelské prostředí. Umožňuje aktualizace dat v reálném čase a odstraňuje nutnost znovu načítat celou stránku při každé interakci. Úvodní načítání však může být časově pomalejší a vyhledávače mohou mít problémy s indexováním obsahu.
U moderních webových aplikací je běžné, že se v jedné webové aplikaci používá kombinace renderovacích metod, což bude provedeno rovněž v praktické části této bakalářské práce.

3.5 Next.js

3.5.1 Přehled framworků Next.js

Jak uvádí Jartarghar et al. (2022) Next.js je framework pro vytváření webových aplikací. Je postaven na Reactu a nabízí dodatečné struktury, funkce a optimalizace, které zjednodušují proces vývoje. Díky Next.js se mohou vývojáři soustředit na vytváření svých aplikací, místo aby trávili čas nastavováním nástrojů.
Klíčové vlastnosti:

- Automatické rozdělení kódu: Next.js podporuje automatické dělení kódu, které rozdělí aplikaci na menší části potřebné pro každý vstupní bod. Tím se zkrátí úvodní doba načítání aplikace tím, že se načte pouze kód potřebný pro každou stránku, čímž se zkrátí doba načítání; (Riva, 2022)
- Líné načítání: Next.js podporuje dynamický import, který umožňuje importovat React komponenty podle potřeby. Tato technika zlepšuje úvodní dobu načítání tím, že snižuje množství JavaScriptu potřebného předem;
- Vestavěná podpora CSS: Next.js umožňuje importovat styly CSS ze souborů JavaScriptu, což umožňuje rychlejší renderování díky načítání pouze těch stylů, které jsou potřeba pro každou stránku;
- SEO: Next.js poskytuje nástroje pro Search Engine Optimization (SEO), které umožňují snadno přidávat nadpisy, klíčová slova a další prvky související se SEO na každou stránku pomocí komponenty Head; (Shah 2022)
- Vercel: Open source hostingová platforma poskytovaná pro hostování webových aplikací Next.js;

Next.js si získal velkou popularitu a používá ho více než 100 000 webových stránek, včetně velkých společností, jako Netflix a Uber. (wappalyzer 2024b)

3.5.2 CSR v Next.js

Jak uvádí Riva (2022), při renderování na straně klienta (CSR) pomocí Next.js si prohlížeč nejprve stáhne minimální stránku HTML a poté JavaScript, který je pro danou stránku potřeba. JavaScript dá instrukce k aktualizaci DOM a vykreslí zbytek stránky.

Po prvním načtení stránky je navigace mezi různými stránkami na stejném webu je rychlejší, neboť se načtou pouze potřebná data a JavaScript pouze vykreslí části stránky, aniž by vyžadoval obnovení celé stránky.

V aplikaci Next.js je možné renderování na straně klienta implementovat dvěma způsoby:

1. Pomocí React háčku[7] useEffect() místo používání funkcí renderování na straně serveru; (getStaticProps a getServerSideProps)
2. Použití knihovny pro načítání dat, jako je SWR nebo TanStack Query, pro načítání dat na straně klienta (doporučeno týmem Next.js).

3.5.3 SSR v Next.js

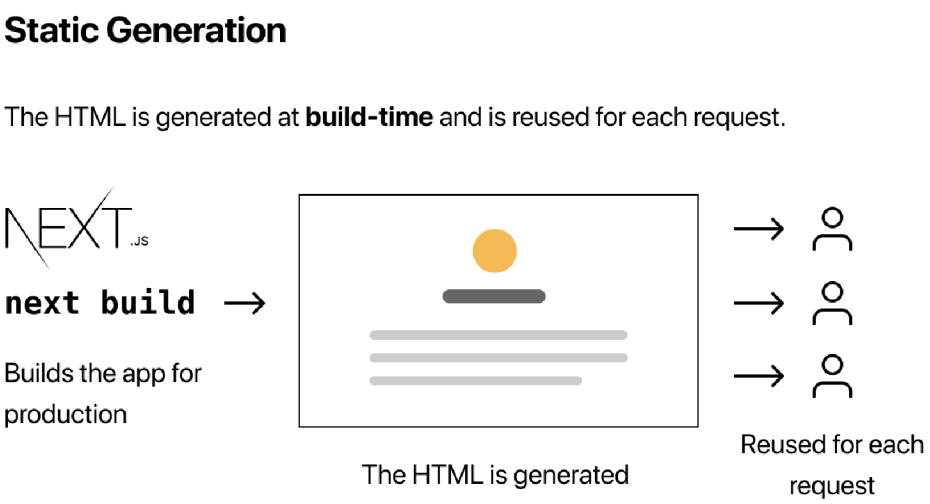
Při použití renderování na straně serveru v aplikaci Next.js se kód HTML webové stránky generuje na serveru pro každý jednotlivý požadavek. K tomu slouží pomocná funkce getServerSideProps.

Next.js uvadí (2024a), že na klientovi se HTML použije pouze k rychlému zobrazení neinteraktivní stránky, zatímco React použije instrukce JavaScriptu a JSON[8] data k tomu, aby komponenty byly interaktivní(například na obrázku níže připojí obsluhu události kliknutí na tlačítko). Tento proces se nazývá hydratace.

3.5.4 SSG v Next.js

V Dokumentaci Next.js (2024c) se uvádí, že pokud stránka Next.js používá funkci Static Site Generation, HTML stránky se generuje v době kompilace (krok sestavení). K tomu slouží pomocná funkce podobná té, která se používá v SSR, tedy *getStaticProps*. Pomocí této pomocné funkce vygeneruje Next.js HTML stránky, když se spustí příkaz *next build* Tento HTML kód se pak bude opakovaně používán při každém požadavku.

[7] React háčky jsou funkce JavaScriptu, které umožňují opakované použití jednotlivých komponent. (Larsen, J. 2021)
[8] JSON je textový datový formát založený na objektové syntaxi jazyka JavaScript. (Mozilla, 2022)



Obrázek 1 Generování statických stránek v Next.js (next.js 2024b)

3.6 Gatsby framework

3.6.1 Přehled framworků Gatsby

Podle oficiální dokumentace (Gatsby, 2021). Gatsby je framework, který byl vytvořen pro kombinaci Reactu a statického HTML. Gatsby poskytuje vývojářům frontendový framework, který pomáhá vytvářet rychlé a dynamické webové stránky, spolu s cloudovou platformou, která je poskytuje na CDN.

Klíčové vlastnosti:

- Alternativní možnosti vykreslování: Od verze Gatsby 4 má uživatel možnost zvolit si 3 možnosti vykreslování pro jednotlivé stránky. (Gatsby 2024c);
- Předběžné načítání stranky: Gatsby obsahuje integrovanou podporu pro přednačítání webových stránek;
- Rozdělení kódu: Gatsby používá rozdělení kódu pro zvýšení výkonu aplikace. Kód se během procesu kompilace rozdělí na menší části, což umožňuje načítání částí kódu podle potřeby; (Ikaheimo, 2022)
- Rozšiřitelnost a přizpůsobení: Gatsby nabízí bohatý ekosystém rozšiřujících modulů;
- Gatsby Cloud: Gatsby nabízí vlastní platformu pro hostování webových aplikací.

K 1. 7. 2023 je Gatsby druhým nejoblíbenějším renderovacím frameworkem hned po Next.js. používá ho více než 45 000 webových stránek, včetně velkých společností, jako jsou Meta a Microsoft. (wappalyzer 2024a)

3.6.2 CSR v Gatsby

V systému Gatsby probíhá vykreslování na straně klienta (CSR) podobně jako v Next.js, kdy se nejprve stáhne stránka HTML a poté se pomocí JavaScriptu aktualizuje objektový model dokumentu (DOM). Gatsby však při implementaci CSR používá jiný přístup, a to pomocí routerů, konkrétně @reach/router, který určuje typ webové stránky přidružené k dané adrese URL. To umožňuje dynamické vykreslování a navigaci v rámci aplikace Gatsby. (GatsbyJs, 2021)

3.6.3 SSG v Gatsby

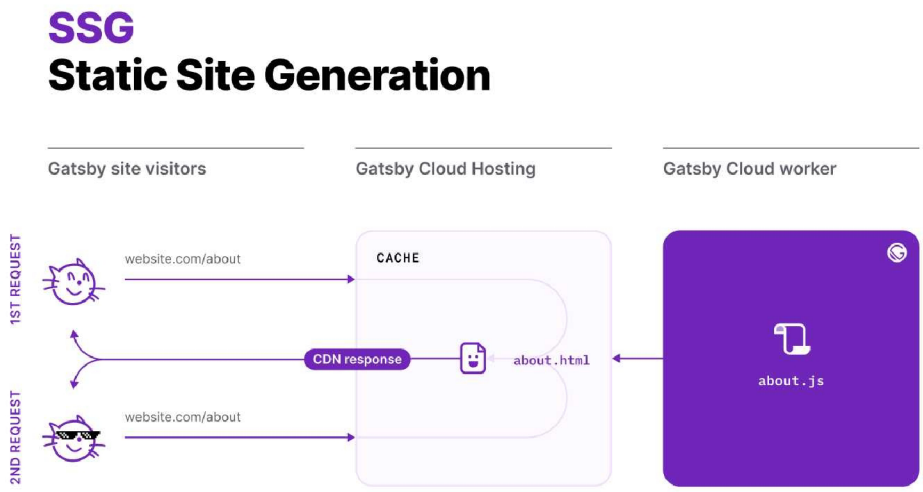
SSG je standardní renderovací režim aplikace Gatsby. SSG v Gatsby vykresluje a předzpracovává veškeré HTML, CSS a JS v době kompilace. Cela stránka je připravena k použití ještě předtím, než ji návštěvník navštíví.

SSG v Gatsby funguje takto:

1. Gatsby generuje HTML a veškeré zdroje pro stránky SSG v době kompilace na vybraném kompilačním serveru (např. osobní laptop, kompilační služba nebo Gatsby Cloud worker);
2. Vygenerované statické soubory se poté nahrají do CDN (Content Delivery Network), která je obsluhuje uživatelům. Po dokončení kroku sestavení lze sestavovací server dokonce vypnout.

SSG neznamená, že web není dynamický. JavaScript lze stále používat ke komunikaci s API, k tomu slouží přidání soukromých sekcí webu pro oprávněné uživatele prostřednictvím vykreslování na straně klienta (3.6.1).

Zde je uvedeno, jak SSG funguje v Gatsby Cloud (princip je stejný u jakéhokoli jiného poskytovatele sestavení a CDN):



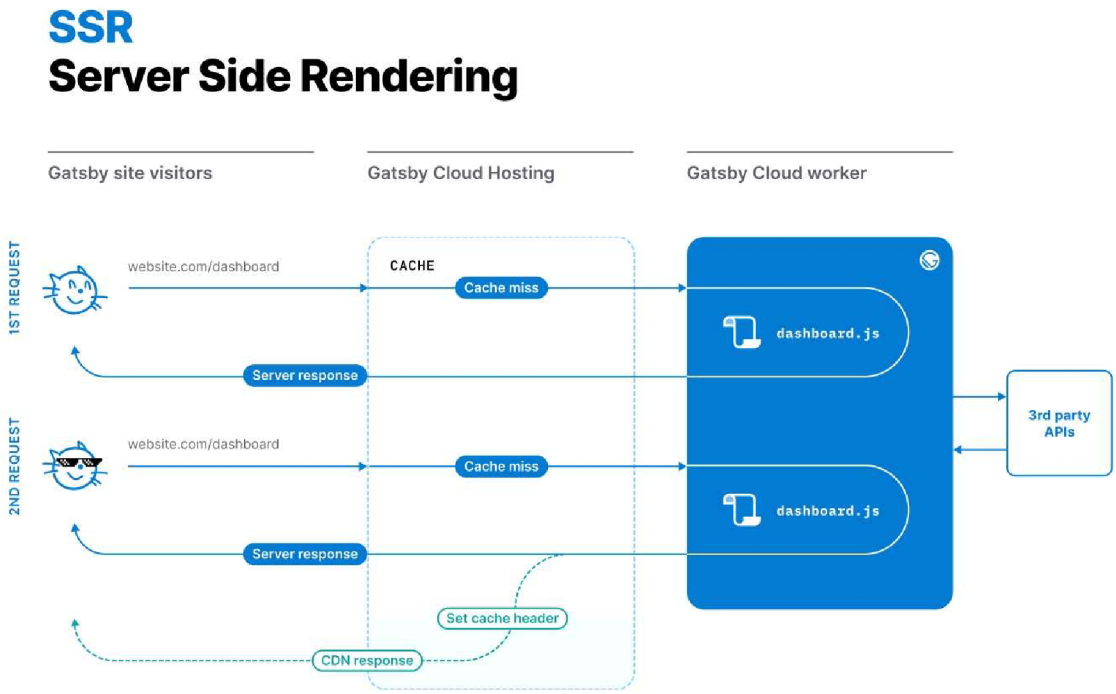
Obrázek 2 Generování statických stránek v Gatsby (Gatsby 2024c)

Jednou z nevýhod SSG v Gatsby stejně jako v Next.js je doba sestavení. S rostoucím počtem stránek webové aplikace roste i doba sestavení.

3.6.4 SSR v Gatsby

Server-Side renderování je nový prvek v systému Gatsby. SSR v Gatsby se chová podobně jako Next.js. Webová stránka, kterou návštěvníci vidí, je vždy ta nejnovější ze serveru. Stejně jako Next.js používal pomocnou funkci, která informovala o tom, že se aktuální stránka renderuje na serveru, Gatsby používá podobný přístup pomocí funkce *getServerData*. Která může komunikovat s libovolným API třetí strany, jakmile někdo

požádá o webovou stránku. (Gatsby 2024c)



Obrázek 3 Renderování na serverové straně v Gatsby (Gatsby 2024c)

3.7 Obdobná řešení

V následujících částech jsou uvedeny významné studie týkající se webových frameworků. Mezi těmito studiemi je i výzkum výkonnostních metrik a měřitelných atributů frameworků. Další studie se zabývá charakteristickými aspekty a zkoumá vlastnosti a funkce frameworků.

3.7.1 Ikaheimovo srovnání frameworků Next.js a Gatsby

Podobné srovnání bylo provedeno v článku od Ikaheimo (2022). Srovnání se zaměřuje především na kvalitativní[9] porovnání frameworků Next.js a Gatsby. Tato práce se liší v tom, že se provádí srovnání na základě různých kritérií a bude hlavně zahrnovat hodnocení kvantitativní[10] .

[9] Kvalitativní analytika zahrnuje analýzu nečíselných dat a subjektivních zkušeností pro hlubší pochopení.
[10] Kvantitativní analytika zahrnuje analýzu číselných dat a statistik za účelem získání poznatků.

3.7.2 Srovnání JavaScriptových frameworků od Olssona a Ockelberga

Olsson a Ockelberg (2020) zkoumali vhodnost různých JavaScriptových frameworků pro velké dynamické aplikace ve své bakalářské práci. Některá měření provedená v jejich práci posloužila jako inspirace pro testy provedené v praktické části této práce.

3.7.3 Výzkum aplikací postavených na architektuře Jamstack

Výzkum aplikací postavených na architektuře Jamstack s využitím vykreslovací metody SSG, který provedl Denysov (2021), zahrnuje kvantitativní hodnoty sdílené jak Next.js, tak Gatsby. Některé z nich, které jsou přítomny v jeho studii, se shodují s kritérii, která budou hodnocena v této práci.

3.8 Shrnuti

V teoretické části této práce je poskytnuto stručné shrnutí klíčových teoretických konceptů souvisejících s frameworky Next.js a Gatsby. Teoretická část byla se zaměřena na aspekty jako rendering, serverová a klientová strana, optimalizace výkonu, modularita a další. Díky této teoretické přípravě bylo získáno hlubší porozumění technickým principům, které formují tyto frameworky. Tyto poznatky budou následně aplikovány v praktické části práce, kde budou vytvořeny a testovány konkrétní aplikace s využitím frameworků Next.js a Gatsby.

4 Vlastní práce

Po uvedení teoretických konceptů a pevném porozumění metodikám vykreslování SSR i SSG se nyní tato práce zaměří na praktické využití. Tato část tvoří základ praktického aspektu této práce a zahrnuje návrh a konstrukci dvou aplikací, které jsou příkladem praktického využití SSR a SSG.

První aplikace bude dynamická, vytvořená pomocí SSR a bude prostřednictvím volání API komunikovat s Imgurem za účelem správy a zobrazování obrázků. Tím bude zvýrazněn způsob, jakým oba frameworky zvládají dynamické vykreslování na straně serveru v reálném čase.

Druhá aplikace bude naopak vytvořena pomocí SSG s důrazem na rychlost, výkon a efektivní přidělování prostředků. Její konstrukce bude zahrnovat hostování předem optimalizovaných obrázků uložených lokálně, čímž se využijí silné stránky statického generování stránek z hlediska optimalizace a rychlosti výkonu.

4.1 Počáteční nastavení

V této počáteční fázi projektu je prvním zásadním krokem vybudování robustního základu, který usnadní efektivní průběh vývojových úkolů a zajistí hladký průběh práce. Klíčové procesy v rámci tohoto nastavení zahrnují strukturování a vytvoření systému správy verzí, získání ID klienta pro rozhraní Imgur API, konfiguraci místního prostředí pro hostování aplikací.

4.1.1 Nastavení repozitáře git

Pro efektivní správu správy verzí bude vytvořen místní repozitář Git. Tento zásadní krok umožní systematické sledování, správu a kontrolu změn v rámci projektu. Tyto změny budou zahrnovat nejen úpravy zdrojového kódu, ale umožní také sledování a kontrolu aktualizací napříč návrhovými dokumenty, prostředky, skripty a dalšími prvky nezbytnými pro vývoj projektu.

Zpočátku se všechny transformace provedené v rámci adresáře projektu začlení do místního úložiště pomocí příkazu git add . Tato operace zahrnuje všechny nově zavedené soubory nebo adresáře od předchozího snímku.

Následně, aby byl zachycen okamžik vývoje projektu a uchován v historii verzí, přichází ke slovu příkaz git commit -m commit_message. Tento příkaz umožňuje sledovat

vývoj projektu v různých fázích a účinně slouží jako systém kontrolních bodů pro vývoj kódové základny.

4.1.2 Generating Client ID for Imgur API Integration

Aby bylo možné integrovat rozhraní API služby Imgur do aplikací Next.js a Gatsby SSR, je třeba získat ID klienta od společnosti Imgur. Toto ID klienta bude sloužit jako ověřovací mechanismus při zadávání požadavků na rozhraní Imgur API.

Pro získání klientského ID je třeba získat přístup na webové stránky Imgur a přejít do sekce pro vývojáře. Toto klientské ID je poskytováno zdarma a je nezbytné pro bezpečný přístup ke zdrojům rozhraní Imgur API.

Po získání ID klienta je bezpečně uloženo v proměnných prostředí aplikace. Toho se dosáhne vytvořením souboru .env.local v adresáři projektu, pokud tam ještě není. V tomto souboru je deklarována proměnná s názvem CLIENT_ID, které je přiřazena hodnota získaného ID klienta.

4.1.3 Hostování aplikace

Pro usnadnění místního hostování aplikací se používají specifické příkazy přizpůsobené jednotlivým frameworkům. Pro aplikace Gatsby se doporučuje použít příkaz pnpm serve. Naopak pro aplikace Next.js je preferovanou metodou příkaz pnpm start. Před hostováním je však nutné sestavit produkční balíček, což je zásadní krok prováděný pomocí příkazu pnpm build. Tato přípravná akce zajistí, že aplikace bude plně optimalizována pro nasazení, což zaručí optimální výkon a funkčnost v produkčním prostředí.

4.2 Vývoj SSR aplikací

Pro aplikaci SSR byl zvolen návrh, který zahrnuje funkce zahrnující požadavky API na album Imgur. Obrázky obsažené v tomto albu pocházejí z Unsplash, úložiště volně použitelných obrázků. Zvolený přístup zahrnuje načítání dat z rozhraní Imgur API a následné zobrazení načtených obrázků v rozhraní aplikace.

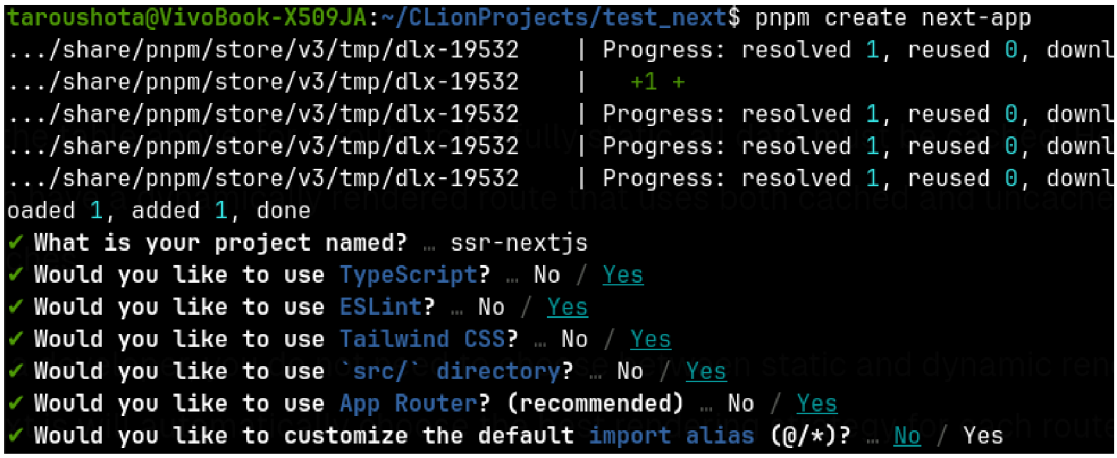
Cílem této volby návrhu je demonstrovat schopnost aplikací SSR dynamicky vykreslovat obsah získaný z externích zdrojů. Integrací požadavků API do aplikace SSR je záměrem předvést možnosti načítání dat v reálném čase v aplikacích Next.js a Gatsby.

4.2.1 Startovací šablona pro Next.js

Úvodní krok zahrnuje vygenerování nového projektu Next.js pomocí rozhraní příkazového řádku (CLI). V terminálu spusťte příkaz pnpm create next-app ssr-nextjs. Příkaz inicializuje aplikaci Next.js s názvem ssr-nextjs v aktuálním adresáři. Během procesu nastavení CLI vyzve uživatele k zadání různých konfigurací projektu. Pro naše hodnocení je zahrnuta integrace jazyka TypeScript, dále ESLint pro zajištění kvality kódu, Tailwind CSS pro zjednodušené stylování přímo v souborech .JSX a využití adresáře src/ pro organizaci zdrojových souborů. App Router nebude používán, proto je během procesu nastavení zvolena odpověď ne.

Rozhodnutí začlenit jazyk TypeScript zajišťuje zvýšenou typovou bezpečnost, zmírňuje běžné programátorské chyby a usnadňuje údržbu kódu. Integrace ESLint pomáhá při prosazování konzistentních standardů kódování a odhalování potenciálních problémů v rané fázi procesu vývoje.

Po dokončení procesu nastavení se přechod do adresáře projektu uskuteční pomocí terminálu, který se spustí příkazem cd ssr-nextjs. Po úspěšném přechodu do zamýšleného adresáře se zahájí proces iniciace vývojového serveru. To vyžaduje provedení příkazu pnpm dev.



Obrázek 4 Vytvoření startovací šablony pro Next.js v CLI

4.2.2 Startovací šablona pro Gatsby

Po spuštění příkazu pnpm create gatsby se spustí rozhraní řízené výzvou, které usnadní vytvoření nového webu Gatsby. Uživatel je vyzván k zadání různých konfigurací pro nastavení webu.

Po zahájení procesu je uživatel vyzván k zadání požadovaného názvu webu Gatsby. V tomto případě je web pojmenován ssr-gatsby. Dále je uživatel vyzván k určení názvu složky pro vytvoření webu, přičemž pro složku umístěnou v adresáři test_next je zvolen název ssr-gatsby.

Mezi další možnosti přizpůsobení patří výběr programovacího jazyka mezi JavaScriptem a TypeScriptem, přičemž pro tento případ byl zvolen TypeScript. Uživatel má navíc možnost integrovat systém pro správu obsahu, což je v tomto scénáři odmítnuto. Jako možnost je také uvedena integrace systému stylování, přičemž pro integraci stylování je vybrán Tailwind CSS.

Následně po výběru možností přizpůsobení nejsou k instalaci vybrány žádné další funkce ani doplňkové moduly.

```
What would you like to call your site?
✔ · ssr-gatsby
What would you like to name the folder where your site will be created?
✔ test_next/ ssr-gatsby
✔ Will you be using JavaScript or TypeScript?
· TypeScript
✔ Will you be using a CMS?
· No (or I'll add it later)
✔ Would you like to install a styling system?
· Tailwind CSS
✔ Would you like to install additional features with other plugins?No items were selecte
d
```

Obrázek 5: Vytvoření startovací šablony pro Gatsby v CLI

4.2.3 Typy odpovědí rozhraní Imgur API

Aplikace Next.js i Gatsby budou používat stejné typy odpovědí pro strukturování dat načtených z rozhraní API služby Imgur a pro vyjádření stavu požadavku API. Tyto typy odpovědí, ImgurDataResponse a ResponseStatus, zajišťují konzistenci při zpracování dat a správě chyb v obou frameworcích.

Typ ImgurDataResponse představuje data získaná z rozhraní Imgur API, včetně atributů, jako je název obrázku, popis, autor a adresa URL. Typ ResponseStatus označuje výsledek požadavku API, včetně stavových kódů a chybových zpráv, což usnadňuje zpracování chyb v aplikaci.

Definováním těchto typů odpovědí je vytvořen strukturovaný přístup ke zpracování a správě dat získaných z rozhraní Imgur API, což zvyšuje spolehlivost aplikace.

```typescript
export const ResponseStatus = {
  sucess: 200,
  notFound: 404,
  internalError: 500,
  noResponse: 0
} as const;


export interface FetchResult {
  data: ImgurDataResponse[],
  sucess: Boolean,
  status: typeof ResponseStatus[keyof typeof ResponseStatus],
}
interface ImgurDataResponse {
  description: null | string;
  edited: string;
  height: number;
  link: string;
  title: null | string;
  type: string;
  width: number;
}
```

Obrázek 6: Zdrojový kód typů odpovědí rozhraní Imgur API

4.2.4 Vytvoření požadavku Fetch API

V implementaci je k načítání obrázků z rozhraní Imgur API použito rozhraní Fetch API, přičemž do parametrů požadavku je zahrnuto ID klienta a hash alba. Tím je zajištěno ověření a umožněn přístup ke konkrétnímu albu obsahujícímu požadované obrázky.

Volání rozhraní Fetch API je zapouzdřeno ve funkci getServerSideProps v Next.js a v getServerData v Gatsby, což jsou vestavěné funkce poskytované respektivním frameworkem pro získávání dat na straně serveru.

```typescript
import type { FetchResult } from './types'

const myHeaders = new Headers();
myHeaders.append("Authorization", `Client-ID ${process.env.CLIENT_ID}`);

const requestOptions = {
  method: 'GET',
  headers: myHeaders,
  redirect: 'follow' as RequestRedirect
};
const ALBUM_HASH = "nW9BuSk"

export default {
  fetch: async ():Promise<FetchResult> =>{
    try{
      const response = await fetch(`https://api.imgur.com/3/album/${ALBUM_HASH}/images`, requestOptions);
      const responseResult = await response.json();
      return responseResult;
    }catch(error){
      console.log('error', error);
    }
  }
};
```

Obrázek 7: Zdrojový kód implementace rozhraní fetch api

4.2.5 SSR komponenta

V Next.js i Gatsby se k zobrazení obrázků načtených ze serveru používá komponenta SSRProductPage. Tato komponenta je zodpovědná za vykreslování obrázků získaných z procesu získávání dat na straně serveru.

V rámci komponenty SSRProductPage jsou načtená data nejprve zkontrolována, aby byla zajištěna úspěšná odpověď. Pokud je stav odpovědi úspěšný, pokračuje komponenta v iteraci načtených dat.

Každá položka v datovém poli odpovídá obrázku načtenému z rozhraní Imgur API. K vygenerování jednotlivých prvků pro každý obrázek je použita funkce map. Tyto prvky jsou pak zobrazeny v mřížkovém rozvržení.

```
const SSRProductPage = ({ serverData }: Props) => {
  const { fetchResult } = serverData
  if (fetchResult.status !== ResponseStatus.sucess) {
    return (
      <>
        <NotFoundPage />
      </>
    )
  }
  return (
    <div className='flex-1 max-w-full'>
      <div className="flex -mb-40 lg:-mb-56 col-auto columns-1 gap-1 justify-center flex-wrap basis-1 flex-auto size-auto">
        {fetchResult.data.map((a: any, index: number) => (
          <Link key={`link-order-${index}`} to={a.description.replace(/\s/g, '')}>
            <LazyLoadImage
              src={a.link}
              key={`image-order-${index}`}
              className="my-6 size-auto object-contain"
              alt={a.description}
              height={a.height}
              width={a.width}
              placeholder={<Image />}
              loading={index < 2 ? "eager" : "lazy"}
              threshold={200}
              effect="blur"
            />
          </Link>
        ))}
      </div>

    </div>
  )
}

export default SSRProductPage
```

Obrázek 8: Hlavní SSR Gatsby komponenta

```
const SSRProductPage = ({ serverData }: Props) => {
  const { fetchResult } = serverData
  if (fetchResult.status !== ResponseStatus.sucess) {
    return (
      <>
        <NotFoundPage />
      </>
    )
  }
  return (
    <div className='flex-1 max-w-full'>
      <div className="flex -mb-40 lg:-mb-56 col-auto columns-1 gap-1 justify-center flex-wrap basis-1 flex-auto size-auto">
        {fetchResult.data.map((a: any, index: number) => (
          <Link key={`link-order-${index}`} to={a.description.replace(/\s/g, '')}>
            <LazyLoadImage
              src={a.link}
              key={`image-order-${index}`}
              className="my-6 size-auto object-contain"
              alt={a.description}
              height={a.height}
              width={a.width}
              placeholder={<Image />}
              loading={index < 2 ? "eager" : "lazy"}
              threshold={200}
              effect="blur"
            />
          </Link>
        ))}
      </div>

    </div>
  )
}

export default SSRProductPage
```

Obrázek 9: Hlavní SSR Next.js komponenta

4.3 Vývoj SSG aplikací

U aplikace SSG (Static Site Generation) byl ve srovnání s obdobnou aplikací SSR (Server-Side Rendering) zvolen odlišný přístup k návrhu. V tomto návrhu jsou obrázky získávány lokálně, nikoli dynamicky z externích rozhraní API. Rozhodnutí využít lokálně uložené obrázky bylo učiněno kvůli požadavku nástroje pro zpracování obrázků Gatsby-Sharp, který vyžaduje lokální uložení obrázků pro optimalizaci obrázků.

4.3.1 Typescript typy pro SSG komponenty

Typy používané v aplikacích Next.js i Gatsby zůstávají konzistentní, s výjimkou specifického požadavku v Gatsby, kde je nutný typ ImageNode. Tento rozdíl vzniká proto, že Gatsby používá k načítání obrázků dotazy GraphQL, což vyžaduje použití typu ImageNode k reprezentaci obrázkových dat ve schématu GraphQL. Naproti tomu Next.js používá relativní cesty k odkazování na lokálně uložené obrázky přímo v kódu aplikace, aniž by bylo nutné používat dotazy GraphQL.

```
type ImageNode = {
  node: {
    childImageSharp: {
      gatsbyImageData: IGatsbyImageData
    }
  }
}

type AllFileData = {
  allFile: {
    edges: ImageNode[]
  }
}


interface Product {
  id: string
  title: string
  description: string
  price: number
  stock: number,
  name: string,
  features: string,
  action: string;
}

interface Props {
  product: Product,
  data: AllFileData
}
```

Obrázek 10 Typescript typy pro SSG komponenty

4.3.2 Komponenta statické stránky Gatsby

Nyní, když jsou definovány typy, byla vytvořena komponenta StaticPage, která zobrazuje obrázky vyřazené z dotazu graphql, což lze vidět ve funkci data.allFile.edges.map.

```
const queryData = useStaticQuery(graphql`
query {
  allFile(filter: { extension: { eq: "jpg" } }) {
    edges {
      node {
        childImageSharp {
          gatsbyImageData(width: 320, height: 320, transformOptions:{fit:INSIDE})
        }
      }
    }
  }
}
```

Obrázek 11: Dotaz GraphQL v SSG komponentě

```
function StaticPage({ product, data }: Props) {
  return (
    <div className="min-h-screen h- bg-black text-white  flex items-center justify-center">
      <div className="p-8  max-w-2xl text-center">
        <div className="mb-16 grid grid-cols-2 grid-rows-2 gap-4" >
          {data.allFile.edges.map((i: { node: { childImageSharp: { gatsbyImageData: IGatsbyImageData } } }, index: number) => (
            <div key={`container-${index}`}>
              <GatsbyImage
                key={`img-${index}`}
                image={getImage(i.node.childImageSharp.gatsbyImageData) as IGatsbyImageData}
                alt={`Product ${index}`}
              />
            </div>
          ))}
        </div>
      </div>
    </div>
  )
}
```

Obrázek 12: Gatsby zdrojový kód komponenty StaticPage

4.3.3 Komponenta statické stránky Next.js

Komponenta StaticPage v Next.js nevyžaduje použití dotazu GraphQL pro zobrazení obrázků. Funkce product.images.map iteruje přes pole relativních cest k příslušným obrázkům.

```
function StaticPage({ product }: Props) {
  return (
    <div className="min-h-screen h- bg-black text-white  flex items-center justify-center">
      <div className="p-8  max-w-2xl text-center">
        <div className="mb-16 grid grid-cols-2 grid-rows-2 gap-4" >
          {product.images.map((i, index) => (
            <div key={`container-${index}`}>
              <Image
                key={`img-${index}`}
                priority={index < 2}
                src={i}
                alt={i}
                className="rounded-md mx-auto"
              />
              <h1 className="text-xl font-bold mt-2">${10 + index * 5}</h1>
            </div>
          )
          )}
        </div>

      </div>
    </div>
  )
}
```

Obrázek 13: Next.js zdrojový kód komponenty StaticPage

4.4 Hardware

V této části budou popsány testy, které byly provedeny za účelem získání údajů, které jsou zde uvedeny.

a analyzována v následující kapitole. Všechny testy byly provedeny v prohlížeči Google Chrome verze 120.0.6099.199 (64bitový) v režimu inkognito a byly provedeny na počítači ASUS X509JA s následujícími specifikacemi:

- processor: i3-1005G1 @ 1.20GHz;
- RAM: 8 GB DDR4 2667MHz;
- integrovaná grafika: 10th Gen UHD Graphics;
- operační systém: Zorin OS 16.3;
- internet: stahování a odesílání až do 500 Mb/s.

4.5 Kritéria hodnocení

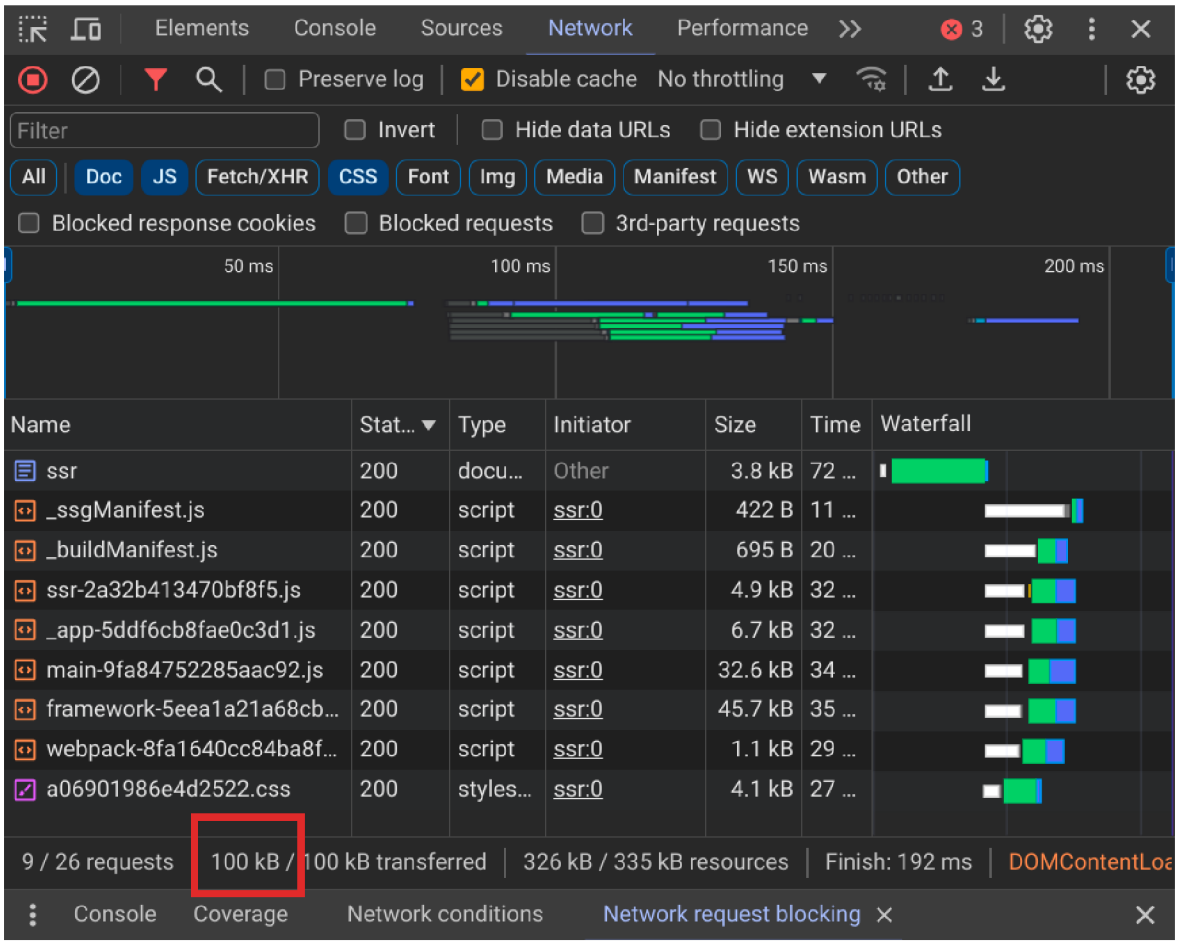
V této části jsou uvedeny způsoby, jakými budou testovány aplikace SSR a SSG. Prvním kritériem je doba potřebná k vytvoření balíčku aplikace připravené k použití. Poté velikost balené aplikace, včetně prostředků JavaScript, CSS a HTML. Následuje, že rychlost načítání stránky a také možnosti optimalizace obrázků ve dvou metodách vykreslování. Také bude zahrnovat celkové množství dostupných balíčků npm

4.5.1 Doba sestavení baliku

Příkaz linux time bude použit k výpočtu času potřebného k vytvoření balíčku aplikace připravené k produkci pro Gatsby A Next.js v obou metodách vykreslování. Podle Waterse (1996) je linuxový příkaz time nástrojem, který se používá k určení výkonu jednoho programu. Příkaz time informuje o době, která uplynula od začátku do konce programu.

4.5.2 Velikost baliku

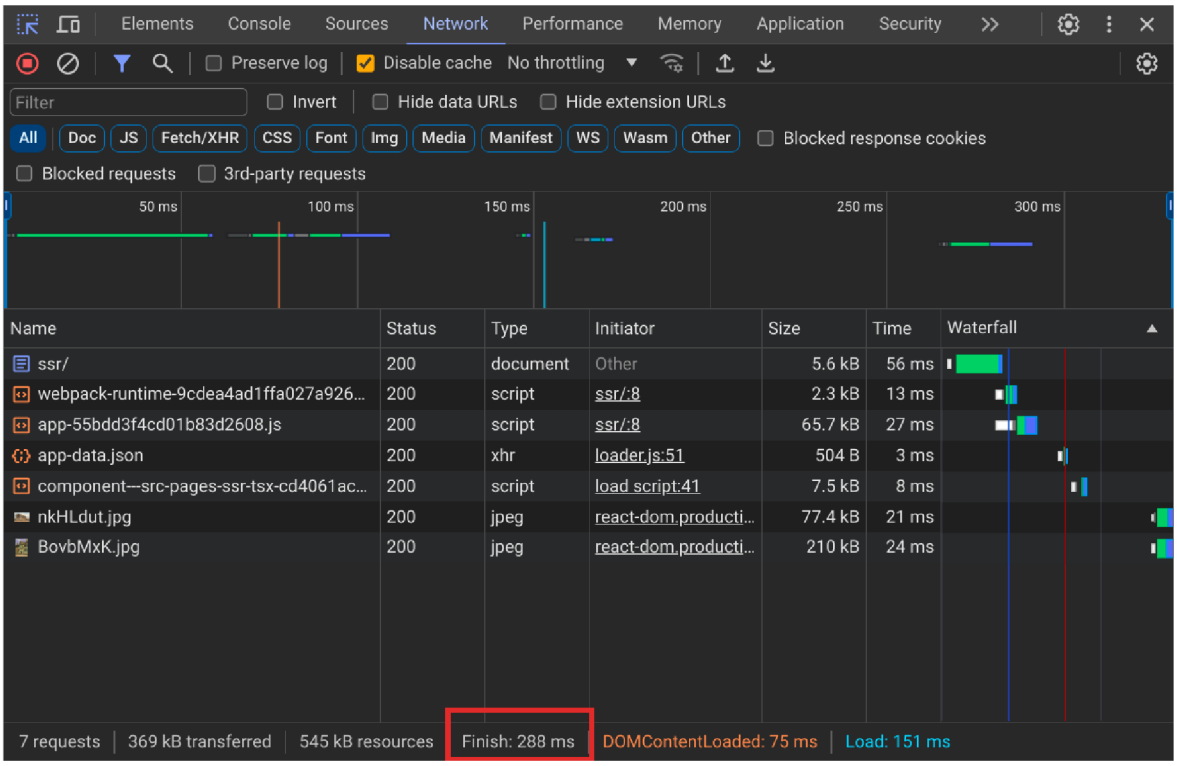
K přesnému měření velikosti produkčního balíčku se používají nástroje Chrome DevTools. V nástroji DevTools je vybrána karta Síť a jsou vybrány pouze požadavky na HTML, CSS a JavaScript. Tento výběr zajišťuje, že jsou do balíku zahrnuty všechny prostředky a že jsou vyloučeny všechny další prostředky, které by mohly vykazovanou velikost nafouknout. A k měření původní velikosti aplikace se použije správce souborů Nautilus, což je výchozí správce souborů pro Zorin OS 16.3.



Obrázek 14: Porovnání velikosti balíčků aplikací v nástroji Chrome DevTools

4.5.3 Doba spuštění aplikace

K měření doby spuštění se použije nástroj DevTools. Stránka je považována za funkční po dokončení všech požadavků na všechny soubory v produkčním balíčku. Mezi které patří soubory HTML a JavaScript. Proto budou v tomto srovnání použity výsledky události Finished, protože udávají celkovou dobu potřebnou k dokončení požadavků. Po události Finished jsou všechny požadavky vyřízeny. Protože je server pro webové aplikace umístěn lokálně, mají obě aplikace průměrně 13ms zpoždění požadavků.



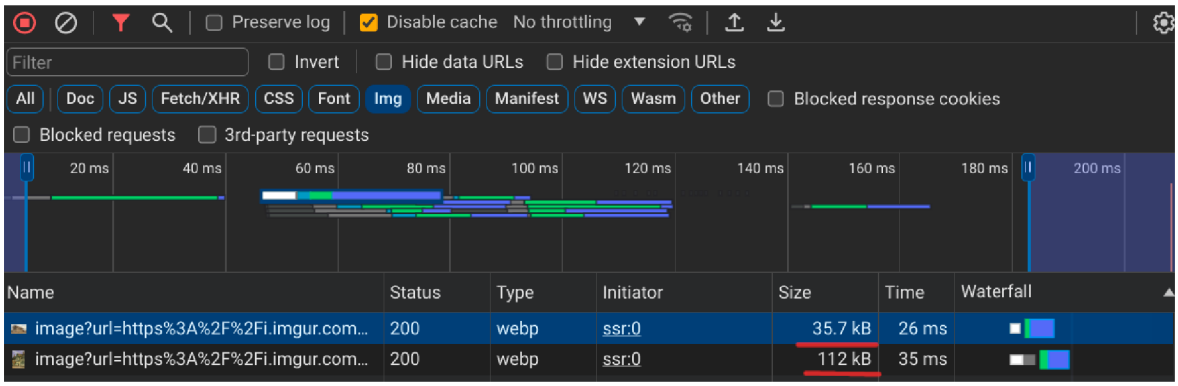
Obrázek 15: Porovnání časů spouštění aplikací v nástroji Chrome DevTools

4.5.4 Optimalizace obrázků

Pro analýzu možností optimalizace obrázků obou rámců budou změřeny původní velikosti obrázků a velikosti obrázků načtených na produkční stránce pomocí karty Síť v nástroji Chrome DevTools a správce souborů Nautilus.

V případě nástroje Chrome DevTools jde o kontrolu síťových požadavků provedených stránkou a jejich filtrování se zaměřením konkrétně na požadavky na obrázky.

V případě správce souborů Nautilus se jedná o kontrolu složky /public, která obsahuje všechny obrázky.



Obrázek 16: Porovnání optimalizovaných velikostí obrázků v nástroji Chrome DevTools

4.5.5 Bohatý ekosystém balíčků

Abychom mohli posoudit rozsáhlost a bohatost ekosystému balíčků, které jsou přiřazeny jednotlivým frameworkům, bereme v úvahu výsledky vyhledávání získané z npm - správce balíčků Node. Jako nedílná součást běhového prostředí JavaScriptu Node.js je npm úložištěm veřejných knihoven a balíčků.

Nejde jen o správce balíčků, npm slouží jako tržiště pro vývojáře open source, kteří mohou sdílet a používat balíčky. Data získaná z výsledků vyhledávání v npm tak nabízejí neocenitelný přehled o šíři a hloubce dostupných balíčků, knihoven a zásuvných modulů vytvořených speciálně pro jednotlivé frameworky - Next.js a Gatsby - nebo s nimi kompatibilních.

5 Výsledky a diskuse

Tato část obsahuje komplexní analýzu výkonnostních ukazatelů spojených s aplikacemi využívajícími metody vykreslování na straně serveru (SSR) a statického generování stránek (SSG). Vyhodnocení zahrnuje klíčová kritéria včetně času potřebného k vytvoření balíčku aplikace, velikosti vzniklého balíčku, doby spuštění, optimalizace obrázků a balicku ekosystému. Prostřednictvím systematického testování a analýzy jsou testovány výkonnostní charakteristiky frameworků Next.js a Gatsby

5.1 Výsledky

V této části je zobrazen výkon aplikací využívajících metody vykreslování na straně serveru (SSR) a statického generování stránek (SSG). Kritérii jsou doba potřebná k vytvoření balíčku aplikace, velikost hotového balíčku, rychlost načítání stránek a optimalizace obrázků.

5.1.1 Doba balíčkování

Při porovnání rychlosti vytváření balíků u obou frameworků pro staticky generované webové stránky dosáhl nextjs lepších výsledků v průměru o 9,884 sekundy v absolutní hodnotě a o 38,83 % v relativních hodnotách.

Tabulka 1: Porovnani doby balíčkování statické generovaných aplikaci

	vzorků	průměr [s]	MIN [s]	MAX [s]
Gatsby	20	25,452	27,004	27,094
Next.js	20	15,568	15,191	15,929
Rozdíl	20	38,83%	43,74%	41,20%

A v případě aplikací, které byly vykreslovány na straně serveru, dosáhl nextjs také lepších výsledků, když vykázal rozdíl 24,328 sekundy v absolutních hodnotách a 56,95 % v relativních hodnotách.

Tabulka 2: Porovnani doby balíčkování aplikací renderovaných na straně serveru

	vzorků	průměr [s]	MIN [s]	MAX [s]
Gatsby	20	42,714	41,597	46,276
Next.js	20	18,386	18,264	18,531
Rozdíl	20	56,95%	56,09%	59,95%

5.1.2 **Velikost baliku**

Měření velikosti balíčku po sestavení pro aplikace generované na straně serveru zahrnovalo všechny soubory v produkčním balíčku. Velikost po sestavení aplikací gatsby a next.js na straně serveru se lišila o 18 kB.

Tabulka 3: Porovnání velikosti balíků aplikací vykreslovaných na straně serveru

	Size after build [kB]
Gatsby	81
Next.js	99
Rozdíl	-22,22%

Měření velikosti balíčku zahrnovalo všechny soubory v produkčním balíčku aplikace. Velikost po sestavení staticky generovaných webových stránek gatsby a next.js se lišila o 20 kB.

Tabulka 4: Srovnání velikosti balíčků aplikací pro staticky generované aplikace

	Size after build [kB]
Gatsby	103
Next.js	98,9
Rozdíl	4,85%

5.1.3 **Optimalizace obrázků**

Hodnocení optimalizace velikosti obrázků bylo omezeno na staticky generované webové stránky z důvodu absence podpory dynamické optimalizace obrázků v aplikacích

vykreslovaných na straně serveru v rámci systému Gatsby do 01.02.2024. Naproti tomu Next.js takovou podporu nabízí a umožňuje optimalizaci obrázků ve staticky generovaných aplikacích i v aplikacích vykreslovaných na straně serveru.

Výsledky měření ukazují, že next.js měl velikost svazku menší o absolutních 25 kB, což je v relativních hodnotách o 7,3 % méně než Gatsby.

Tabulka 5: Srovnání velikostí obrázků pro staticky generované aplikace

	Původní velikost [kB]	Komprimovaná velikost [kB]
Gatsby	1,6 MB	342 kB
Next.js	1,6 MB	317 kB
Rozdíl	0	7,30%

5.1.4 **Doba spuštění**

V době potřebné k načtení staticky generovaných webových stránek byl Gatsby konzistentně překonán frameworkem Next.js, přičemž průměrný rozdíl mezi oběma frameworky činil 79,38 milisekundy.

Tabulka 6: Srovnání doby spuštění staticky generovaných aplikací

	vzorků	průměr [ms]	MIN [ms]	MAX [ms]
Gatsby	20	206,88	173	260
Next.js	20	127,50	100	162
Rozdíl	20	38,37%	42,19%	37,69%

V době načítání serverem renderovaných webových stránek Next.js vykazuje také lepší hodnoty. Gatsby byl opět konzistentně překonán Next.js, přičemž průměrný rozdíl mezi oběma frameworky činil 75,29 milisekundy.

Tabulka 7: Srovnání doby spuštění aplikací vykreslovaných na straně serveru

	vzorků	průměr [ms]	MIN [ms]	MAX [ms]
Gatsby	20	222,14	195	268
Next.js	20	146,85	115	193
Rozdíl	20	33,89%	41,02%	27,98%

5.1.5 Bohatý ekosystém balíčků

K datu 01.02.2024 bylo zjištěno, že Gatsby má o 3039 dostupných balíčků více než Next.js.

Tabulka 8: Celkové množství NPM balíků dostupných k 01.02.2024 pro oba frameworky

	Celkové množství NPM balíků
Gatsby	4240
Next.js	1201
Rozdíl	72,33%

5.2 Zhodnocení

Tabulka 9: Bodové rozdělení při měření kvantifikovatelných hodnot.

Hodnocení	Význam	Rozdíl [%]
0	Nelze použít	
1	Mnohem horší	(-∞, -25)
2	Horší	[-25, -10)
3	Stejný	[-10, 10)
4	Lepší	[10, 25)
5	Mnohem lepší	[25, ∞)

Výše je uvedeno, jak jsou body rozděleny pro měření měřitelných údajů, čímž se stanoví měřítka pro hodnocení výkonu framworků Next.js a Gatsby v metodách

vykreslování na straně serveru a statického generování stránek. Konečné rozložení bodů je uvedeno níže.

Tabulka 10: Přehled bodového hodnocení studie

Criteria	Next.js SSR	Gatsby SSR	Next.js SSG	Gatsby SSG
Doba balíčkování	5	1	5	1
Velikost baliku	2	4	3	3
Optimalizace obrázků	5	0	3	3
Doba spuštění aplikace	5	1	5	1
Bohatý ekosystem balíčků	1	5	1	5
Součet	**18**	**11**	**17**	**13**

Na základě výsledků testů vykázal Next.js 7 bodový rozdíl oproti Gatsby v aplikacích SSR a 4 bodový rozdíl v metodě vykreslování SSG. Absence optimalizace obrazu v SSR Gatsby byla ve srovnání vedle pomalejší doby spuštění a rychlosti balíčkování výraznou nevýhodou.

Naproti tomu Gatsby SSG vykazoval lepší výkon, což lze přičíst zejména srovnatelným schopnostem optimalizace obrazu. Nedostatky však přetrvávaly, pokud jde o rychlost balíčkování a dobu spouštění. Kromě toho se zvýšila velikost balíčků.

Tato zjištění se shodují se závěry, které učinil Ikaheimo (2022), a naznačují, že vykreslovací schopnosti Gatsby na straně serveru nejsou tak dokonalé jako u Next.js, jak dokazují výsledky testování uvedené v této práci.

Další krietaria, která se shodují se zjištěními Ikaheimo (2022), modulový ekosystém Gatsby překonal ekosystém Next.js. Toto pozorování poukazuje na potenciální výhodu pro vývojáře využívající Gatsby, protože bohatší ekosystém poskytuje přístup k širšímu spektru zdrojů a nástrojů pro vývoj aplikací.

Velikost obrázků po optimalizaci se objevuje také ve výzkumu Denyšova (2021) o porovnávání webových stránek, postaveném na metodě vykreslování SSG. Výsledky testování v této práci se shodují s výsledky zobrazenými Děnisovem. Rozdíly ve velikosti

obrázků mezi Gatsby a Next.js pro mobilní uživatele se v jeho studiích také pohybují kolem 0 až 10 %.

Srovnání velikosti souborů .js se objevuje také ve výzkumu Denysova (2021). Výsledky testování ukazují rozdíl 4,85 % mezi produkčními balíčky Next.js a Gatsby. Velikost svazku Next.js SSG je v Denysovově reserši také větší než u Gatsby.

5.3 Diskuse

Takže začít diskusi o tom, proč výsledky benchmarků byly differnt počínaje rychlostí svazku. V případě next.js se používá kompilátor SWC napsaný v programovacím jazyce Rust, který se kompiluje do strojového kódu. Podle Kanga (2020) je SWC 20krát rychlejší než Babel, který ke kompilaci používá Gatsby.

Co se týče velikosti balíčku v kontextu generování statických stránek (SSG), Gatsby a Next.js používají při poskytování zdrojů na straně klienta odlišné postupy. Gatsby ve své implementaci SSG dodává na stranu klienta stručnou sadu 5 souborů, z nichž největší složkou je soubor JavaScriptu o velikosti 65 kB. Zbytek o velikosti 30 kB je rozdělen mezi zbývající soubory. Naproti tomu Next.js se drží podobného architektonického paradigmatu, ale klientovi dodává širší sadu 10 souborů, přičemž hlavní soubor JavaScript má velikost 50 kB. Rozdělení zbývajících 49 kB mezi doplňkové soubory dále podtrhuje přístup Next.js k optimalizaci přidělování prostředků pro scénáře SSG.

V případě SSR gatsby doručuje celkem 4 soubory namísto 5 jako v SSG, čímž se jeho velikost sníží přibližně o 20 kB. Při kontrole licence souboru je vidět, že se jedná o starší rozhraní API react-dom/server. Po vyhodnocení obdržených výsledků bylo zjištěno, že gatsby odesílá tento soubor v případě, že je v aplikaci iniciován dotaz GraphQL. V případě staticky generovaných webových stránek se jednalo o querry na knihovnu gatsby-sharp pro optimalizaci lokálních obrázků. Protože v Gatsby SSR není žádný GraphQL querry, neexistuje ani soubor react-dom-server, proto je mezi Gatsby SSR a Gatsby SSG rozdíl 20 kbs. Když Nextjs používá novější verzi react-dom/server namísto starší verze. Která se klientovi servíruje v aplikacích SSG i SSR.

Optimalizace kritérií obrázků v kontextu SSG, Next.js a Gatsby vykazovaly podobný výkon s rozdíly menšími než 10 %.

V případě doby spuštění byl rozdíl více než 30 %, což naznačuje, že next.js je v obou metodách vykreslování mnohem rychlejší než gatsby.

Po zobrazení vodopádu v nástroji Chrome DevTools pro oba počáteční požadavky aplikace na svazek s HTML a JavaScriptem odešle Next.js nejprve požadavek na HTML, znovu jej přijme, parsuje a poté najednou odešle požadavek na ostatní soubory .js, .css a také obrázky, které jsou ve viewportu. V případě obrazovky 1980x1080 to jsou čtyři obrázky. Pokud stránka nebyla posunuta dolů, aby se zobrazily zbývající 4 obrázky, pak se zbývající 4 obrázky stáhnou po dokončení prvních 4 požadavků na obrázky. Poté na webové stránce nezůstanou žádné nevyřízené požadavky a Finish událost je vyvolána.

Nyní v případě gatsby nejprve odešle požadavek na html, počká, obdrží jej, parsuje jej, pak odešle požadavek na 2 soubory .js, počká, až se vyřeší, pak odešle požadavek na všechny 8 obrázky a teprve poté odešle požadavek na zbývající 2 soubory .js. Zbývající 2 soubory .js jsou nezbytné pro funkčnost webové stránky, protože se jedná o komponentu React pro samotnou stránku, proto dokud nejsou načteny tyto 2 soubory .js, nemůže stránka používat háčky React, jako je například useState, který se používá pro správu stavu. Událost finish je vyvolána poté, co jsou vyřešeny odpovědi těchto zbývajících 2 souborů .js.

Rozdíly v mechanismu načítání souboru JavaScript .js lze přičíst odlišným konfiguracím WebPacku, které používají jednotlivé frameworky. Proces načítání v Gatsby, který se vyznačuje řazením do fronty a dodatečným čekáním na každou dvojici požadavků, vede k pomalejšímu načítání ve srovnání s aplikacemi Next.js a vykazuje výkonnostní zpoždění 38,37 %.

Co se týče bohatosti ekosystému balíčků, dostupnost zásuvných modulů Gatsby a Next.js zůstává konzistentní napříč jejich metodami vykreslování. Next.js za Gatsby výrazně zaostává, protože má v NPM k dispozici o 3039 pluginů méně. Tento rozdíl v dostupnosti pluginů je důvodem vyššího skóre Gatsby o 5 bodů, zatímco Next.js získává v tomto kritériu 1 bod.

6 Závěr

Hlavním cílem této práce bylo identifikovat nejlepší případy použití frameworků Gatsby a Next.js, což je cíl dosažitelný díky komplexnímu srovnání těchto frameworků a jejich příslušných vykreslovacích metod. Next.js se ukázal jako lídr ve výsledcích obou vykreslovacích metod, zejména jeho architektura SSR vykazuje vynikající výkonnostní charakteristiky. Proto bylo zjištěno, že nejvhodnějším případem použití Next.js jsou dynamické aplikace SSR využívající malý počet pluginů.

Přestože aplikace SSG Gatsby dosahovaly lepších výsledků než její protějšek SSR, stále vykazovaly horší výkon ve srovnání s aplikací SSG Next.js, což bylo způsobeno především nižší rychlostí balíčkování a dobou spouštění. Gatsby však disponuje rozsáhlým ekosystémem pluginů, který nabízí potenciál pro zrychlení vývoje aplikací. Proto jsou nejlepším případem použití systému Gatsby aplikace SSG s využitím jeho obrovského množství pluginů.

Prvním dílčím cílem práce bylo vytvoření aplikací s využitím obou frameworků, což vedlo k vytvoření čtyř aplikací, z nichž dvě byly postaveny na každém frameworku. Klíčový rozdíl mezi nimi spočívá ve způsobu vykreslování. V práci byl zpracován postup a zpřístupněny zdrojové kódy těchto aplikací, které jsou rovněž uvedeny v části příloh.

Druhým dílčím cílem práce bylo komplexně zhodnotit výkonnost aplikace spolu s dalšími relevantními faktory. Prostřednictvím pečlivé analýzy předložené v rámci práce sloužila tato hodnocení jako srovnávací kritéria, která osvětlují odlišné výhody jednotlivých aplikací. Tato komplexní analýza poskytla cenné poznatky o výkonnostních možnostech aplikací a přispěla k hlubšímu pochopení jejich funkčnosti.

Práce představuje unikátní přínos díky propojení teoretických konceptů s praktickými experimenty na vybraných frameworcích. Je pozoruhodné, že k 1. únoru 2024 existuje nedostatek kvantitativních výzkumů, které by se konkrétně zabývaly srovnáním rychlosti a doby spouštění balíčků aplikací v obou metodách vykreslování. Tato výzkumná mezera podtrhuje význam této práce, která poskytuje vhled do výkonnostních charakteristik Next.js a Gatsby v různých scénářích.

Rozšíření této studie o další vykreslovací frameworky kromě těch postavených na Reactu, jako jsou Nuxt.js, SvelteKit a Astro, by mohlo poskytnout cenné poznatky o výkonnostních charakteristikách širšího spektra technologií. Zahrnutím těchto frameworků by mohlo být možné rozeznat rozdíly v jejich schopnostech a určit optimální případy

použití pro každý z nich. Takové rozšíření by mohlo obohatit poznání širšího spektra vykreslovacích frameworků a přispět k informovanějšímu rozhodování v praxi vývoje webových stránek.

7 Seznam použitých zdrojů

ARTEMIJ FEDOSEJEV a Alex BUSH, 2015. *React.js essentials : a fast-paced guide to designing and building scalable and maintainable web apps with React.js*. Birmingham: Packt Publishing. ISBN 9781783551620.

AZUBUIKE, Prince , 2023. Understanding Virtual DOM vs Real DOM as a frontend Developer. *www.zuri.team* [online] [vid. 2024-03-09]. Dostupné z: https://www.zuri.team/resources/understanding-virtual-dom-vs-real-dom-as-a-frontend-developer

CDNETWORKS, 2021. What Is An Origin Server? *CDNetworks* [online]. Dostupné z: https://www.cdnetworks.com/knowledge-center/what-is-origin-server/

CLOUDFLARE, 2023. What is a CDN? | How do CDNs work? | Cloudflare. *Cloudflare* [online]. Dostupné z: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

DAVIS, Sean, 2020. Choose the right CMS for you | Ample Blog. *www.ample.co* [online]. Dostupné z: https://www.ample.co/blog/api-driven-or-git-based-figuring-out-the-right-cms-for-you

DENYSOV, Artem, 2021. The 2021 Web Almanac: Jamstack. *almanac.httparchive.org* [online] [vid. 2024-03-15]. Dostupné z: https://almanac.httparchive.org/en/2021/jamstack

FACEBOOK, 2019. facebook/react. *GitHub* [online]. Dostupné z: https://github.com/facebook/react

FRAIN, Ben, 2015. Architect CSS and scale CSS with the ECSS CSS methodology. *ecss.benfrain.com* [online] [vid. 2025-03-12]. Dostupné z: https://ecss.benfrain.com/

GATSBY, 2024a. GatsbyJS | The Framework for Frontend Developers. *Gatsby* [online] [vid. 2024-03-15]. Dostupné z: https://www.gatsbyjs.com/front-end-framework/

GATSBY, 2024b. Overview of the Gatsby Build Process. *Gatsby* [online] [vid. 2024-03-10]. Dostupné z: https://www.gatsbyjs.com/docs/conceptual/overview-of-the-gatsby-build-process/

GATSBY, 2024c. Rendering Options. *Gatsby* [online] [vid. 2024-03-15]. Dostupné z: https://www.gatsbyjs.com/docs/conceptual/rendering-options/

GRUDL, David , 2024. Caching. *Nette Documentation* [online] [vid. 2024-03-15]. Dostupné z: https://doc.nette.org/en/caching

HEAVY.AI, 2024. What is Server-Side Rendering? Definition and FAQs | HEAVY.AI. *www.heavy.ai* [online]. Dostupné z: https://www.heavy.ai/technical-glossary/server-side-rendering

HTTPS ://DAVIDGRUDL.COM, David Grudl;, 2024. Cache. *Nette Dokumentace* [online]. Dostupné z: https://doc.nette.org/cs/caching

IKAHEIMO, Max , 2022. Next.js vs. Gatsby: A full comparison. *Ikius - Make your vision happen with a dedicated Nordic tech partner* [online] [vid. 2024-03-15]. Dostupné z: https://ikius.com/blog/next-js-vs-gatsby#id0

JAMNADASS, Reah , 2024. CDN: what is edge CDN and virtual CDN (vCDN)? *STL Partners* [online]. Dostupné z: https://stlpartners.com/articles/edge-computing/cdn-what-is-edge-cdn-and-virtual-cdn-vcdn/

JARTARGHAR, Harish A., Girish Rao SALANKE, Ashok Kumar A.R, Sharvani G.S a Shivakumar DALALI, 2022. React Apps with Server-Side Rendering: Next.js. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* [online]. **14**(4), 25–29. Dostupné z: https://jtec.utem.edu.my/jtec/article/view/6192

KALUŽA, Marin, Marijana KALANJ a Bernard VUKELIĆ, 2019. A comparison of back-end
frameworks for web application development. *Zbornik Veleučilišta u Rijeci* [online]. **7**(1), 317–332. Dostupné z: doi:https://doi.org/10.31784/zvr.7.1.10

KANG, DongYoon , 2020. Performance Comparison of SWC and Babel – SWC. *swc.rs* [online] [vid. 2024-03-01]. Dostupné z: https://swc.rs/blog/perf-swc-vs-babel

MDN WEB DOCS, 2018. HTML: HyperText Markup Language. *MDN Web Docs* [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

META OPEN SOURCE, 2024. Writing Markup with JSX – React. *react.dev* [online]. Dostupné z: https://react.dev/learn/writing-markup-with-jsx

MONUS, Anna, 2022. What is server-side rendering and how does it improve site speed? | DebugBear. *www.debugbear.com* [online]. Dostupné z: https://www.debugbear.com/blog/server-side-rendering

MOZILLA, 2019. Introduction. *MDN Web Docs* [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

MOZILLA, 2024a. Compile - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *developer.mozilla.org* [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Compile

MOZILLA, 2024b. Understanding latency - Web Performance | MDN. *developer.mozilla.org* [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Performance/Understanding_latency

MOZILLA CORPORATION, 2019. Introduction to the DOM. *MDN Web Docs* [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

NEXT.JS, 2024a. Data Fetching: getServerSideProps | Next.js. *nextjs.org* [online]. Dostupné z: https://nextjs.org/docs/pages/building-your-application/data-fetching/get-server-side-props

NEXT.JS, 2024b. Learn | Next.js. *nextjs.org* [online] [vid. 2024-03-07]. Dostupné z: https://nextjs.org/learn-pages-router/basics/data-fetching/two-forms

NEXT.JS, 2024c. Rendering: Static Site Generation (SSG) | Next.js. *nextjs.org* [online]. Dostupné z: https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation

OLSSON, Niclas a Nicklas OCKELBERG, 2020. *Performance, Modularity and Usability, a Comparison of JavaScript Frameworks* [online] [vid. 2024-01-05]. Dostupné z: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-272109

ORACLE, 2022. What is a database? *www.oracle.com* [online]. Dostupné z: https://www.oracle.com/database/what-is-database/

OSMANI, Addy a Jason MILLER, 2024. Rendering on the Web | Articles. *web.dev* [online]. Dostupné z: https://web.dev/articles/rendering-on-the-web

RIVA, Michele, 2022. *Real-World Next.js*. B.m.: Packt Publishing Ltd. ISBN 9781801079877.

SEBESTA, Robert W., 2016. *Concepts of programming languages*. Boston: Pearson. ISBN 9780133943023.

SHAH, Ritesh , 2022. Build Powerful Web Applications With Next.js & Strapi. *WEBO Digital* [online]. Dostupné z: https://webo.digital/blog/build-robust-web-apps-with-nextjs-and-strapi/

SHENOY, Aravind a Anirudh PRABHU, 2018. Choosing Lightweight Frameworks for Intuitive Web Design. In: [online]. s. 1–14. ISBN 9781484233986. Dostupné z: doi:https://doi.org/10.1007/9781484233993_1

STACK OVERFLOW, 2023. Stack Overflow Developer Survey 2023. *Stack Overflow* [online]. Dostupné z: https://survey.stackoverflow.co/2023/

SVEN CASTELEYN, Peter DOLOG a CESARE PAUTASSO, 2016. *Current Trends in Web Engineering*. B.m.: Springer. ISBN 9783319469638.

THAKKAR, Mohit, 2020. *Building React Apps with Server-Side Rendering*. B.m.: Apress. ISBN 9781484258699.

THE GRAPHQL FOUNDATION, 2012. GraphQL: A query language for APIs. *Graphql.org* [online]. Dostupné z: https://graphql.org/

WAPPALYZER, 2024a. Wappalyzer. *www.wappalyzer.com* [online] [vid. 2024-03-15]. Dostupné z: https://www.wappalyzer.com/technologies/javascript-frameworks/gatsby/

WAPPALYZER, 2024b. Websites using Next.js - Wappalyzer. *www.wappalyzer.com* [online] [vid. 2024-03-15]. Dostupné z: https://www.wappalyzer.com/technologies/javascript-frameworks/next-js/

WATERS, Frank, 1996. *AIX Performance Tuning* [online]. B.m.: Prentice Hall [vid. 2024-03-11]. Dostupné z: https://sites.ualberta.ca/dept/chemeng/AIX-43/share/man/info/C/a_doc_lib/aixbman/prftungd/toc.htm

YASUL, Matthew , 2023. NextJS Bundle Management 101. *Matthew Yasul's Blog* [online] [vid. 2024-03-15]. Dostupné z: https://www.mattyasul.com/blog/nextjs-bundle-management/

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

SSG – Static Site Generation
SSR – Server-Side Rendering
js – JavaScript
CSR – Client-Side Rendering
HTML – Hypertext Markup Language
CSS – Cascading Style Sheets

Přílohy

Příloha 1……ssg_gatsby.zip
Příloha 2 ……ssg_nextjs.zip
Příloha 3 ……ssr_gatsby.zip
Příloha 4 ……ssr_nextjs.zip