Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Problematika systémů pro správu obsahu postavených na technologii GraphQL

Diplomová práce

Bc. Vojtěch Koutský

Školitel: PhDr. Milan Novák, Ph.D.

České Budějovice 2024

Bc. Koutský, Vojtěch, 2024: Problematika systémů pro správu obsahu postavených na technologii GraphQL. [Issues of content management systems built on GraphQL technology. Mgr. Thesis, In Czech.] – 49 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato magisterská práce se zaměřuje na problematiku systémů pro správu obsahu postavených na technologii GraphQL. Popisuje technologii GraphQL, která je klíčová pro návrh a realizaci Headless systémů. Na základě provedených rešerší daných technologií navrhuje a implementuje vlastní řešení. Vytvořené řešení obsahuje Headless CMS s napojením na GraphQL API a frontend napsaný v React.js.

Klíčová slova

GraphQL, Headless CMS, API, React.js

Annotation

This master thesis focuses on the issue of content management systems built on GraphQL technology. It describes GraphQL technology, which is key to the design and implementation of Headless systems. Based on the research of the given technologies, it proposes and implements its own solution. The created solution includes Headless CMS with connection to GraphQL API and frontend written in React.js.

Key words

GraphQL, Headless CMS, API, React.js

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval(a) pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 12.4.2024 _____

Bc. Vojtěch Koutský

Poděkování

Chtěl bych tímto velice poděkovat svému vedoucímu diplomové práce PhDr. Milanovi Novákovi, Ph.D. za cenné rady a připomínky, všestrannou pomoc při vypracování této diplomové práce a trpělivost. Dále bych chtěl poděkovat své rodině a přátelům za podporu během mého studia.

Obsah

Seznam zkratek

API – Application Programming Interface

JSON – JavaScript Object Notation

REST – Representational state transfer

HTTP - Hypertext Transfer Protocol

SEO - Search engine optimization

ID - Identifier

CRUD - Create, Read, Update, and Delete

UI - User interface

CSS - Cascading Style Sheets

NPM - Node.js package manager

NPX - Node Package eXecute

1 Úvod

V dnešní době, kdy digitalizace a vstup do online prostředí je součástí každodenního života a podnikání, je správa obsahu klíčovým prvkem úspěchu online projektů. Společnosti, organizace i firmy všech velikostí se snaží poskytnout uživatelům relevantní obsah prostřednictvím různých kanálů a zařízení. V této souvislosti představují technologie GraphQL a koncept Headless CMS revoluční přístup k řízení a distribuci dat a obsahu.

Tato práce se zabývá zkoumáním a definováním možností technologie GraphQL, které jsou klíčové pro návrh a realizaci Headless systémů. Headless CMS díky tomu přináší revoluční přístup ke správě obsahu, kdy obsahová vrstva je kompletně oddělená od prezentace, což umožňuje větší flexibilitu a možnost využití na různých zařízeních a platformách.

Diplomová práce je rozdělena do dvou hlavních částí. První část se zaměřuje na teoretickou analýzu technologií, včetně GraphQL a konceptu Headless CMS. V této části je popisováno fungování jednotlivých technologií, jejich klíčové vlastnosti a možnosti efektivního využívání ve spojení s Headless CMS. Druhá část se zabývá praktickou implementací řešení GraphQL pro Headless CMS a je zde podrobně popsán vývoj a implementace tohoto řešení.

Cílem této práce je poskytnout ucelený pohled na možnosti a výhody technologie GraphQL v kontextu Headless CMS a navrhnout praktické řešení, které demonstruje efektivní využití této technologie v praxi.

1.1 Cíle práce

Cílem práce je zjistit a definovat možnosti technologie GraphQL, které jsou klíčové pro návrh a realizaci Headless systémů. Systém pro správu obsahu bude fungovat tak, že jeho rozhraní bude využívat optimální dotazování a zefektivnění práce například pomocí API.

2 Motivace

Motivací pro tuto práci je rostoucí potřeba efektivní správy obsahu a jeho prezentace v různých kontextech, která vzniká v důsledku dynamického prostředí digitálního trhu a narůstajících očekávání uživatelů. S nástupem nových typů zařízení, platforem a kanálů pro interakci s obsahem se stává stále obtížnější udržovat konzistentní a kvalitní uživatelskou zkušenost napříč všemi používanými kanály.

Headless architektura, která umožňuje oddělit obsah od jeho prezentace, nabízí řešení těchto výzev. Tento nástroj pomáhá vývojářům a obsahovým tvůrcům flexibilně pracovat s daty, aniž by byli vázáni na konkrétní druh jejich prezentace. Výsledkem je, že stejný obsah může být snadno a efektivně využíván napříč různými kanály a zařízeními, jako jsou například webové stránky, mobilní aplikace, digitální billboardy a další.

V tomto kontextu hraje technologie GraphQL klíčovou roli. Její schopnost poskytovat flexibilní a výkonné rozhraní pro dotazování dat přináší do Headless architektury další úroveň dynamiky a efektivity. GraphQL umožňuje klientům přesně specifikovat data, která požadují, a zároveň je získat ve strukturované podobě. Dochází tak k maximálně efektivnímu využití dostupného šířkového pásma, protože při použití toho rozhraní nejsou přenášena nadbytečná data.

Největší výhodou kombinace Headless architektury a technologie GraphQL je větší flexibilita, agilní vývoj a vysoká efektivita práce s obsahem. Tyto faktory jsou klíčové pro zajištění konkurenceschopnosti a udržení kvality služeb v dnešním dynamicky se rozvíjejícím digitálním prostředí. Jejich studium a studium jejich aplikace je proto velmi relevantní a důležité.

3 Výzkumné úkoly/problémy

Pro úspěšné řešení problematiky efektivní správy obsahu a jeho prezentace napříč různými kanály a zařízeními je nutné provést důkladnou analýzu současných řešení a technologií v této oblasti.

Analýza je v této práci zaměřena na Headless architektury a zahrnuje zejména analýzu přínosů tohoto nástroje a možnost jeho efektivní integrace do existujících systémů. Vedle toho je zároveň důležité vnímat potřeby a očekávání uživatelů v oblasti správy obsahu a jeho prezentace.

Pozornost je primárně věnována technologii GraphQL, která nabízí flexibilní a výkonné rozhraní pro dotazování dat. Je provedena detailní analýza vlastností a výhod této technologie a její porovnání s tradičními přístupy k práci s daty. Snahou je identifikovat, v jakých situacích je vhodnější využití rozhraní GraphQL a v jakých naopak konvenční správy obsahu a prezentace.

Celkově je provádění analýzy současných možných řešení a technologií klíčové pro úspěšné navržení a realizaci systému pro správu obsahu. Získané poznatky umožní popsat optimální postupy a nástroje pro dosažení maximální efektivity práce a s tím spojené vyšší konkurenceschopnosti a kvality.

4 Headless CMS

Headless CMS slouží pro moderním přístup ke správě obsahu. Umožňuje oddělit strukturu a správu obsahu od prezentace na různých zařízeních a kanálech. Na rozdíl od tradičních CMS, které používají kombinaci obsahu a prezentace, Headless CMS dovoluje vytvářet a spravovat obsah nezávisle na konkrétním prostředí.

Klíčovou rolí Headless CMS je poskytovat obsah jako službu, kde data jsou vybírána prostřednictvím API, a to zejména v podobě RESTful nebo GraphQL. Využitím těchto nástrojů mají vývojáři a tvůrci obsahu větší kontrolu a přehled nad tím, jak je obsah spravován a prezentován.

Headless CMS je výhodné zvolit v situacích, kdy je obsah distribuován na různé platformy a zařízení s různými potřebami prezentace, jako jsou například webové stránky, mobilní aplikace či digitální obrazovky. Díky jeho schopnosti poskytovat obsah přes API, začíná být Headless CMS využíván v dnešním digitálním prostředí stále častěji.

Další důležitou funkcionalitou je možnost tvořit nový či upravovat předchozí obsah, a to bez toho, aniž by bylo nutné vědět, jakým způsobem bude tento obsah prezentován uživatelům.

Headless CMS využívá model "content-as-a-service". Tento model je cíleně orientovaný na poskytování služby v podobě obsahu, a to prostřednictvím rozhraní API. API je nejčastěji realizováno pomocí technologií RESTful nebo GraphQL. Tyto technologie zaručí efektivní distribuci obsahu a jeho další prezentaci na různých frontendových technologiích jako například React.js, Vue.js a podobně.

RESTful API používá standardní HTTP metody (GET, POST, PUT, DELETE) pro manipulaci s daty. GraphQL na rozdíl od RESTful API umožňuje klientům získat přesně ta data, která potřebují, jedním dotazem. Oba typy rozhraní jsou podrobněji popsány v dalších kapitolách.

Další nedílnou součástí celého systému je zabezpečení, které je v dnešní době absolutní nutností. Z důvodu ochrany citlivého obsahu a dat musí být systém pečlivě zabezpečen. Zabezpečení se řeší zpravidla pomocí autentizace, například přes API klíče, či OAuth nebo tokeny. Pomocí těchto mechanismů ověřujeme, zda má klient práva pro přístup k obsahu [1; 2; 3].

Struktura a formát dat jsou většinou definovány ve formátu JSON. JSON je jedním z formátů pro výměnu dat a jak již název napovídá, je úzce spojen s jazykem JavaScript. Na tomto jazyku je však nezávislý a může být využíván v kombinaci s mnoha jinými programovacími jazyky. Jeho výhodou je univerzálnost a efektivita přenosu dat, zároveň je velmi intuitivní a snadno čitelný pro vývojáře. Tento formát používají všechny konvenční API nejčastěji.

API je používáno k účinné komunikaci mezi serverovou částí CMS a klienty, kteří požadují data. Klienty jsou v těchto případech webové stránky, mobilní aplikace, či jiné systémy, které potřebují přistupovat k obsahu.

Z důvodu rychlého vývoje v této oblasti je nutné paralelně řešit také otázku aktualizací a nových verzí Headless CMS a API. Tyto aktualizace jsou postupně nasazovány pomocí verzí. Vytváření nových verzí řeší problém udržení kompatibility a nehrozí tak riziko narušení již existujících systémů [4; 5; 6].

4.1 Důvod vzniku Headless CMS

První zmínka o konceptu Headless CMS se objevila v začátku dvacátého prvního století, kdy se začala vytvářet vyšší poptávka a potřeba spravovat obsah odděleně od jeho prezentace. Tradiční CMS byly často navrženy tak, aby kombinovaly obsah a jeho prezentaci, což mělo za následek velké množství omezení [7].

Se zvyšující se dostupností pokročilých frontendových frameworků a technologií došlo k většímu důrazu na efektivní distribuci obsahu. Nástup technologií jako AJAX a RESTful API umožnil vývojářům efektivnější a dynamičtější přístup k obsahu bez nutnosti kompletní obnovy stránky.

S nástupem GraphQL v druhé dekádě dvacátého prvního století se otevřely nové možnosti pro efektivní manipulaci s obsahem. GraphQL pomáhal klientům specifikovat, jaká data chtějí, a vytvářet tak účinnější komunikaci mezi klienty a serverem.

V průběhu poslední dekády začal Headless CMS získávat širší uplatnění a stále více organizací se přiklání k tomuto přístupu ke správě obsahu. Tato rostoucí popularita je způsobena potřebou vytvářet obsah, který je snadno dostupný a použitelný na různých platformách a zařízeních.

Historie Headless CMS je spojena s postupným technologickým vývojem a rostoucí potřebou distribuce obsahu v digitálním prostředí. Tento vývoj dokazuje stále vyšší důležitost tohoto konceptu v moderním webovém a digitálním prostředí.

Headless CMS hraje klíčovou roli v současném digitálním prostředí, kde rychlost, flexibilita a efektivní distribuce obsahu mají zásadní význam. Z těchto důvodů organizace stále více přecházejí k Headless CMS při plánování svých digitálních strategií.

S narůstající poptávkou po obsahu na různých zařízeních a kanálech, Headless CMS představuje základní stavební kámen pro tvorbu a správu obsahu, který lze snadno implementovat do širšího spektra digitálních aplikací [6; 7].

5 Dostupné Headless CMS systémy

Pro srovnání bylo vybráno pět Headless CMS systémů, které patří mezi nejrozšířenější. Jsou to Strapi, Directus, Burdy, Ghost, Squidex. Všechny tyto systémy jsou dostupné zdarma, je však důležité dodat, že zdarma dostupné verze těchto systémů mohou mít v některých případech určitá omezení, například v počtu uživatelů, záznamů nebo funkcí. Toto srovnání proběhlo z důvodu výběru vhodného Headless CMS pro praktickou část práce. Při výběru Headless CMS je důležité zvážit potřeby projektu a zkontrolovat podrobnosti o dostupných funkcích a omezeních u jednotlivých poskytovatelů CMS [3; 8].

5.1 Strapi

Moderní open-source Headless CMS, který nabízí široké možnosti tvorby, správy a publikace digitálního obsahu pro webové stránky, mobilní aplikace a další digitální projekty.

Jednou z klíčových vlastností Strapi je jeho schopnost definovat a spravovat vlastní datové modely. Uživatelé mohou jednoduše vytvářet struktury dat, definovat vztahy mezi nimi a provádět různé operace, jako je například přidávání, úpravy nebo mazání obsahu. Díky tomu je možné vytvářet složité obsahové stránky, které splňují specifické požadavky projektu.

Další důležitou vlastností Strapi je jeho plná podpora pro práci s API. Platforma poskytuje robustní API, které umožňuje komunikaci s jakýmkoli frontendovým frameworkem nebo technologií. To znamená, že uživatelé mohou snadno integrovat obsah publikovaný v Strapi do svých webových stránek, mobilních aplikací nebo jiných digitálních produktů.

Strapi dále nabízí široké možnosti správy uživatelů a jejich oprávnění. Administrátoři mohou definovat různé uživatelské role a nastavit přístupová práva pro jednotlivé části systému. To umožňuje efektivní řízení přístupu k obsahu a funkcím platformy.

Strapi je vysoce flexibilní a rozšiřitelné díky open-source licenci a podpoře pro vlastní pluginy a rozšíření. To umožňuje uživatelům přizpůsobit Strapi podle svých specifických potřeb a požadavků projektu [9].

5.2 Directus

Open-source Headless CMS s důrazem na flexibilitu a možnost vlastního přizpůsobení. Jeho architektura je plně headless, to umožňuje stejně jako u Strapi a dalších Headless CMS oddělení obsahu od prezentace.

Další z výhod je také srovnatelná se Strapi, jedná se o poskytnutí úplné kontroly nad datovými modely pro aktuální uživatele. Uživatelé mohou snadno definovat vlastní datové struktury a vztahy mezi nimi pomocí intuitivního uživatelského rozhraní. To umožňuje vytvářet složité datové modely a obsahové stránky přesně podle potřeb projektu.

Důležitou funkcí Directus je také jeho plně rozšiřitelná architektura. Uživatelé mají možnost vytvářet vlastní pluginy a rozšíření, která umožňují rozšiřovat funkcionalitu Directus podle konkrétních požadavků projektu.

Velkou výhodou tohoto systému je jeho vhodnost pro širokou škálu projektů, od jednoduchých webových stránek po komplexní aplikace.

Co se týče správy uživatelů a jejich oprávnění, nabízí Directus možnost definovat různé uživatelské role a nastavit přístupová práva pro jednotlivé části systému [10].

5.3 Burdy

Inovativní Headless CMS, který si klade za cíl nabídnout uživatelům jednoduchost, flexibilitu a velké možnosti přizpůsobení. Jeho architektura je stejně jako u předchozích systémů plně headless.

Jedním z hlavních rysů Burdy je ho uživatelsky přívětivé rozhraní, které je navrženo tak, aby bylo maximálně intuitivní a snadno ovladatelné, a to i pro uživatele bez hlubších odborných znalostí.

Stejně jako předchozí systémy umožňuje Burdy efektivně řídit přístupy k obsahu a funkcím a plně tak zajišťovat bezpečnost dat [8].

5.4 Ghost

Headless CMS, který se specializuje na tvorbu a správu obsahu pro blogy, publikace a média. Jeho hlavním cílem je poskytnout uživatelům prostředí optimalizované pro psaní a publikaci obsahu, s důrazem na jednoduchost, rychlost a efektivitu. Ghost je navržen tak, aby umožňoval tvůrcům obsahu soustředit se na psaní a tvorbu obsahu, aniž by byli rušeni složitými technickými detaily.

Jedním z hlavních rysů Ghost je jeho minimalistické a elegantní uživatelské rozhraní, které je zaměřeno na snadnou navigaci a pohodlné psaní obsahu. Uživatelé mohou psát články v čistém a přehledném editoru, který umožňuje formátování textu, přidávání obrázků a médií a další úpravy obsahu s minimálním úsilím.

Ghost se zaměřuje na poskytování špičkového výkonu a rychlosti. Díky své minimalistické povaze a optimalizaci kódu dosahuje Ghost vynikajících výsledků v rychlosti načítání stránek, což je klíčové pro uživatelský zážitek a SEO.

Další významnou vlastností Ghost je jeho plná podpora pro práci s API. Ghost je také známý svou širokou škálou možností přizpůsobení a rozšíření. Uživatelé mohou využívat různé šablony a motivy pro úpravu vzhledu svého blogu nebo publikace a mohou si také vytvářet vlastní pluginy a rozšíření umožňující přizpůsobení funkcí Ghost podle svých potřeb [11].

5.5 Squidex

Open-source Headless CMS a správce obsahu, který se zaměřuje na flexibilitu, škálovatelnost a snadnou správu digitálního obsahu pro webové stránky, mobilní aplikace a další digitální nástroje. Jeho hlavním cílem je umožnit uživatelům vytvářet, spravovat a publikovat obsah prostřednictvím intuitivního uživatelského rozhraní a silného API.

Jednou z klíčových vlastností Squidex je jeho schopnost definovat a spravovat flexibilní datové modely. Uživatelé mohou jednoduše definovat vlastní datové struktury a vztahy mezi nimi pomocí přívětivého uživatelského rozhraní, což umožňuje vytvářet složité a dynamické obsahové stránky a aplikace.

Další významnou funkcí Squidex je jeho plná podpora pro práci s API. Squidex dále nabízí široké možnosti správy uživatelů a jejich oprávnění. Administrátoři mají možnost definovat různé uživatelské role a nastavit přístupová práva pro jednotlivé části systému.

Platforma Squidex je také vysoce škálovatelná a robustní, a je ji tedy možné využít pro projekty různých velikostí a rozsahu. Je postavena na moderních technologiích a architektuře, což zajišťuje vysokou dostupnost a spolehlivost [12].

5.6 Srovnání Headless CMS

Po provedení rešerše a srovnání vybraných CMS nelze s určitostí říci, které z Headless CMS je nejlepší. Nejvíce záleží na konkrétních požadavcích a potřebách daného projektu. Každý z těchto Headless CMS má určité výhody a je vhodné jej použít v jiném konkrétním případě. Nicméně je možné rozdělit vybrané Headless CMS alespoň dle hlavních rysů. Strapi je vhodné pro projekty, které potřebují vysokou flexibilitu a přizpůsobitelné datové modely, také pro škálovatelnost, silné API a přívětivé uživatelské rozhraní. Jedna z největších výhod je také silná komunita uživatelů a široké využití. Zvolit Directus je dobré v případě, že je potřeba maximální kontrola nad daty a flexibilita. K dalším výhodám patří také možnost jednoduchého rozšíření a přizpůsobení přímo na míru danému projektu. Burdy je nejčastěji používán v situacích, kdy je potřeba jednoduché a snadno použitelné řešení, například pro menší nebo středně velké projekty. Výhodou je totiž jeho minimalistické řešení, které je ideální volbou pro uživatele, kteří nemají příliš široké odborné znalosti. Ghost se také používá nejčastěji v situacích a projektech, kde je potřeba efektivní a jednoduché prostředí pro psaní a správu obsahu. Je minimalisticky řešený a využívá se zejména pro menší a středně velké projekty. Silnými stránkami Squidex jsou zejména silné API a škálovatelnost. Je možné jej použít se na projekty různých velikostí a rozsahů. Níže v Tabulce 1 jsou shrnuty základní parametry vybraných druhů Headless CMS.

	Strapi	Directus	Burdy	Ghost	Squidex
Licence	MIT	GPLv3	MIT	MIT	MIT
Cloudové služby	Ne	Ano	Ne	Ano	Ano
Komerční podpora	Ano	Ano	Ano	Ano	Ano
Placené plány	Ano	Ano	Ano	Ano	Ano
Cenová dostupnost	Základní verze zdarma	Otevřený zdrojový kód	Bezplatný tarif pro malé projekty	Základní verze zdarma	Základní verze zdarma
Podpora	Podpora od vývojářského týmu, návody	Podpora od vývojářského týmu, návody	Podpora od vývojářského týmu, návody	Komunita, návody	Podpora
Dokumentace	Ano	Ano	Ano	Ano	Ano
CLI	Ano	Ano	Ano	Ano	Ne
Plugin systém	Ano	Ano	Částečný	Ne	Ano
GraphQL	Ano	Ano	Ne	Ne	Ne
Rest	Ano	Ano	Ano	Ano	Ano
API	Ano	Ano	Ano	Částečně	Ano
Fulltextové vyhledávání	Ano	Ano	Ano	Částečně	Ano
Flexibilita	Vysoká	Střední	Střední (Nové CMS možnost vylepšení)	Nízká	Vysoká
Uživatelské rozhraní	Přehledné, intuitivní	Minimalistické, čisté	Přehledné, intuitivní	Jednoduché, minimalistické	Přehledné, intuitivní
Zabezpečení	Vysoké	Vysoké	Vysoké	Základní	Střední

Tabulka 1 - Funkcionality porovnávaných systémů

6 Používané technologie v Headless CMS

6.1 REST

RESTful API je architektonický styl pro aplikační programové rozhraní, které používá HTTP požadavky pro přístup a použití dat. Tato data lze použít k datovým typům GET, PUT, POST a DELETE, používaných při čtení, aktualizacích, vytváření a mazání operací týkajících se zdrojů. Je založené na přenosu reprezentativního stavu, což je architektonický styl a přístup ke komunikaci často používaný při vývoji webových služeb. Technologie REST je obecně používanější než jiné podobné technologie. Hlavním důvodem je to, že využívá menší šířku pásma, takže je vhodnější pro efektivní používání internetu. RESTful API lze také vytvořit pomocí programovacích jazyků, jako je JavaScript nebo Python [13; 14].

Základní charakteristiky technologie REST:

- klient-server – klient a server jsou na sobě navzájem nezávislí,
- bezestavový – server stav klienta průběžně nezaznamenává,
- ukládání do mezipaměti – server ukládaná část nejčastěji používaných dat do mezipaměti a zároveň tato data označuje,
- jednotné rozhraní – server poskytuje klientovi výstupy jednotným a předvídatelným způsobem,
- vícevrstvý systém – prostředníci mezi klientem a serverem neovlivňují jejich chování,
- kód na vyžádání – server klientovi může přidat další funkce prostřednictvím zaslaného kódu, který může tento klient spustit [15].

Hlavní výhody REST:

- jednoduchost a srozumitelnost – tato technologie je založena na standardních HTTP metodách (GET, POST, PUT, DELETE), které usnadňují pochopení a implementaci,

- nezávislost na platformě – neobjevuje se zde závislost na žádné specifické platformě nebo technologii. REST je podporován většinou moderních programovacích jazyků a frameworků,
- škálovatelnost – každá služba je nezávislá a může být distribuována na různé servery,
- stavová nezávislost – všechny požadavky obsahují potřebné informace pro zpracování, to umožňuje serverům být stavově nezávislí,
- cachování – může výrazně zlepšit výkon a efektivitu aplikace [13; 15].

Hlavní nevýhody REST:

- nadměrná komplexita – v některých případech může být implementace příliš složitá a náročná na porozumění,
- přenášená data – často přenášena nadbytečná data, tento problém může vést k nadměrnému zatížení sítě,
- omezení na HTTP metody – používá pouze omezený počet HTTP metod (GET, POST, PUT, DELETE), což může být v některých případech omezující,
- nedostatečná podpora pro real-time aktualizace – není vhodná pro implementaci real-time aktualizací, jako je například sledování změn v reálném čase,
- nízká interoperabilita – v některých případech může interoperabilita mezi různými systémy a službami být obtížná kvůli různým interpretacím specifikace REST [13; 15].

6.2 GraphQL

GraphQL je dotazovací jazyk pro API a prostředí pro plnění dotazů. GraphQL zajišťuje úplný a srozumitelný popis dat v API a dává klientům možnost přesně specifikovat své potřeby. Usnadňuje vývoj API v průběhu času a využívá výkonné vývojářské nástroje.

Po odeslání dotazu GraphQL do API jsou vrácena přesně ta data, která byla požadována. Dotazy GraphQL vždy vracejí předvídatelné výsledky. Aplikace využívající GraphQL jsou rychlé a stabilní, protože řídí pouze data, která získávají.

Dotazy GraphQL přistupují nejen k vlastnostem jednoho zdroje, ale také plynule sledují odkazy mezi nimi. Zatímco typická REST API vyžadují načítání z více adres URL, GraphQL API získávají všechna data, která konkrétní aplikace potřebuje, v jediném požadavku. Aplikace využívající GraphQL mohou být rychlé i při pomalém připojení k internetu.

Rozhraní API GraphQL jsou organizována podle typů a polí, nikoli podle koncových bodů. Získávají tak přístup ke všem datům z jednoho koncového bodu.

Umožňuje přesně zjistit, jaká data je možné vyžádat z API, aniž by bylo nutné opustit editor. Zároveň upozorní na potenciální problémy před odesláním dotazu za využití vylepšené inteligence kódu. GraphQL usnadňuje vytváření výkonných nástrojů s pomocím typového systému API.

GraphQL vytváří jednotné API napříč celou aplikací, aniž by byl omezen konkrétním úložištěm. Poskytuje funkce pro každé pole v typovém systému a poté je volá s optimální souběžností.

Existuje několik různých přístupů k tvorbě API, nejznámější konkurencí je však REST. Při porovnání právě s REST architekturou je jednou z velkých výhod GraphQL to, že je v něm možné předem specifikovat konkrétní data, která se dostávají ze serverové části na klientskou. Uživatel tak neobdrží data, o která si explicitně neřekl, a tím se eliminuje přenos nepotřebných dat. GraphQL je silně typované. V případě, že je potřeba pracovat s konkrétní datovou entitou, musí být pro ni definovaný typ. Pokud by byl zaslán požadavek na neznámý typ, volání končí neúspěšně.

GraphQL vychází z matematické teorie grafů, kde se dle této teorie grafy skládají z vrcholů a hran. Pro GraphQL to pak prakticky znamená, že veškerá data musí být uspořádána do datových typů a každý tento typ v teorii grafů znázorňuje jeden vrchol. Hrany spojující tyto vrcholy pak vyjadřují vztahy mezi těmito datovými typy [16; 17].

Hlavní výhody GraphQL:

- flexibilita v dotazování – umožňuje klientům specifikovat přesně, jaká data chtějí získat, čímž eliminuje přenos nadbytečných dat a zlepšuje výkon aplikace,
- jednoznačné schéma dat – používá jednoznačné schéma dat, které definuje strukturu dat poskytovaných API. To usnadňuje porozumění a integraci s API,

- zpětná kompatibilita – umožňuje přidávat nová pole nebo typy do schématu bez zásahu do existujících klientů. Díky tomu je zajištěna zpětná kompatibilita a snižuje se tak riziko porušení API,
- jednoduchá integrace více zdrojů dat – umožňuje snadnou integraci dat z různých zdrojů, jako jsou databáze, externí API nebo soubory,
- podpora real-time aktualizací – díky možnosti použití Subscriptions je možné implementovat real-time aktualizace, umožňuje tedy klientům sledovat změny dat v reálném čase [18; 19; 20].

Hlavní nevýhody GraphQL:

- komplexita implementace – může být náročný na implementaci, zejména pokud jde o správu schématu dat a optimalizaci výkonu,
- závislost na správné implementaci serveru – efektivní použití GraphQL vyžaduje správnou implementaci serverové části, to může být v některých případech obtížné,
- potenciální zvýšená náročnost na serverové zdroje – pokud není GraphQL správně optimalizován, může docházet k nadměrné zátěži na serverové zdroje kvůli velkému množství dotazů,
- omezená podpora pro cachování – cachování dat v GraphQL může být složité, protože každý dotaz může mít jiné požadavky na data,
- omezená podpora pro starší prohlížeče – některé funkce GraphQL mohou být omezeny nebo nedostupné v starších prohlížečích, což může limitovat jejich použitelnost v určitých prostředích [18; 19; 20].

6.3 Rozdíl mezi GraphQL a REST

Typickou charakteristikou pro REST je velké množství výstupů. I pro jednodušší operaci je tedy často zapotřebí poslat na server více požadavků, zatímco GraphQL má výstup pouze jeden. Současně se vždy musí požadovaná data přesně definovat. Eliminuje se tak nejen množství příchozích dat, ale i množství požadavků, které je potřeba na server odeslat.

V GraphQL tedy neexistuje způsob, jakým se obecně zeptat na všechno, jako je typické například u dotazovacího jazyka SQL (př.: SELECT * FROM table;). Pokud se chceme pomocí GraphQL dotázat na všechno, musíme vyjmenovat všechny atributy.

Mezi další významné rozdíly patří i to, že GraphQL je nezávislé na protokolu, naopak REST je určen výhradně pro práci s daty pomocí metod odpovídajících HTTP protokolu. Z toho rozdílu pak vyplývá i rozdílná práce s chybovými hláškami. Zatímco u REST architektury obsahuje odpověď příslušný HTTP status, na který lze následně reagovat, tak u GraphQL je přístup kvůli nezávislosti na protokolu odlišný. Nejpoužívanější přístup, který se objevuje u většiny knihoven, obsahuje jen dva statusy, a to status 400 pro chybný požadavek a status 200 pro vše ostatní. Pokud dojde k nějaké chybě, tak popis této chyby se následně objeví standardně v JSON odpovědi od serveru v části pojmenované „error“ [18; 19].

6.4 Proč GraphQL

Jak již bylo popsáno výše, GraphQL umožňuje požádat přímo o data, které uživatel potřebuje, a to v jednom dotazu. Dále je možné požadovat vlastní jedinečné specifikace dat. Tyto vlastnosti snižují náročnost a šetří výkon serveru, který tak nemusí procházet velké množství dotazů. Další nespornou výhodou je jedna verze grafu, která dává přehled, jak se graf v průběhu času mění. Registr se chová podobně jako Git a stává se centrálním místem, kde je možné sledovat aktuální stav grafu. Není tak potřeba řešit nejnovější aktuální verzi API, jako například u REST. Dalším důvodem, proč zvolit GraphQL, je již přesně zpracovaná dokumentace. Existuje pak také několik nástrojů na prozkoumání grafu například, Apollo Studio nebo GraphQL Playground [20].

6.5 Schéma a typy

V GraphQL jsou tři základní objektové typy – Query, Mutation a Subscription. Tyto základní typy jsou vstupy do GraphQL. Typ Query je povinný a je v něm popsán seznam všech dotazů, které je možné prostřednictvím GraphQL volat. Pokud by typ Query nebyl vůbec definovaný nebo by neměl definovanou alespoň jednu položku, API nedokáže vrátit žádná data a není možné jej správně používat. Výhodou rozdělení na tyto tři typy je

především jednoduchost a rozdělení funkcionalit, které vedou k lepší bezpečnosti a škálovatelnosti. Každý z těchto objektových typů má jinou funkci. Query slouží pro čtení dat, Mutation pro zápis a editaci a Subscription pro asynchronní sledování změn.

Dále se dají definovat takzvané objektové custom typy, kterými lze definovat návratovou hodnotu jednotlivých položek (queries, mutations, subscriptions nebo položek jiných custom typů). Tyto custom typy jsou samozřejmě opakovaně použitelné, takže jejich vhodné definování dokáže velmi zpřehlednit a zkrátit výsledný kód.

Návratová hodnota položky může být buďto některý z definovaných custom typů, nebo některý ze základních typů, tzv. skalární typ. Defaultními skalárními typy jsou pro GraphQL Integer, Float, String, Boolean a ID (unikátní identifikátor reprezentovaný jako String).

Stejně jako je možné definovat objektové custom typy, tak je možné definovat i skalární custom typy. Často používaným skalárním custom typem je například typ „Date“ pro datum. Návratová hodnota může být definována i jako pole skalárních nebo objektových typů. Návratové hodnoty lze ještě omezit tak, že není povoleno vracet hodnotu ”null”. To se definuje přidáním vykřičníku za danou návratovou hodnotu nebo pole hodnot.

Základem GraphQL je schéma, tedy kolekce všech objektových typů a jejich definic. Schématem se v podstatě definuje, jaká data mohou být poslána, přijímána a jaké jsou mezi nimi vazby. Na základě tohoto schématu je pak možné vygenerovat dokumentaci. Schéma je zpravidla definováno na jednom místě (např. pro JavaScript v souboru index.js). Pro rozsáhlejší aplikace může ale schéma narůstat do obrovských rozměrů a stane se tak velmi nepřehledné. Existují proto nástroje, které umožňují rozložit schéma do více souborů [16; 19].

7 Jaký systém a proč byl vybrán pro další práci

Pro praktickou část práce byl vybrán systém Strapi. Hlavním důvodem použití Strapi je jeho schopnost definovat a spravovat vlastní datové modely a také jeho plná podpora pro práci s API. Dále byla vybrána technologie GraphQL. Hlavním důvodem byla vyšší flexibilita a výkon, protože GraphQL pokládá dotazy na specifická data a při správném nastavení a implementaci snižuje zatížení na server. Také je velice výhodné, že nepoužívá pouze HTTP metody. Tato architektura poskytuje benefity především při složitých datových požadavcích nebo když je potřeba efektivně spravovat a integrovat data.

8 Nasazení GraphQL

8.1 Jak nasadit GraphQL

Strapi řeší nasazení GraphQL pomocí GraphQL pluginu. Následně dochází ke konfiguraci schématu a ke zmapování existujících datových modelů na odpovídající typy a pole. Je možné také definovat vlastní resolvery pro lepší operace a transformace dat, které budou prováděny. Po dokončení konfigurace je nutné provést testování a ladění celého API. Ověří se správnost funkcionality API, implementace a výkonu. Pro testování je možné použít různé nástroje pro interaktivní dotazování a správnost odpovědí, například GraphQL Playground. Poté co je provedeno důkladné testování, může být GraphQL API nasazen do produkčního prostředí.

8.2 Popis principu v Headless systémech

U systému Strapi je použití GraphQL vyřešeno nativně pomocí GraphQL pluginu, který umožňuje provádět dotazy a mutace. Výsledky lze poté filtrovat, třídit a stránkovat. Odpovědi jsou sumarizovány v GraphQL API. Dotazy jsou používány pro načítání dat bez jejich úpravy. Pro zjednodušení a automatizaci vytvoření GraphQL schéma používá Strapi Shadow CRUD, který na základě modelů generuje definice typů, dotazy a mutace. Pro dotazy u každého modelu jsou pak napodobeny základní operace CRUD (find many, findOne, create, update, delete).

Další funkce, které lze provádět pomocí GraphQL pluginu u Strapi, jsou mutace, filtrování a řazení. Mutace se používají pro úpravu dat a vytváření relací. Dotazy mohou obsahovat parametr filters, který pomáhá filtrovat pouze obsah, který je v danou chvíli potřeba. U dotazů můžeme použít také řazení. Data, která dostaneme z dotazu, je možné řadit dle parametrů, které si zvolíme, například sestupně či vzestupně. Poslední parametr, který můžeme využít u dotazů, je stránkování, které vypíše pouze určitý množství obsahu na danou stránku. U Directus, Burdy a Ghost systémů je implementace možná pomocí rozšíření nebo doplňků. U systému Squidex je součástí nativního API [5].

8.3 Výhody GraphQL

První výhodou GraphQL je bezesporu to, že klienti mohou přesně vymezit, jaká data požadují. To eliminuje získávání nadbytečných dat. Jeden GraphQL výstup umožňuje klientům dostat všechna požadovaná data v rámci jednoho dotazu.

Další výhodou je předem daný datový model. To znamená, že chyby v datové struktuře mohou být odhaleny již při jeho návrhu. Namísto více výstupů, jak je obvyklé u RESTu, GraphQL používá pouze jedno spojení, což zjednodušuje jeho správu a konfiguraci. Tím, že klienti získávají pouze potřebná data, dochází ke značné úspoře šířky pásma.

Klienti mohou efektivněji pracovat s API bez potřeby čekat na úpravy serverové strany [12].

8.4 Nevýhody GraphQL

Nevýhodou této technologie je potenciální složitost dotazů. Komplexní dotazy jsou často náchylné k chybám nebo mohou způsobit zvýšenou zátěž na serveru.

Také implementace GraphQL může být složitější než u jednodušších REST API, zejména pro menší projekty nebo jednoduché použití.

Při vývoji může být náročnější ladit chyby v GraphQL dotazech a mutacích.

Při nevhodném použití může vést ke složitosti, která není vždy nutná pro jednoduché aplikace nebo menší týmy.

Při nevhodné konfiguraci může dojít k bezpečnostním problémům např. únik citlivých dat [12].

8.5 Proč využít GraphQL?

GraphQL poskytuje ve srovnání s REST API mnoho výhod. Je vhodné pro nové moderní aplikace, které vyžadují flexibilitu a výkonné rozhraní pro získávání dat. Mezi jeho hlavní výhody patří flexibilita v dotazech. Umožňuje klientům získat přesná specifikovaná data, což vede k efektivní komunikaci mezi serverem a klientem. Další výhodou je snadná integrace s frontendem. Je vhodný k použití s frontendovými technologiemi jako je React,

Angular nebo Vue.js. Klienti mohou požadovat data přímo ve formátu, který odpovídá jejich komponentám. To usnadňuje práci vývojářům a snižuje čas potřebný k implementaci nových funkcí. Pro GraphQL se také v průběhu jeho vývoje vytvořila přehledná a kompletní dokumentace schématu dat, dostupných dotazů a mutací, které lze využít. Výhodou je také škálovatelnost a výkon, který je možné optimalizovat přesně cílenými dotazy [14].

9 Návrh a implementace vlastního řešení

Tato kapitola popisuje návrh a implementaci vlastního řešení, které používá pro správu obsahu Headless CMS a GraphQL. Bude zde popsán logický rámec a scénáře užití, v další části pak bude specifikována architektura řešení, přiblíženy jednotlivé technologie, které byly použity. Dále bude popsána implementace jednotlivých funkčních celků, které toto řešení nabízí, její dokumentace a způsob testování.

9.1 Abstraktní popis řešení

Na základě výběru Headless CMS systém Strapi, který podporuje GraphQL, budou vytvořeny přístupové údaje pro komunikaci s API, definujeme GraphQL schéma, dotazy, mutace apod. Dále dojde ke konfiguraci frontend aplikace, která bude používat React.js. Následovat bude nastavení klienta pro připojení k GraphQL API pro provádění dotazů a mutací v aplikaci a vytvoří se React.js komponenty, zobrazující data získaná z GraphQL API. Následně budou vytvořeny dodatečné funkcionality, například filtry, řazení apod. dle potřeby. Před nasazením aplikace na server proběhne důkladné testování a ladění. Po úspěšném testování bude možné aplikaci nasadit na vhodné hostingové služby a server.

9.2 Co realizací vznikne?

Realizací vznikne webová stránka, která bude pro správu dat používat Headless CMS a dotazovací jazyk GraphQL pro API. Na frontendu webové stránky bude použit React.js. Tato webová stránka bude obsahovat seznam filmů, kategorií a herců.

9.3 Výběr vhodných technologií pro provoz Headless systémů

Narozdíl od běžně používaných CMS, které integrují vše v jednom systému, Headless CMS systém byl vybrán kvůli oddělení obsahu od prezentace. Dalším důležitým důvodem bylo navržení tohoto systému pro jednoduché poskytování obsahu pomocí API. Technologie GraphQL byla zvolena z důvodu flexibility v dotazování a specifikace přesných dat, které jsou požadovány. Knihovna React.js byla vybrána pro tvorbu uživatelského rozhraní

z důvodu rozsáhlé aktivní komunity, která tuto knihovnu používá, komponentové architektury a jednosměrného datového toku.

10 Analýza

10.1 Metodika vývoje

Pro vývoj byla použita agilní metodika vývoje Scrum, která se běžně využívá pro práci v týmu, nicméně byla mírně upravena pro potřeby této práce. Práce byla rozdělena do čtyř částí:

- plánování,
- implementace,
- testování.

10.2 Funkční požadavky

V předchozích kapitolách byly provedeny rešerše a analýzy, při kterých byly identifikovány základní funkční požadavky pro realizaci vlastního řešení. Funkční požadavky jsou vypsány v Tabulce 2.

Kód	**Funkční požadavek**
FP1	Implementace Strapi s GraphQL API
FP2	Vytvoření datových modelů
FP3	Autentizace a autorizace
FP4	Frontend v Reactu
FP5	Podpora pro responsivní design
FP6	Integrace s GraphQL API
FP7	Optimalizace výkonu
FP8	Testování
FP9	Ladění
FP10	Dokumentace
FP11	Nasazení na server

Tabulka 2 - Funkční požadavky

10.3 Nefunkční požadavky

Nefunkční požadavky vychází hlavně z potřeb a identifikovaných slabin analyzovaného řešení, které jsou popsány v Tabulce 3.

Kód	Nefunkční požadavky
NP1	Zajištění vysokého výkonu
NP2	Možnost škálovat API
NP3	Zajištění spolehlivého provozu
NP4	Zabezpečení datových toků
NP5	Zajištění vysoké dostupnosti
NP6	Poskytnutí stabilního výsledky s dlouhodobou podporou a aktualizacemi

Tabulka 3 - Nefunkční požadavky

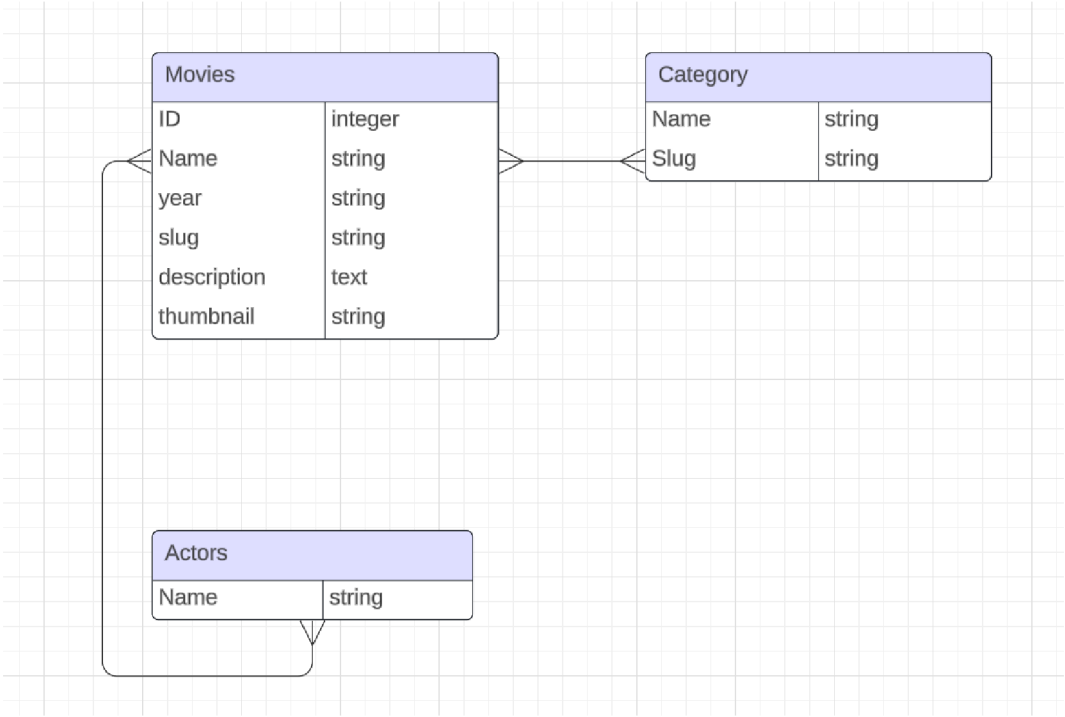
10.4 Logický rámec

Cíl	**Objektivně ověřitelné ukazatele**	**Prostředky ověření**	**Předpoklady**
Snazší, efektivnější a rychlejší způsob dotazování pro API a prostředí pro plnění těchto dotazů se stávajícími daty	Snížení složitosti dotazů a tím i náročnosti pro výkon serveru při pracování s velkým objemem dat	Statistiky odezvy serveru a webové stránky	
Účel	**Objektivně ověřitelné ukazatele**	**Prostředky ověření**	**Předpoklady**
Možnost dotazování přímo na určitá a specifická data, bez potřeby dostávat i okolní nepotřebná data	V další části budou představeny tyto dotazy, které vrací specifická data	Dokumentace projektu, která popisuje způsob, jak se ptát na specifická data	Projekt je dostupný online a má uživatelské rozhraní. Vzájemná integrace API a výstupu uživatelsky definovaných dat
Výstupy	**Objektivně ověřitelné ukazatele**	**Prostředky ověření**	**Předpoklady**
Webová stránka s obsahem dat načtených přes API z Headless CMS	Online repositář v GitHub databázi	Databáze Github.com	Vytvoření postupu pro tvorbu webové stránky s napojením na GraphQL a Headless CMS Open source licence Dokumentace
Činnosti	**Zdroje**	**Časový rámec**	**Předpoklady**
Definice klíčových vlastností, Tvorba Headless CMS s příslušnými pluginy, Napojení GraphQL API, Tvorba fontendu pomocí knihovny React, Testování	Online dokumentace vybraného Headless CMS. Online dokumentace k GraphQL API. Online dokumentace knihovny React. Server a hosting. Počítač s přístupem k internetu	01/2024 - 04/2024	Kvalitní analýza a definice klíčových vlastností Popis práce odpovídající akademickým standardům

Tabulka 4 - Logický rámec projektu

10.5 Datový model

Na Obrázku 1 níže je popsán datový model aplikace, podle něhož byla vytvořena struktura praktické části diplomové práce.



Obrázek 1 Datový model

11 Scénáře použití

11.1 Přidat nový záznam do Collection type

Nový záznam byl přidán vyplněním formuláře u daného collection type v administraci Strapi.

Základní scénář:

1. Uživatel vyplňuje všechny potřebné vstupy v collection type.
2. Uživatel uloží vytvořený záznam.
3. Uživatel publikuje daný záznam.

11.2 Upravit stávající záznam v collection type

Vytvořený záznam byl upraven v collection type v administraci Strapi.

Základní scénář:

1. Uživatel upraví všechny potřebné vstupy v collection type.
2. Uživatel uloží vytvořený záznam.

11.3 Smazat záznamu v collection type

Vytvořený záznam byl smazán z collection type v administraci Strapi.

Základní scénář:

1. Uživatel smaže záznam označením ve výpisu vstupů v collection type.

11.4 Skrytí záznamu v collection type

Vytvořený záznam byl skryt v collection type v administraci Strapi.

Základní scénář:

1. Uživatel přepne vybraný záznam na nepublikovaný ve výpisu vstupů v collection type.

11.5 Vypsat záznamy z administrace Strapi na frontend

Vypsání záznamů bylo provedeno pomocí Query a komponentu, který dané záznamy vypíše.

Základní scénář:

1. Uživatel vytvoří nové Query pro vypsání záznamů z administrace.
2. Uživatel na základě vytvořeného Query vytvoří komponent s danými vstupy z administrace.

11.6 Prohlížení seznamu filmů

Uživatel může procházet seznam dostupných filmů.

Základní scénář:

1. Uživatel na frontendu aplikace může procházet seznam dostupných filmů.

11.7 Prohlížení detailů filmu

Uživatel může zobrazit podrobnosti o konkrétním filmu.

Základní scénář:

1. Uživatel na frontendu aplikace rozklikne vybraný film.
2. Uživateli se zobrazí podrobnosti o vybraném filmu.

11.8 Filtrování filmů podle kategorie

Uživatel může filtrovat filmy podle vybrané kategorie.

Základní scénář:

1. Uživatel na frontendu aplikace vybere konkrétní kategorii.
2. Uživateli se zobrazí výpis filmů v dané kategorii.

11.9 Zobrazení herců daného filmu

Uživatel může zobrazit výpis herců ve vybraném filmu.

Základní scénář:

1. Uživatel na frontendu aplikace vybere konkrétní film.
2. Uživateli se zobrazí podrobnosti o daném filmu včetně výpisu herců.

11.10 Zobrazení kategorií daného filmu

Uživatel může zobrazit výpis kategorií ve vybraném filmu.

Základní scénář

1. Uživatel na frontendu aplikace vybere konkrétní film.
2. Uživateli se zobrazí podrobnosti o daném filmu v četně výpisu kategorií.

12 Specifikace softwarových požadavků

Backendový systém byl vyvinut tak, aby poskytoval GraphQL API pro správu obsahu. Toto API umožnilo efektivní dotazování a manipulaci s daty, což zahrnuje vytváření, editaci a odstraňování obsahu. Důraz byl kladen především na flexibilitu a výkonnost, aby uživatelé mohli pracovat s obsahem prostřednictvím GraphQL dotazů a mutací.

Na frontendové straně byl systém postaven na frameworku React.js. Navržené uživatelské rozhraní poskytovalo pohodlné a intuitivní prostředí pro práci s obsahem. Zobrazení a interakce s obsahem byly dynamicky aktualizovány na základě dat získaných z GraphQL API, což umožnilo uživatelům rychle procházet a upravovat obsah bez nutnosti znovu obnovení celé stránky.

Backendový systém musel také zahrnovat autentizační a autorizační mechanismy pro zabezpečení přístupu k datům. Uživatelé tedy musí projít ověřovacím procesem a mají přístup pouze k datům, pro která mají oprávnění.

Celkově byl systém navržen tak, aby poskytoval efektivní a uživatelsky přívětivé prostředí pro správu obsahu pomocí kombinace moderní technologie GraphQL a frameworku React.js.

12.1 Popis zvolené technologie

Webová stránka byla napsaná v jazyce JavaScript s využitím nadstavby TypeScript pro statickou kontrolu typů. Vykreslení UI bylo vytvořeno pomocí knihovny React.js. Komponenty vytvořené v rámci webové frontend aplikace byly psány funkcionálním způsobem s využitím hooků, které React.js nabízí. Vzhled aplikace byl upraven pomocí CSS.

Pro fungování backendu bylo potřeba Node.js, který se používá jako běžný nástroj pro vývoj aplikací společně se správcem balíčků NPM. Vlastní řešení integrovalo backendovou část postavenou na platformě Strapi. Strapi poskytuje robustní a flexibilní řešení pro správu obsahu a umožňuje použití GraphQL API, které bylo klíčové pro komunikaci mezi backendem a frontendem. Efektivní dotazování a manipulace s daty byla zpracována pomocí GraphQL dotazů a mutací. GraphQL API bylo realizováno pomocí knihovny

Apollo Server, která přináší snadnou integraci GraphQL do Node.js aplikací a dodává robustní nástroje pro definici schématu, validaci dotazů a zpracování požadavků.

Pro správu verzí a kódování byl použit systém git. Repositář je hostován na platformě GitHub.com.

13 Implementace

Implementace u této diplomové práce byla rozdělena do několika částí, které byly potřeba před začátkem programování připravit a nakonfigurovat.

13.1 Instalace Strapi

Na počátku bylo potřeba nainstalovat a zprovoznit Headless CMS, na které bylo poté napojeno GraphQL API. Pro instalaci tohoto systému bylo nutné použít verzi Node.js v18 nebo v20, ostatní verze nejsou pro tento Headless CMS podporované. Pro instalaci bylo dále využito NPM, které je obecně nejvíce používané, a konkrétně jeho nadstavba NPX.

Jako první byl vytvořen Strapi projekt pomocí příkazu:

```
npx create-strapi-app@latest my-project --quickstart
```

Strapi se po instalaci spustilo automaticky. Pokud by nenastalo automatické spuštění, bylo potřeba jej zapnout pomocí příkazu:

```
npm run develop
```

Dále bylo nutné založit administrátorský účet pro správu a tím bylo vše připraveno pro práci se Strapi.

13.2 GraphQL API

Pro správné fungování GraphQL API bylo potřeba nainstalovat GraphQL plugin příkazem

```
npm run strapi install graphql
```

nebo tento plugin nainstalovat přes Strapi Marketplace, který patří mezi dostupné pluginy.

Po jeho instalaci jsme schopni přidat koncový bod, který slouží pro načtení obsahu. Správnost lokální instalace koncového bodu, byla ověřena pomocí adresy uvedené níže, která poskytuje přístup do GraphQL Playground.

```
http://localhost:1337/graphql
```

GraphQL API bylo použito pro výpis dat ze Strapi na frontend, který byl napsán v React.js. Pro použití toho GraphQL API následovala instalace a nastavení Apollo Client pomocí příkazu:

```
npm install @apollo/client graphql
```

13.3 React.js frontend

Následovala instalace aplikace React.js pro vypsání dat na frontend, která je viditelná pro reálné uživatele. Aplikace byla nainstalována příkazem níže:

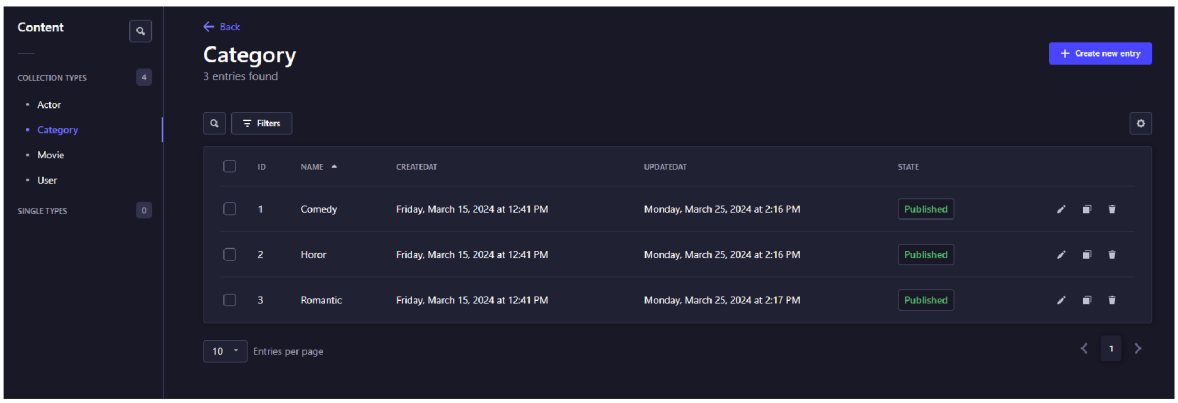
```
npx create-react-app <project-name>
```

Nakonec byla aplikace spuštěna pomocí příkazu:

```
npm start
```

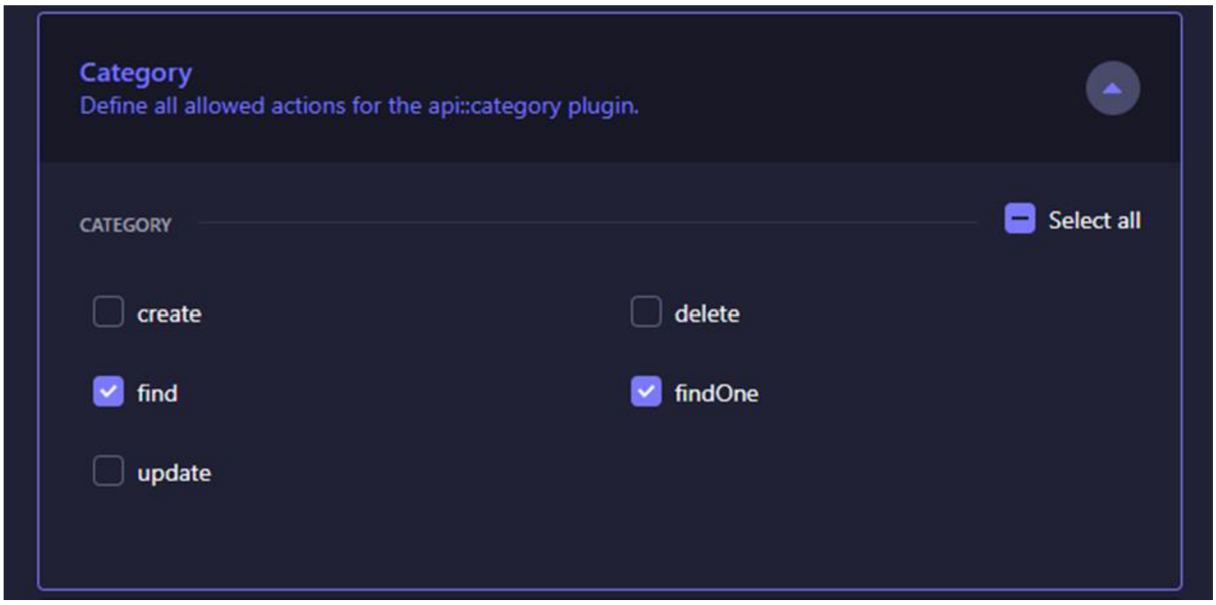
14 Vývoj

Po instalaci všech potřebných technologií následoval vývoj samotné aplikace. Nejdříve byly v administraci Headless CMS Strapi vytvořeny content-types, konkrétně content-types Movies, Actors, Categories. Každý z těchto content-types má své specifické vstupy, jako jsou například name, slug, description, image atd. Pojmenování a jednotlivé vstupy již záleží na potřebě daného projektu a na jeho vývojáři, tudíž se mohou lišit. Na Obrázku 2 je uveden vzorový příklad content-types v systému Strapi.



Obrázek 2 - Strapi content-types

U zvolených content-types bylo nastaveno oprávnění a přístup pro přihlášené uživatele i pro veřejnost. Přihlášeným uživatelům s plným přístupem byly definovány všechny akce, které systém nabízí. Zároveň byl nastaven omezený přístup pro veřejnost a GraphQL API k hledání jednotlivých nebo všech záznamů u daného content-typu viz Obrázek 3 níže.



Obrázek 3 - Strapi povolené akce

Ve frontendové části bylo nastaveno napojení na GraphQL API, které se realizuje přes již nainstalovaný Apollo Client.

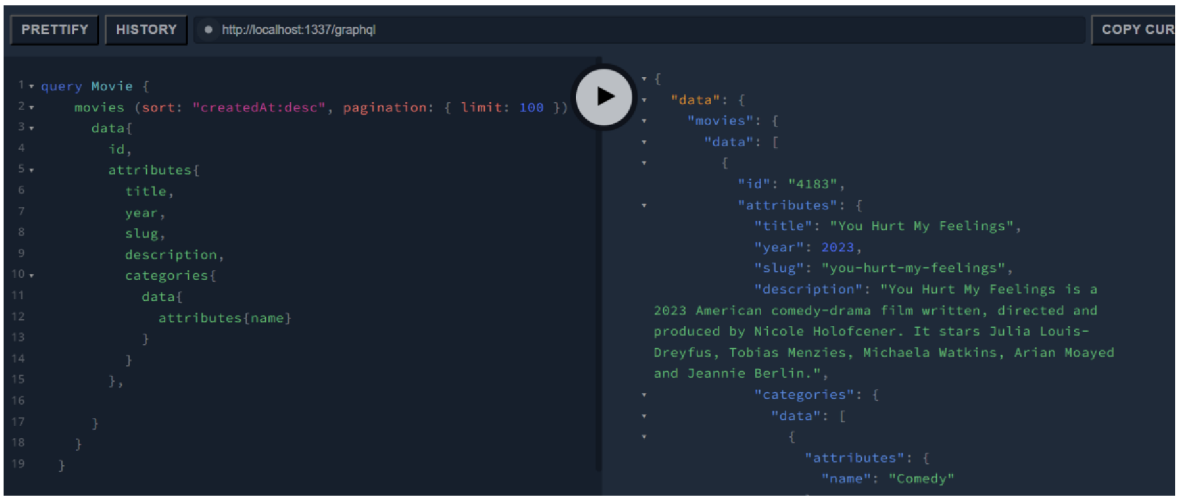
```
const client = new ApolloClient(
  {
    uri: 'http://localhost:1337/graphql',
    cache: new InMemoryCache()
  })
```

Před použitím Queries byl nastaven Apollo Client a současně přidán import Apollo Client a nasazení Apollo Provider. Data ze Strapi backendu na frontend byla vypisována pomocí Queries. Níže je uveden příklad, jak taková Query může vypadat s nutným parametrem pagination. Tento parametr musí být použit z toho důvodu, že Strapi od verze 4 vypisuje defaultně pouze prvních deset záznamů a zadání tohoto parametru umožní vypsání požadovaného počtu záznamů.

```
MOVIES_QUERY = gql`
  query Movies {
    movies (sort: "createdAt:desc", pagination: { limit: 100
}){
      data{
        id,
        attributes{
          title,
          year,
          slug,
          description,
          thumbnail,
          categories{
            data{
              attributes{name}
            }
          }
          actors{
            data{
              attributes{name}
            }
          }
        },

      }
    }
  }
`;
```

Queries byly nejprve ověřeny pomocí GraphQL Playground, který slouží k simulaci výsledků zadaných dotazů. Simulace při úspěšném zadání vrátí požadovaná data ve formátu JSON. Při špatném zadání dotazu se objeví error, který dává zpětnou vazbu k chybách. Na Obrázku 4 je uveden názorný příklad úspěšného dotazu z GraphQL Playground.



Obrázek 4 - GraphQL Playground – vzorová Query

Vytvořená data z Queries byla vypsána přímo na frontend, kde jsou již přístupná uživatelům. Níže je ukázka, jak správně vložit Query do stránky.

```
<Query query={MOVIES_QUERY}>
                {({ data: { movies } }) => {
                  return <Movies movies={movies.data} />;
                }}
</Query>
```

Z důvodu psaní kódu v React.js, který používá komponentovou architekturu, byl definován komponent obsahující list všech záznamů.

```
<div className="movies__list">
        {movies.map((movie) => {
          return (
            <Movie
            movie={movie}
            key={`movie__${movie.attributes.title}`}
            />
            );
          })}
    </div>
```

Ve struktuře je také potřeba další komponent k obdržení konkrétního výpisu, který obsahuje výpis jednoho záznamu.

```
<Link to={`/film/${movie.attributes.slug}`}
className="movies__item">
          <div className="movies__item-content">
          <div className="movies__thumbnail">
            <img src={movie.attributes.thumbnail}
alt={movie.attributes.title} />
          </div>
            <h3>
              {movie.attributes.title}
            </h3>
            <div
className="movies__rating">{movie.attributes.year}</div>
            <div className="movies__description">
            {movie.attributes.description.length > 100 ?
              `${movie.attributes.description.slice(0,
100)}...` :
              movie.attributes.description
            }
            </div>
            <div className="movies__categories">
            <h4>Kategorie:</h4>
            <ul>
              {movie.attributes.categories.data.map(category
=> (
                <li
key={category.attributes.name}>{category.attributes.name}</li
>
              ))}
            </ul>
          </div>

          <div className="movies__actors">
            <h4>Herci:</h4>
            <ul>
              {movie.attributes.actors.data.map(actor => (
                <li
key={actor.attributes.name}>{actor.attributes.name}</li>
```

```
                ))}
              </ul>
            </div>
            </div>
    </Link>
```

Tento postup byl také proveden pro ostatní Queries, které byly potřeba vypsat na frontend a jsou viditelné pro uživatele. Nakonec došlo k úpravě stylů vypsaných komponent. Styly byly upraveny pomocí jazyka CSS.

15 Dokumentace

Dokumentace projektu byla napsána pomocí značkovacího jazyka Markdown. Za použití nástroje Jekyll byly soubory Markdown přetransformovány do HTML stránek, které jsou hostované na GitHub Pages. GitHub Pages je platforma pro hostování statických webových stránek přímo z repositáře na github.com. Dokumentace je psaná v anglickém jazyce.

Dokumentace obsahuje základní dokument README.md, který je zobrazen při návštěvě repositáře na github.com. V tomto repositáři se nachází hlavní informace, které se může uživatel dozvědět o projektu. README.md obsahuje popis, způsob instalace a postup jejího lokální spuštění. Dále je zde odkaz na dokumentaci, která uživatele přesměruje na statické stránky.

Úvodní strana dokumentace obsahuje informace o projektu a motivaci, která vedla k vytvoření. Jsou zde popsány výhody a nevýhody použití GraphQL a Headless CMS.

Následující stránka Installation neboli instalace, popisuje způsob instalace a možnosti lokálního spuštění projektu. Další stránka Queries neboli dotazy obsahuje příklad dotazu, pomocí kterého si lze vyžádat data ze Strapi na frontend aplikace, kde dojde k jejich vypsání. Následující strana Components neboli komponenty zobrazuje příklady použitých komponent v projektu. Poslední strana Pages neboli stránky obsahuje příklad vytvořených stránek, které jsou využívány v projektu.

16 Testování

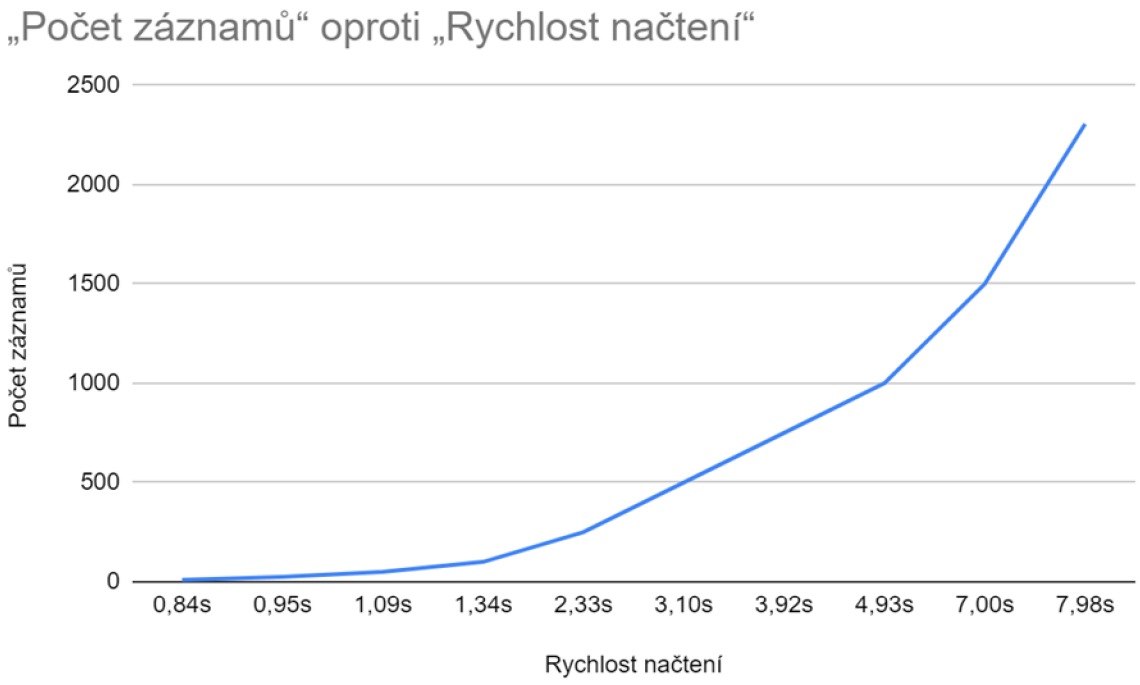
Testování projektu proběhlo pomocí nahrání vzorových dat. Jednalo se o seznam filmů, u kterých byly uvedeny kategorie, herci, popis, odkaz na plakát filmu a rok vydání. Pomocí těchto dat byly následně otestovány jednotlivé části.

Nejdříve proběhlo testování Headless CMS Strapi, kterým bylo potvrzeno, že byla data správně uložena do content-types. Strapi je poté bylo schopné používat a provázat příslušnými relacemi.

Následně bylo otestováno GraphQL API, které data ze Strapi zpracovalo a poskytlo na frontend a následně vypsalo pomocí GraphQL Queries. GraphQL Queries byly otestovány na to, zda předávají přesná data, která byla potřeba vypsat na frontendu. Díky poskytnutí přesných dat byl umožněn nejvyšší možný výkon a rychlost.

Další částí testování bylo ověření správného vypsání všech dat na frontend. I toto testování proběhlo v pořádku a data se z Headless CMS přes GraphQL API správně propsala.

Na závěr se uskutečnilo testování celé aplikace s měnícím se množství dat a porovnána reakce odezvy na poskytnutý objem dat. Testování proběhlo při nastavené rychlosti internetu 10 Mbit/s download a 2 Mbit/s upload a ping 10 sekund. Obrázek 5 zobrazuje výsledek testování bez použití stránkování, a dochází tak k načtení všech dostupných záznamů v jeden čas.



Obrázek 5 - Graf počtu záznamů oproti rychlosti načtení

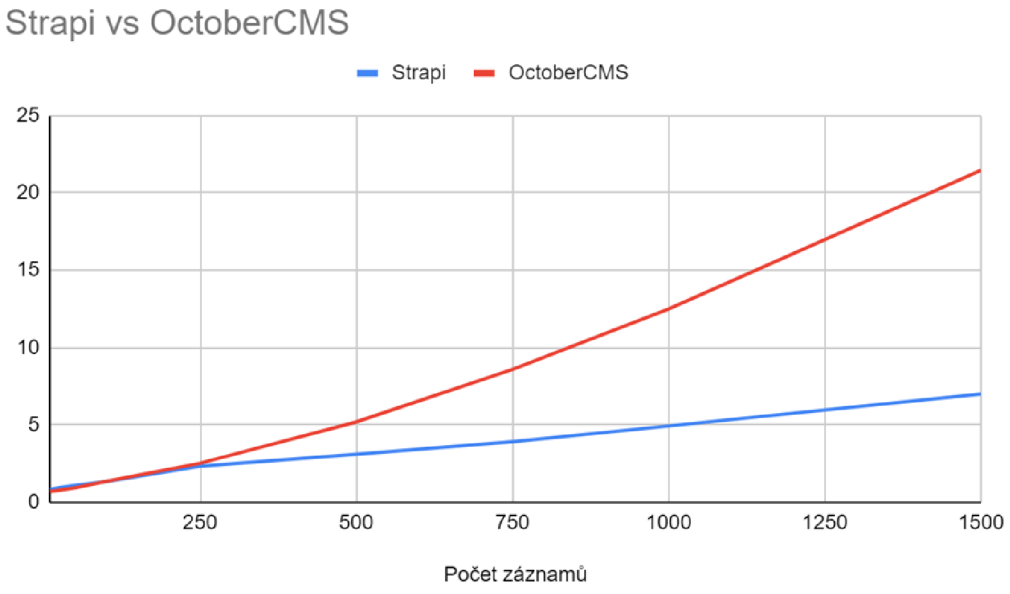
17 Porovnání s konkurenčním řešením

Po provedení testování bylo řešení s použitím GraphQL a Headless CMS porovnáno s tradičním CMS, konkrétně proběhlo srovnání s October CMS.

Srovnání mělo za cíl shrnout budoucím vývojářům možnosti Headless CMS a tradičních CMS a zároveň rozdíly mezi nimi. Výsledek porovnání by měl pomoci vývojářům v rozhodnutí zda je obecně lepší využívat Headless CMS pro různé druhy projektů, anebo jestli tradiční CMS mohou být u některých projektů stále lepší volbou.

Oba systémy byly nastaveny pomocí CSS totožně i z pohledu stylu, aby byla dodržena objektivita měření. Byla zde znovu použita i stejná rychlost internetu jako při testování. Připojení k internetu bylo 10 Mbit/s download a 2 Mbit/s upload a ping 10 sekund.

Obrázek 6 níže porovnává výsledky odezvy z měření Headless CMS Strapi a tradičního řešení pomocí October CMS. Ve svislé ose je zaznamenána rychlost odezvy v sekundách a ve vodorovné ose počet záznamů, které bylo potřeba načíst.



Obrázek 6 - Graf Strapi vs October CMS

Z tohoto grafu vyplývá, že řešení s GraphQL a Headless CMS je vhodné využít především u větších a rozsáhlejších projektů. U typu projektů, které obsahují rozsáhlou strukturu dat a relací, je výrazný rozdíl ve výkonu a načítání dat samotných, proto je s velkou pravděpodobností vhodnější použít technologii GraphQL.

Výrazný rozdíl ve výkonu v závislosti na množství dotazů vyplývá především z optimalizovaného dotazováním GraphQL, které pomocí Queries požaduje konkrétní data, která následně využívá tak, aby nedošlo jako u tradičních řešení k načítání nepotřebných dat.

GraphQL obecně poskytuje větší flexibilitu a rychlost. Bez existujícího GraphQL API, které nemusí být u všech používaných CMS dostupné, může být implementace GraphQL komplikovanější a časově náročnější. Především z tohoto důvodu může být v některých případech použití pomalejších tradičních CMS stále vhodnějším řešením, především díky své jednoduchosti a poskytnutí hotových výstupů, které je možné následně využít. Nasazení tradičních CMS je zároveň rychlejší a jednodušší oproti systému Headless CMS s GraphQL API.

Pokud tedy má klient jednoduché požadavky na správu obsahu a nasazení, CMS se jeví jako vhodnější volba oproti Headless CMS. Jestliže je však kladen důraz na větší flexibilitu a efektivní komunikaci mezi klientem a serverem, je Headless CMS s GraphQL API lepší variantou.

18 Závěr

V rámci této diplomové práce byly zkoumány a shrnuty možnosti technologie GraphQL, které je možné využít, v návrhu a implementaci Headless systémů. Cílem bylo porozumět tomu, jak může být tato technologie využívána k efektivní distribuci obsahu a jak je možné jejím prostřednictvím optimalizovat dotazování dat pomocí API.

Praktická část práce byla zaměřena na implementaci systému pro správu obsahu, který využívá přístup Headless CMS. Pro tuto implementaci bylo využito Strapi Headless CMS, které poskytuje rozhraní pro správu obsahu a komunikaci pomocí GraphQL API. Pro vypracování UI byla použita knihovna React.js, která umožňuje efektivní tvorbu uživatelského rozhraní.

V závěru práce bylo provedeno srovnání tohoto řešení s October CMS. Byly popsány výhody a nevýhody obou přístupů, a to zejména z hlediska flexibility, výkonu, škálovatelnosti a snadné integrace s dalšími technologiemi.

Celkově lze konstatovat, že použití technologie GraphQL ve spojení s Headless CMS a knihovnou React.js přináší řadu výhod pro návrh a realizaci moderních webových aplikací. Toto řešení poskytuje flexibilitu v oddělení obsahu a prezentace, efektivní dotazování dat a integraci s dalšími technologiemi. Zároveň však bylo konstatováno, že volba mezi Headless a tradičním CMS závisí na konkrétních potřebách a požadavcích projektu. Není tedy obecně doporučeno využívat GraphQL ve všech případech, a to především z důvodu složitého nasazení a implementace oproti tradičním CMS.

Aplikace je dostupná v repositáři na adrese https://github.com/vkoutsky. Dokumentace je samostatně dostupná na https://vkoutsky.github.io/vk/.

Seznam literatury

[1] Headless CMS Explained: What is it? Why does it matter? [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://www.storyblok.com/tp/headless-cms-explained

[2] HYGRAPH. Headless CMS: Everything you need to know. Online. Hygraph.com. 2024. Dostupné z: https://hygraph.com/learn/headless-cms. [cit. 2024-03-11].

[3] GARCIA, Veronica a WRITER, Staff. Should your content management system go headless? Online. Theamericangenius. 2015. Dostupné z: https://theamericangenius.com/business-marketing/should-your-content-management-system-go-headless/. [cit. 2024-03-11].

[4] What are the pros and cons of headless CMS vs. other CMS architecture approaches? [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://www.brightspot.com/cms-architecture/headless-cms/headless-cms-pros-and-cons#headless-cms-pros-and-cons

[5] LAMOUREUX, Chris. What Is a Headless CMS? Pros and Cons of Decoupling Your CMS. Online. Veriday. 2019. Dostupné z: https://veriday.com/headless-cms-pros-cons-decoupling/. [cit. 2024-03-11].

[6] DAVIS, Kim. Diversity in talent, Google antitrust allegations: Tuesday's daily brief. Online. Martech. 2021. Dostupné z: https://martech.org/diversity-in-talent-google-antitrust-allegations-tuesdays-daily-brief/. [cit. 2024-03-11].

[7] History of content management systems and rise of headless CMS [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms

[8] Welcome to the Headless CMS Comparison Website. Online. 2024. Dostupné z: https://cms-comparison.io/#/card. [cit. 2024-02-09].

[9] Manage Any Content. Anywhere. Online. 2024. Dostupné z: https://strapi.io/. [cit. 2024-02-09].

[10] The new standard for Headless CMS. [online]. 2024 [cit. 2024-02-09]. Dostupné z: https://directus.io/solutions/headless-cms

[11] Independent technology for modern publishing. Online. 2024. Dostupné z: https://ghost.org/. [cit. 2024-03-02].

[12] How modern companies manage their content. Online. 2024. Dostupné z: https://squidex.io/. [cit. 2024-03-02].

[13] REST API (RESTful API). Online. 2020. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. [cit. 2024-04-01].

[14] GUPTA, Lokesh. What is REST? Online. Restfulapi. 2016. Dostupné z: https://restfulapi.net/. [cit. 2024-03-11].

[15] CRUD vs. REST: What's the Difference? Online. 2020. Dostupné z: https://nordicapis.com/crud-vs-rest-whats-the-difference/. [cit. 2024-04-01].

[16] A query language for your API [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://graphql.org/

[17] ULRICH, H.; KERN, J.; TAS, D.; KOCK-SCHOPPENHAUER, A. K.; ÜCKERT, F. et al. QL4MDR: a GraphQL query language for ISO 11179-based metadata repositories. Online. BMC Medical Informatics and Decision Making. 2019, roč. 19, č. 1. ISSN 1472-6947. Dostupné z: https://doi.org/10.1186/s12911-019-0794-z. [cit. 2024-04-11].

[18] Uncover the Benefits of Using GraphQL with Zenesys [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://www.zenesys.com/drawbacks-and-benefits-of-using-graphql

[19] GraphQL is the better REST [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

[20] Why use GraphQL? [online]. 2020 [cit. 2024-03-11]. Dostupné z: https://www.apollographql.com/blog/why-use-graphql

Seznam obrázků

Seznam tabulek