FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Headless CMS s generováním admin panelu
Student:	Jan Hejzlar
Vedoucí:	Ing. Jan Blizničenko
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je návrh a implementace vlastního centrálního řešení pro správu obsahu v rámci organizace. Řešení bude poskytovat obsah pomocí GraphQL API a bude na základě existujících entit automaticky generovat admin panel pro správu obsahu.

1. Analyzujte potřeby klienta
2. Analyzujte metodiky a technologie vhodné pro tvorbu webových API
3. Proveďte rešerši existujících relevantních řešení
4. Navrhněte vhodné řešení pro implementaci prototypu daného API
5. Implementujte prototyp automaticky generovaného admin panelu a prototyp API, které bude umožňovat přístup k obsahu pro webový projekt
6. Prototyp podrobte vhodnému testování
7. Na základě výsledků testování navrhněte vhodné úpravy a vhodný postup pro nasazení a budoucí vývoj

---

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 29. prosince 2021 v Praze.

Bakalářská práce

HEADLESS CMS S GENEROVÁNÍM ADMIN PANELU

Jan Hejzlar

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Blizničenko
4. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu bakalářské práce, inženýru Janu Blizničenkovi, za cennou zpětnou vazbu a rady při psaní této bakalářské práce. Dále chci poděkovat svým rodičům, kamarádům a přítelkyni Elišce, kteří mě podporovali během studia na vysoké škole.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 4. května 2022 .....

Abstrakt

Tato bakalářská práce se zabývá správou obsahu více webových stránek v rámci jedné organizace. Hlavním cílem je usnadnit tvorbu a změny obsahu u jednotlivých webů z jednoho centrálního systému (admin panelu). Zabývá se analýzou aktuálního stavu a procesů, které jsou spojeny s tvorbou webového obsahu v rámci dané organizace.

V teoretické části popisuje metodiky a technologie, které jsou vhodné pro tvorbu webových API a které jsou důležité v implementační části práce. Součástí práce je i návrh a implementace vlastního řešení daného API, které bude automaticky generovat admin panel na základě existujícího GraphQL API a jeho entit.

Klíčová slova Systém pro správu obsahu, Headless CMS, GraphQL, Webové API, Admin Panel, Monorepo, React, Tailwind CSS, NestJS

Abstract

This bachelor thesis deals with content management of multiple websites within an organization. The main goal is to facilitate the creation and modification of content for individual sites from one central system (admin panel). It deals with the analysis of the current status and processes involved in the creation of web content within a given organization.

In the theoretical part, it describes the methodologies and technologies that are suitable for the creation of web APIs, which are important in the implementation part of the thesis. The work also includes the design and implementation of a custom solution for the API, which will automatically generate an admin panel based on an existing GraphQL API and its entities.

Keywords Content Management System, Headless CMS, GraphQL, Web API, Admin Panel, Monorepo, React, Tailwind CSS, NestJS

Seznam zkratek

ACID	Atomicity, Consistency, Isolation, Durability
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CAP	Consistency, Availability, Partition tolerance
CD	Continuous Deployment
CI	Continuous Integration
CMS	Content Management System
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
NoSQL	Not Only SQL
NPM	Node Package Manager
pnpm	Performant NPM
REST	Representational State Transfer
SEO	Search Engine Optimization
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UI	User Interface
UX	User Experience
UXD	User Experience Design
WYSIWYG	What you see is what you get
XML	Extensible Markup Language

Slovník

Frontend	Část systému, kterou vidí návštěvník stránek.
Backend	Část systému, která se stará o logiku a práci s daty.
Admin Panel	Část systému, která slouží k administraci obsahu uživatelem.
Framework	Softwarová struktura, která slouží jako podpora při programování a vývoji.
Rich Text Editor	Jiný název pro WYSIWYG editor.
Layout	Grafické rozvržení nebo struktura webové stránky.
Rozhraní	Slouží k interakci člověka se systémem.
Uživatel	Člověk používající daný systém k dosažení cíle.

Kapitola 1
Úvod

Společnosti v dnešní době využívají k marketingu svých služeb a produktů celou řadu nástrojů. Ve většině případů patří do tohoto mixu i webové stránky. Králem webu je obsah, který je relevantní pro cílovou skupinu klienta. Kvalitní obsah by měl dát návštěvníkovi webu všechny potřebné informace. V nejlepším případě také zapůsobí na jeho emoce (návštěvník se ztotožní se značkou atd.) a plynule se z něho stává zákazník, který nakoupí (konverze).

Pro kvalitní obsah a spoustu nakupujících zákazníků je důležité, abychom při jeho tvorbě brali v potaz potřeby, motivace, nebo obavy našich zákazníků. Obsah webu je tedy potřeba testovat. Testováním zjišťujeme, zda obsah na webu splňuje svůj účel a zda se ze zákazníků stávají platící zákazníci, ať už přímo (e-shop), či nepřímo (užitečné články na blogu, díky kterým si zákazník objedná naše služby). Na základě výsledků takového testování zjistíme, který obsah je kvalitní a který naopak musíme předělat. Obsah na webu je tedy potřeba vytvářet a upravovat podle potřeb našich zákazníků a následně ho na nich opakovaně testovat.

Ke správě obsahu na webu existuje již mnoho řešení. Nicméně většina z nich nabízí pouze základní funkcionality, které využije většina webů a speciální funkcionality si musí vývojář naprogramovat sám, pokud je vůbec možné existující systém rozšiřovat.

Tato bakalářská práce analyzuje potřeby konkrétního zákazníka v oblasti správy obsahu na jeho webových projektech. Na základě této analýzy si lze udělat relevantní obrázek o potřebách v dané organizaci a prozkoumat již existující řešení na základě požadovaných parametrů na systém pro daný účel. Součástí je i návrh a implementace vlastního řešení pro poskytování webového obsahu na základě požadovaných parametrů a procesů v dané organizaci.

Cílem práce je také pomoci vývojářům daného API v rychlejší implementaci dalších entit. Většina obsahu jsou jednoduché kolekce, které vyžadují pouze definici dané entity a veškeré operace (CRUD) jsou z velké části stejné. Je proto nutné zajistit abstrakci nad těmito operacemi pro zrychlení vývoje. Pomocí GraphQL API bude možné přistupovat k obsahu z jednotlivých webových projektů vývojáři daných webů, kteří mohou s obsahem dále pracovat na straně klienta.

Testování je primárně provedeno přes framework Cypress, který se používá pro end-to-end testování. Důležité je také uživatelské testování, které dává relevantní výsledky o použitelnosti a přívětivosti daného administračního prostředí. Na základě těchto výsledků jsou dále navrženy úpravy pro další vylepšení existujících funkcionalit a rozšíření daného prostředí o další funkcionality. Uživatelské testování je provedeno na cílové skupině, která bude systém využívat. Testovat systém na skupině, která má s tvorbou daného obsahu pramálo společného by nepřineslo relevantní výsledky.

V závěru práce je shrnuto, jaké přínosy má systém oproti předchozímu existujícímu řešení. Přínosy jsou zkoumány jak z hlediska lidí, kteří tvoří daný obsah, tak z hlediska rychlosti dalšího rozšíření vývojáři o další typy obsahu.

Kapitola 2
Analýza

Tato kapitola popisuje potřeby klienta vzhledem ke správě obsahu na různých webech. Dále jsou zde také popsána stávající řešení.

2.1 Potřeby klienta

V Bohemia Interactive a.s. existuje několik webů, které marketingově podporují prodávané hry. Na těchto webech mohou návštěvníci obvykle najít základní informace o hře, blog a někdy i pokročilejší funkcionality - např. units.arma3.com, ylands.com. Dále je potřeba upravovat obsah přímo v dané hře - např. notifikace ohledně plánovaného odstavení serverů, in-game workshop. Veškerý tento obsah je nutné spravovat podle aktuálních potřeb.

2.2 Procesy spojené s tvorbou obsahu

Obsah na web tvoří a vymýšlí Brand manažer, který má zodpovědnost za daný projekt. Vymyslí obsah, např. článek na web, a napíše ho v hrubé formě. Protože je většina konzumentů daného obsahu ze zahraničí, většina textového obsahu musí být napsaná v anglickém jazyce. Tento koncept dostane do ruky ještě copywriter, který celý text reviduje. Poté ho předá zpět Brand manažerovi, který obsah umístí na web přes CMS. Chce-li Brand manažer na web umístit zcela nový obsah (nová stránka, nová kolekce) musí se domluvit s designéry, kteří vytvoří grafický návrh pro nový obsah a následně s vývojáři, kteří grafický návrh naimplementují.

2.3 Stávající řešení

V současnosti neexistuje žádné centrální místo pro správu obsahu. Každý web má vlastní administrační prostředí, kde je každé na jiné technologii a ani neobsahuje stejné funkcionality. Každé administrační prostředí tedy nabízí něco jiného.

Většina projektů je dlouhodobě neudržována, mají zastaralé balíky závislostí a mohou představovat potenciální bezpečnostní riziko. Některé není ani možné spustit lokálně na dev prostředí a jsou natolik zastaralé, že ani nelze stáhnout závislosti.

Na projektech se používaly různé technologie - např. Bohemia.net - Laravel 4, Angular, ylands.com - Vue.js, Node.js. Někde byly prováděny úpravy přímo na produkci. Zdrojové soubory nejsou konzistentní s tím, co se nachází v git repozitáři na gitlabu a na produkci. Projekty jsou tedy velmi obtížně rozšiřitelné. Žádný projekt nemá dokumentaci.

2.4 Vysvětlení použitých pojmů

2.4.1 Headless

Termín *headless* se vztahuje k odstranění závislosti na tom, kde budou data zobrazena. Místo toho pouze uchovává data, která lze použít, kdekoli si vývojář zvolí. Často se používá k popisu systému CMS, do kterého lze zadat obsah. Ten se uchovává a poté se samostatně rozhoduje, kde a jak se tento obsah zobrazí [1].

2.4.2 API Driven

API-driven CMS nebo také API-first CMS či API-based CMS je headless systém CMS, který obsluhuje obsah prostřednictvím rozhraní API. Typicky jde o rozhraní REST API nebo GraphQL API, což umožňuje mít obsah zcela oddělený od prezentační vrstvy. Získáváte tedy svá data a způsob jejich strukturování záleží na vás [2].

2.4.3 Git-based

V systému CMS založeném na systému Git nebo v systému CMS s podporou systému Git se změny nejprve odešlou do Git repozitáře, který pak spustí nové sestavení webu. CMS založené na Gitu fungují jako vrstva nad systémem Git a komunikují přímo s úložištěm Git. V podstatě pracujete se soubory uloženými ve vašem Git repozitáři [2].

2.4.4 Self-hosted

Self-hosting je provozování a údržba webových stránek nebo služeb pomocí soukromého webového serveru namísto používání služby mimo vlastní kontrolu. Jako příklad můžeme uvést osobu, která chce psát svůj vlastní blog. Taková osoba může využít hostované služby blogu, případně si na vlastní server může nainstalovat software pro správu obsahu a hostovat blog samostatně [3].

2.4.5 Cloud

Cloud hosting je řešení, kdy organizace nainstaluje a zpřístupňuje software na serveru přes internet, který vlastní a spravuje organizace třetí strany. Ve většině případů nemá zákazník daného řešení přímý přístup k datům, která jsou softwarem spravována [4].

2.4.6 CI/CD

V softwarovém inženýrství je CI/CD nebo CICD kombinací postupů kontinuální integrace (CI) a kontinuálního doručování nebo kontinuálního nasazování (CD) [5].

CI/CD překlenuje rozdíly mezi vývojovými a produkčními činnostmi a týmy tím, že prosazuje automatizaci při vytváření, testování a nasazování aplikací. Služby CI/CD kompilují inkrementální změny kódu provedené vývojáři a následně je propojují a balí do softwarových produktů [5].

2.5 Rešerše existujících řešení

Všechna řešení jsou API-driven. Git-based řešení nejsou uvažována z důvodu požadavků na rozšiřitelnost a flexibilitu úprav v daném API.

2.5.1 Wordpress

WordPress je svobodný open source redakční publikační systém napsaný v PHP a MySQL. Je vyvíjený pod licencí GNU GPL [6].

Dle oficiálních statistik je používán jako CMS u 43 % webových stránek na světě[9] a překonává tak open source CMS jako Joomla či Drupal, které se drží pod 3 % [6].

Wordpress dle [6] nabízí tyto funkce:

- přizpůsobitelný vzhled
- podpora SEO
- responzivní mobilní stránky
- vysoký výkon
- vysoké zabezpečení
- správa médií
- rozšiřitelnost pomocí pluginů
- knihovna médií

Wordpress je většinou pevně spojen s webem, který spravuje. Jedná se tedy o monolitickou architekturu, kdy je frontend pevně svázán s backendem.

Nicméně Wordpress lze využít i jako Headless CMS. Frontend si tedy data získává pomocí REST API, nebo GraphQL (GraphQL formou pluginu). Ve Wordpressu bohužel nelze nativně spravovat více webů. Je přizpůsoben pro 1 jeden web, což není ideálním řešením pro naše potřeby.

2.5.2 Strapi

Strapi je open-source headless CMS. API je generováno automaticky na základě datového modelu. Nabízí rychlý návrh API, snadnou správu obsahu a zaměřuje se na produktivitu vývojářů [7].

Strapi dle [7] nabízí tyto funkce:

- podpora více databází (SQLite, MongoDB, MySQL, Postgres)
- automaticky generovaná dokumentace
- webhooky
- GraphQL nebo REST
- autentizace a autorizace
- internacionalizace (podpora více jazyků)
- Single-types (Vytvoření jednorázového obsahu - domovská stránka, menu, nastavení SEO)
- přizpůsobitelné API
- knihovna médií

Systém byl napsán v Javascriptu pro běhové prostředí Node.js a je možné si vybrat mezi REST nebo GraphQL [7].

Strapi by mohlo být ideálním řešením. Nabízí REST i GraphQL API. Umožňuje “naklikat” si API, vytvořit kolekce, single typy a další. Nicméně opět je vhodné jen pro správu jednoho webu a neumožňuje obsah dělit do workspaců.

2.5.3 KeystoneJS

KeystoneJS je framework pro správu obsahu, což znamená, že jej lze použít pro vývoj různých webových aplikací pomocí Javascriptu. Díky své modulární architektuře a oddělení různých funkcí je vhodný zejména pro vývoj rozsáhlých aplikací, jako jsou portály, fóra, systémy pro správu obsahu (CMS), eCommerce a webové služby REST [8].

KeystoneJS dle [9] nabízí tyto funkcionality:

- automaticky generovaný CRUD
- konfigurovatelný WYSIWYG Editor
- vlastní role a přístupy
- relace mezi entitami
- filtrování
- rozšiřitelné GraphQL API
- databázové migrace
- vývoj v Typescriptu

2.5.4 PayloadCMS

PayloadCMS je headless CMS zaměřené na vývojáře. Podle [10] nabízí především rozšiřitelnost. Protože se jedná o headless CMS, dá se napojit na web, webové aplikace nebo mobilní aplikaci. Nabízí např. admin panel v Reactu, lokalizaci (více jazyků), nahrávání souborů, autentizaci a autorizaci. Nicméně možnosti rozšíření o vlastní funkcionality jsou nedostačující.

2.5.5 Shrnutí

Veškerá popsaná řešení jsou self-hosted. Cloudová řešení automaticky nepřipadají v úvahu, protože nemáme kontrolu nad ukládanými daty. Dále všechna řešení nabízejí pouze administraci jednoho projektu (nelze spravovat více projektů v jedné administraci).

Žádné existující řešení tedy není optimální pro přímé použití, případně by jeho úprava byla složitá. Z tohoto důvodu jsem se rozhodl navrhnout a naimplementovat vlastní řešení, které je více adekvátní vzhledem k procesům na straně klienta.

Kapitola 3
Návrh

3.1 Funkční požadavky

3.1.1 MoSCoW prioritizace

MoSCoW je metoda pro stanovení priorit, která pomáhá pochopit a řídit priority. Písmena znamenají:

- **M**ust Have
- **S**hould Have
- **C**ould Have
- **W**on't Have this time

Tato technika je použita u následujících funkčních požadavků v upravené podobě, kdy všechny neoznačené požadavky jsou považovány za Must Have.

3.1.2 Kolekce

Kolekce reprezentují obsah na jednotlivých webových stránkách (např. články, často kladené otázky). Každá kolekce v admin panelu bude podporovat operace CRUD (Create, Read, Update, Delete), které jsou základem každé správy dat. Operace přidání, editace a odebrání budou moci provádět jen určení uživatelé. Číst daná data bude moci kdokoliv. Zároveň bude u každé položky možné změnit její stav (může se nacházet ve stavech - Draft, Published).

3.1.3 SEO

U kolekcí, které budou na jednotlivých webech jako stránky, je nutné zadávat následující atributy:

F1 Název (Title) - C

F2 URL slug (automaticky generovaný z názvu - např. Bohemia Recap 2022 → bohemia-recap-2022) - C

F3 Meta Description - C

F4 Open Graph Image (bude možné vybrat z knihovny médií) - C

3.1.4 Knihovna médií

Podporovanými médii budou obrázky a videa. Uživatel bude moci v knihovně médií provádět následující operace:

F5 přidání nového média

F6 odebrání existujícího média

F7 odebrání více médií najednou (Bulk Select) - C

F8 stažení média

F9 vyhledávání médií podle názvu nahraného souboru

F10 filtrace médií - C

F10.1 nepoužívaná média

F10.2 média nahraná aktuálně přihlášeným uživatelem

F11 nastavení atributu alt u obrázků

3.1.5 Rich Text

Rich Text editor bude ukládat data v JSONu (nebude ukládat data v HTML) a bude nabízet následující bloky:

1. odstavec (Paragraph)
2. nadpisy (1. - 3. úroveň)
3. nečíslovaný seznam (Bulleted List)
4. číslovaný seznam (Numbered List)
5. obrázek (Image)
6. citace (Quote)
7. oddělovací svislá čára (Divider)
8. vložený obsah (Embed - např. video z YouTube) - C
9. zdrojový kód (Code block)

Při výběru textu bude možné text dále upravovat. Ve vybrané části bude možné nastavit toto formátování:

1. tučně (Bold)
2. kurzíva (Italic)
3. podtržení (Underline)
4. přeškrtnutí (Strike-through)
5. odkaz (Link - přidá odkaz na vybraný text, který uživatel vloží)

3.2 Nefunkční požadavky

N1 Knihovna médií bude pro ukládání souborů využívat Object Storage.

N2 Backend i frontend bude napsán v TypeScriptu a bude využívat framework NestJS a React.

N3 Použitou databází bude MongoDB.

N4 Veškeré texty v uživatelském rozhraní budou napsány v anglickém jazyce.

3.3 Návrh uživatelského rozhraní

3.3.1 User Experience

Prvním požadavkem na příkladnou uživatelskou zkušenost je přesně uspokojit potřeby zákazníka a to bez zbytečného rozruchu a obtěžování. Dále následuje jednoduchost a elegance, které vytvářejí produkty, které je radost vlastnit a používat. Skutečná uživatelská zkušenost zdaleka nezahrnuje jen to, že zákazníkům poskytneme to, co si přejí, nebo že jim poskytneme funkce z kontrolního seznamu. K dosažení kvalitního uživatelského zážitku v nabídce společnosti musí dojít k bezproblémovému propojení služeb více oborů, včetně inženýrství, marketingu, grafického a průmyslového designu a v našem případě především k designu rozhraní [11].

User Experience tedy zahrnuje všechny aspekty interakce koncového uživatele se společností, jejími službami a produkty. Nejedná se tedy pouze o interakci s grafickým uživatelským rozhraním. Definice User Experience má daleko širší pohled [11].

3.3.2 Co je to uživatelské rozhraní?

"*Grafické uživatelské rozhraní (Graphical User Interface - GUI) je kombinací technologií a prostředků, které umožňují uživateli komunikovat s počítačem a aplikací.*" [12] U webových aplikací se jedná především o rozmístění grafických či textových prvků. Může se dokonce jednat i o 3D objekty, se kterými lze manipulovat. Díky těmto ovládacím prvkům může uživatel interagovat s aplikací a provádět potřebné úkony, ke kterým je aplikace navržena. Mezi tyto úkony patří získávání vstupních dat, jejich následné zpracování a reakce na výsledky daného zpracování. Tyto výsledky jsou následně prezentovány uživateli, většinou prostřednictvím webového prohlížeče [12].

Moderní uživatelská rozhraní webových aplikací velmi často využívají nejnovějších technologií implementovaných do webových prohlížečů. Jedná se především o asynchronní zasílaní požadavků (AJAX) nebo vytvoření oboustranného kanálu pro zasílání zpráv (Websockets). Při použití těchto technologií není nutné přenačtení stránky. Designerům tak umožňují navrhovat interakce s aplikací, které výrazně zpříjemní její používání (UX) [12].

V moderních grafických uživatelských rozhraních je obsah striktně oddělen od jeho grafické interpretace. Při větších úpravách, aktualizacích nebo tvorbě nového GUI pro jiné platformy může být většina aplikace zachována a dojde pouze k rozšíření o další GUI. To vede k úspoře času a ekonomických prostředků [12].

Hlavním rysem dobře navrženého grafického rozhraní je spokojenost uživatelů s používáním aplikace. Především jim pomáhá se splněním jejich cílů [12].

Tato orientace na uživatele, a od ní odvíjející se způsob navrhování aplikace, je označován právě jako User Experience Design (UXD). Dle [12] je při této metodice GUI navrhováno mimo jiné s ohledem na:

- Design – chápán nejen jako vizuální forma, ale i tak, jak navrhovaná aplikace funguje; cílem designu má být co možná nejúčelnější propojení funkční a estetické složky navrhované aplikace.

- Použitelnost – jedna z částí, která vytváří uživatelský prožitek, zaměřující se na odstraňování nedostatků v ovládání a vylepšování na základě uživatelských zkušeností s aplikací.

3.3.3 Obecný proces tvorby GUI

Návrh kvalitního a funkčního GUI webové aplikace není jednoduchou záležitostí a jeho kvalita souvisí také se zkušenostmi samotného návrháře. Návrh grafického uživatelského rozhraní není pouze otázkou vizuálního vzhledu a rozmístění prvků na stránce. Pokud má být aplikace úspěšná a má uživatelům přinést komfortní ovládání spolu se splněním jejich cílů, je potřeba komplexnějšího pohledu na celý proces vývoje, a to už od samého počátku. Finální vzhled jednotlivých komponent je až jedním z posledních stádií [12].

User Experience Design je dle [12] primárně zaměřen na uživatele a jeho zážitek z používání aplikace. Zahrnuje v sobě několik disciplín, jež by měly zajistit následující vlastnosti dobré aplikace:

- Funkčnost a přínos celé aplikace pro uživatele – jedno z hlavních kritérií použitelnosti
- Efektivita – rychlost práce a časová náročnost na vykonání nějakého dílčího úkolu
- Jednoduché a intuitivní rozhraní – přizpůsobení aplikace uživatelům a jejich zvykům
- Jednoduchost zapamatování – ohled na limity a možnosti lidské paměti a ochota uživatelů ji při používání aplikace používat, nenutit uživatele přemýšlet.

Celý vývoj GUI se dá shrnout do několika stádií, které na sebe navazují a staví jedno na druhém. Vývoj začíná v čistě abstraktní rovině a pomalu se stává stále konkrétnějším až po definitivní podobu aplikace a jejích GUI prvků [12].

Celý proces připomíná stavbu víceposchoďové budovy, na jejímž vrcholu je vizuální styl komponent. Pokud však budou nižší patra postavena chybně, je kvalita horního patra (např. designové zpracování jednotlivých prvků) irelevantní a tato nekvalita může mít navíc fatální důsledky na fungování aplikace a spokojenost zákazníků – uživatelů [12].

3.3.4 Základní Style Guide

3.3.4.1 Typografie

Na základě podstaty samotného uživatelského rozhraní a celého projektu bylo zvoleno písmo Inter. Inter je bezpatkové písmo optimalizované pro maximální možnou čitelnost na počítačových obrazovkách. Je ideální volbou pro tvorbu administračních rozhraní.

3.3.4.2 Barvy

Protože budeme na klientovi používat Tailwind CSS, který v základu obsahuje velkou paletu předpřipravených barev, dává tedy smysl tuto paletu využít již při samotném grafickém návrhu.

3.3.4.3 Grid

Pro konzistentní mezery mezi komponentami a velikosti komponent budeme využívat 4 bodový grid.

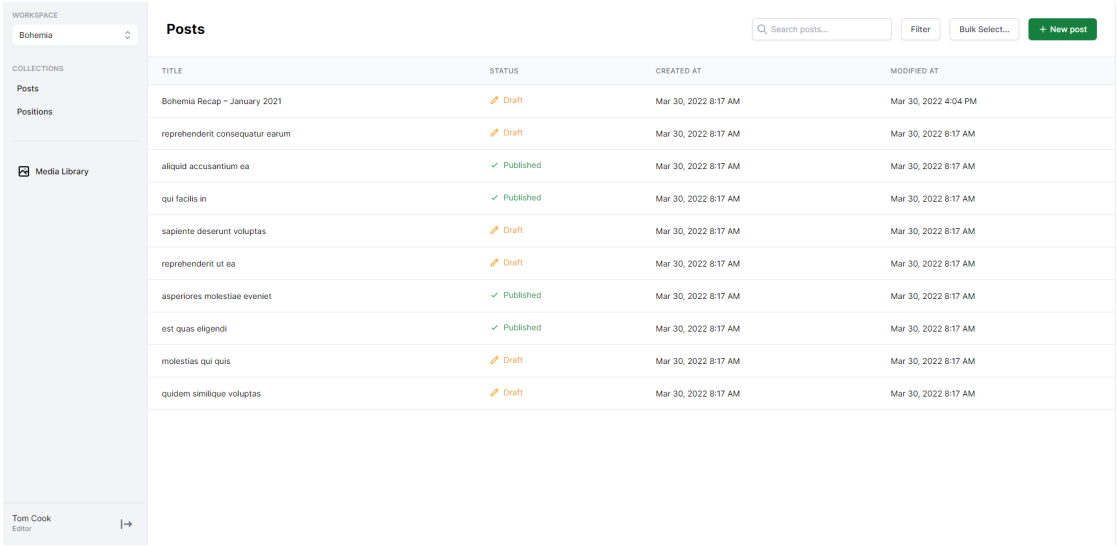
3.3.5 Grafický návrh

3.3.5.1 Kolekce

Výpis a editace záznamů jsou inspirovány výpisem záznamů z Webflow CMS. V levé části se nachází menu, ve kterém je možné se navigovat mezi projekty, kolekcemi a knihovnou médií. Na horním panelu je jasně viditelná primární akce *Přidání záznamu* a další sekundární akce jako hledání, filtrace a hromadný výběr.

Následuje tabulka s výpisem záznamů. Každý řádek tabulky představuje jeden záznam a obsahuje název, stav, datum vytvoření a datum modifikace záznamu. Při kliknutí na řádek se záznamem se dostaneme do detailu záznamu.

Obrázek 3.1 Výpis záznamů z kolekce



V detailu záznamu jsou v horní části zobrazeny základní údaje a možné akce jako je název záznamu, zpětné tlačítko pro návrat do výpisu, stav záznamu a tlačítko pro uložení. Tlačítko pro uložení nabízí také rozjíždějící se menu, kde bude možné změnit stav viditelnosti záznamu. Při změně stavu dojde k uložení záznamu. Pokud u žádné položky nedošlo ke změně, nebude možné záznam uložit, pouze změnit jeho stav, protože provedení takové akce by bylo zbytečné.

U každého záznamu je možné editovat všechny položky. Položky, které jsou povinné, budou tuto skutečnost indikovat pomocí červené hvězdičky (“*”) u popisku dané položky. Pokud došlo ke změně některé z položek, záznam ještě nebyl uložen a uživatel chce z dané obrazovky odejít, je nutné ho upozornit, že provedené změny nejsou uloženy.

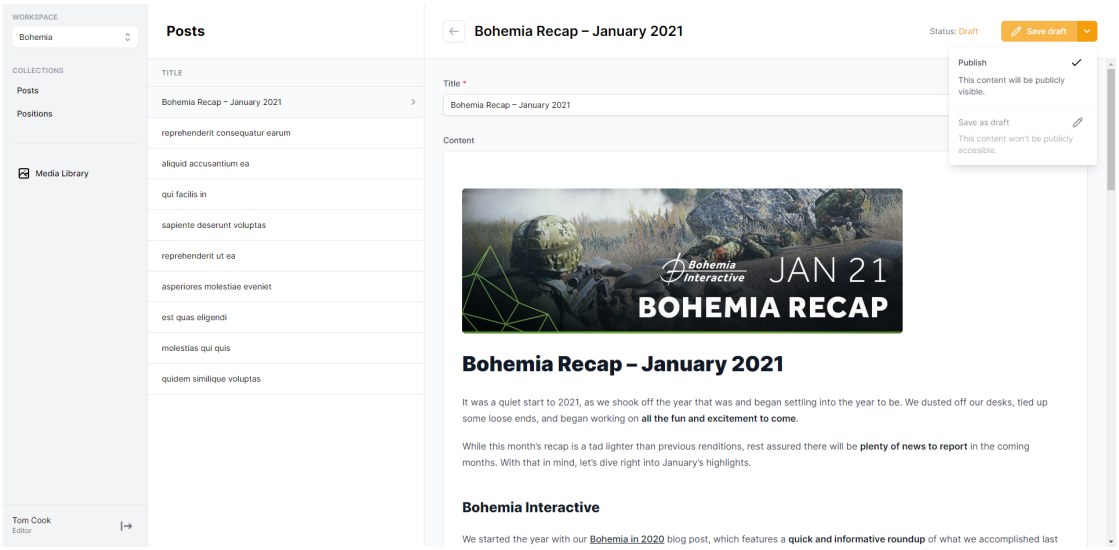
Rich text editor je inspirován aplikací Notion, a to především pro svůj způsob formátování obsahu do různých typů bloků přes příkazy. Tato funkce umožňuje uživateli zůstat u aktuálně formátovaného textu. U dokumentů delšího rozsahu má tedy nesmírnou výhodu oproti ostatním editorům, které mají panel nástrojů v horní části dokumentu.

Ve spodní části detailu je možné nalézt další informace o daném záznamu jako např. ID, datum vytvoření, datum modifikace nebo tlačítko pro smazání záznamu.

3.3.5.2 Knihovna médií

Na základě funkčních požadavků je potřeba přidávat média jako jsou obrázky nebo videa. Přidávání funguje jak pomocí tlačítka “Add media”, které vyvolá okno pro výběr souboru

Obrázek 3.2 Editace záznamu v kolekci



k nahrání, tak pomocí přímého přetažení souboru do okna prohlížeče, přičemž dojde k jeho uložení do knihovny médií.

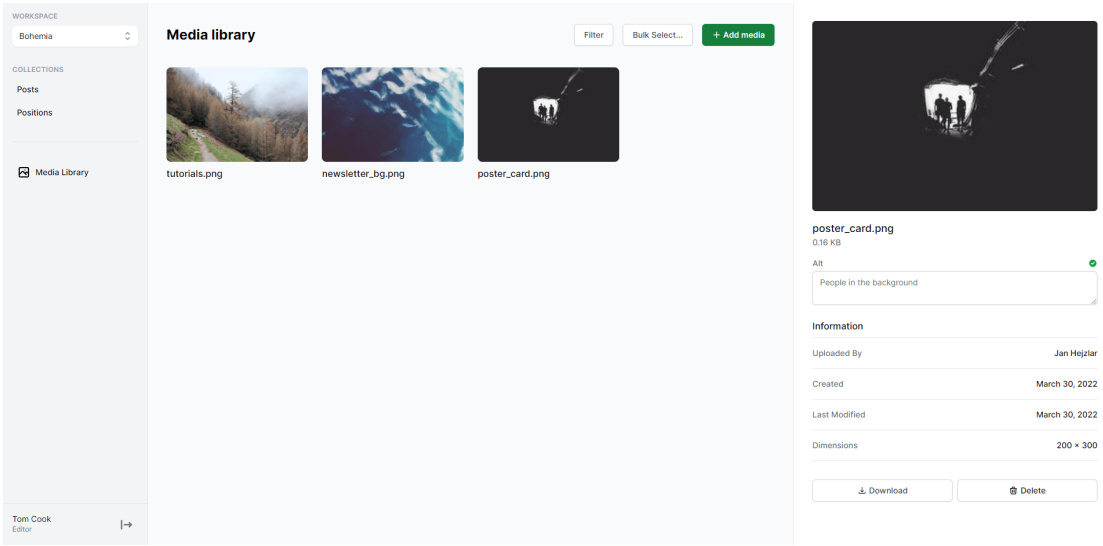
Při nahrávání média se uživateli zobrazí aktuální stav nahrávaného souboru, tedy jak velká část je již nahraná.

Po nahrání si uživatel může zobrazit detail daného média a vidět další užitečné informace. U obrázků je to např. náhled, velikost souboru, alt atribut nebo velikost obrázku v pixelech.

Alt atribut (týká se pouze obrázků) bude možné měnit. K jeho uložení dojde automaticky po “odkliknutí” myší ze zadávacího pole. Při zpracování ukládání se zobrazí ikonka rotujícího kolečka, které ukládání indikuje. Poté co je alt atribut úspěšně uložen, změní se kolečko na ikonku “fajfky”, která indikuje úspěšné uložení.

Dále jsou k dispozici další dvě akce specifikované funkčními požadavky, a to stažení média a smazání média z knihovny.

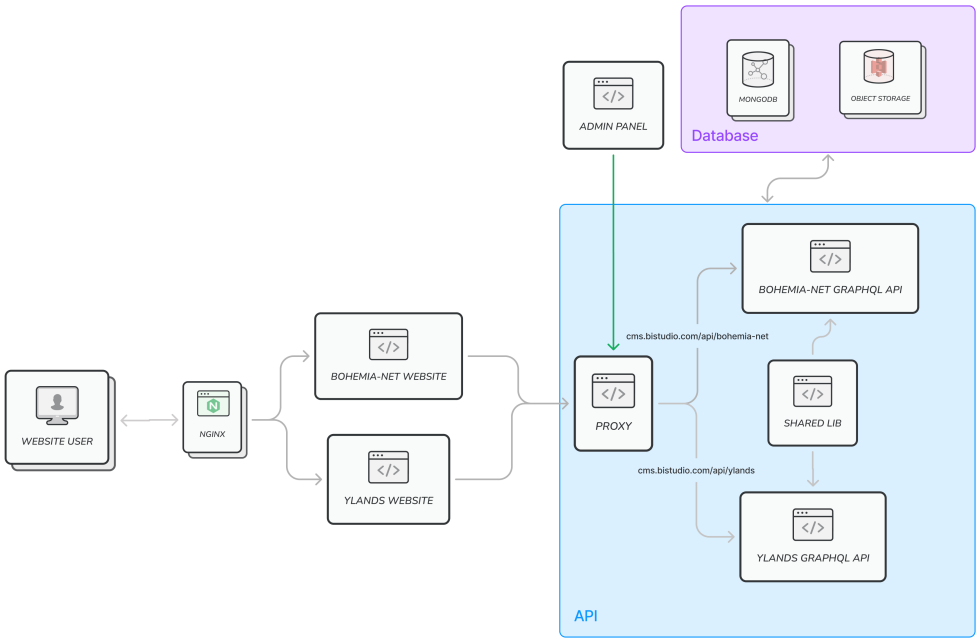
Obrázek 3.3 Knihovna médií



3.4 Architektura

Na obrázku 3.4 je znázorněna architektura CMS. Je navržena s ohledem na vývojáře webových stránek, kteří budou data z CMS získávat a prezentovat uživatelům. Nové weby, které se budou na CMS napojovat, používají framework Next.js, který umožňuje inkrementální statickou regeneraci. Ta umožňuje některé stránky vygenerovat při sestavení celého webu a zbytek vygenerovat v případě požadavku na danou stránku. Vygenerovaná stránka se uloží do cache a není nutné ji znovu generovat při dalších požadavcích. Dále je možné uplatnit strategii cachování stale-while-revalidate. Ta umožňuje, aby v určitých intervalech framework zkusil získat nová data a v případě úspěchu vygeneroval novou stránku, která se uloží do cache. V případě, že dojde k chybě, je uživateli prezentována stará verze stránky (stale). Také je možné cache zneplatnit na základě požadavku (např. nový záznam v CMS). Tyto funkcionality frameworku Next.js nám umožňují velice efektivně cachovat stránky a šetřit tedy výkonem CMS. Namísto dotazování na server a databázi při každém požadavku na zobrazení určité stránky je udržována cache, která se průběžně aktualizuje, pokud dojde ke změně dat.

Obrázek 3.4 Diagram architektury



Každý webový projekt má vlastní GraphQL API, které vývojářům těchto webů nabízí všechna potřebná data. Jednotlivé servery sdílí nemalé množství společného kódu uloženého ve společných knihovnách, které spolu jednotlivé API sdílí a využívají. Požadavky na tyto jednotlivé API jsou směrovány přes proxy server. Data jsou uložena v dokumentové databázi MongoDB a soubory (např. obrázky) jsou ukládány do objektového úložiště.

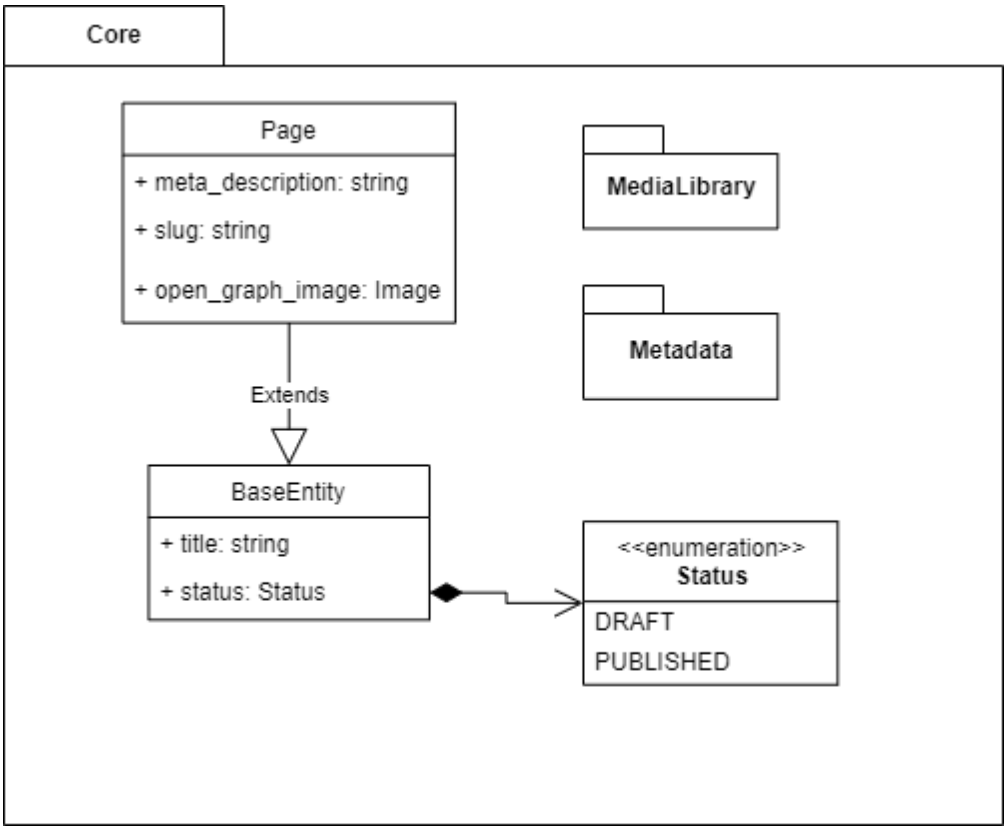
Obsah je spravován přes administrační prostředí, které si automaticky zjistí všechny potřebné informace ohledně daného API (jaké kolekce nabízí, položky dané kolekce atd.).

Veškeré serverové instance využívají pro nasazení, škálování a správu platformu Kubernetes.

3.4.1 Core balíček

Všechny entity spravované v admin panelu budou dědit minimálně třídu BaseEntity, aby bylo možné je v admin panelu zobrazit a spravovat. Entity, které budou dědit třídu PageEntity, jsou objekty, které představují stránky na jednotlivých webech (např. článek) a obsahují navíc položky pro SEO optimalizaci.

Obrázek 3.5 Diagram balíčku Core

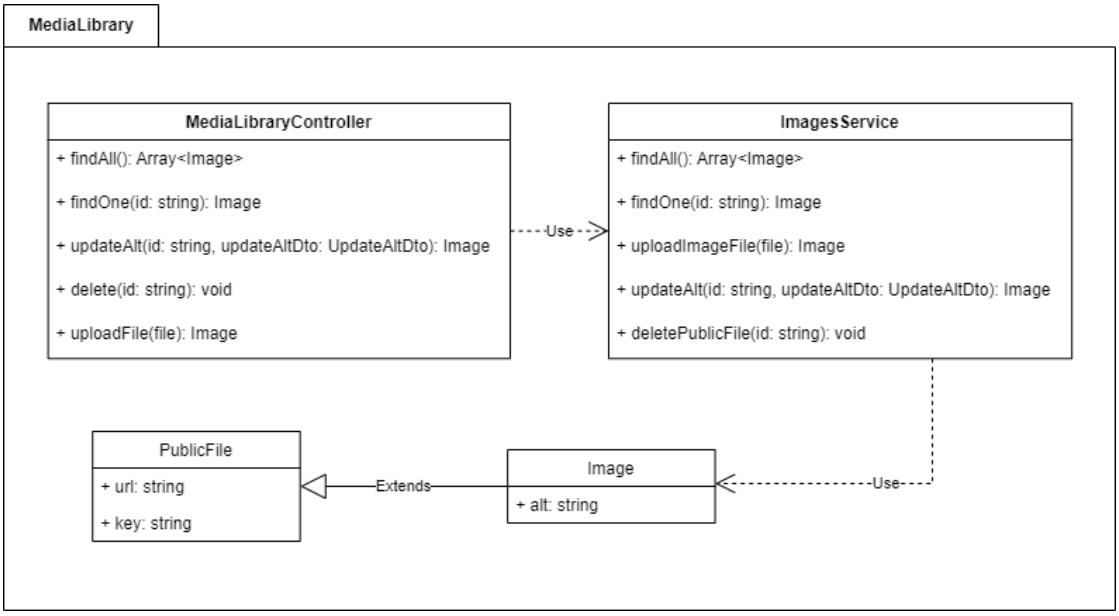


Obrázky v položce Open Graph Image by měly být v minimální velikosti 1200px na 630px a s poměrem stran 1.91:1

3.4.2 Knihovna médií

Knihovna médií bude nabízet rozhraní pro čtení, nahrávání a mazání obrázků. Dále bude možné upravovat atribut alt. Toto rozhraní bude zpřístupněno pouze pro admin panel, kde bude možné s obrázky manipulovat.

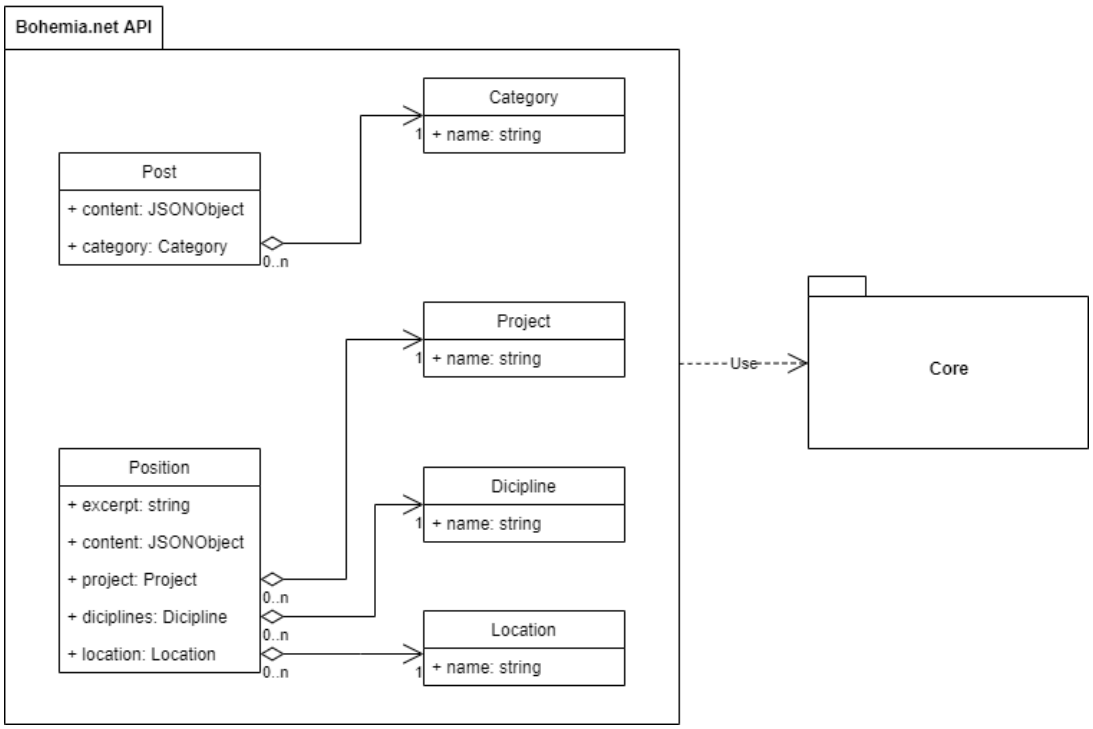
Obrázek 3.6 Diagram balíčku knihovny médií



3.4.3 Bohemia.net API

Na class diagramu 3.7 jsou znázorněny entity projektu Bohemia.net API, které dědí ze základních tříd.

Obrázek 3.7 Diagram entit projektu Bohemia.net



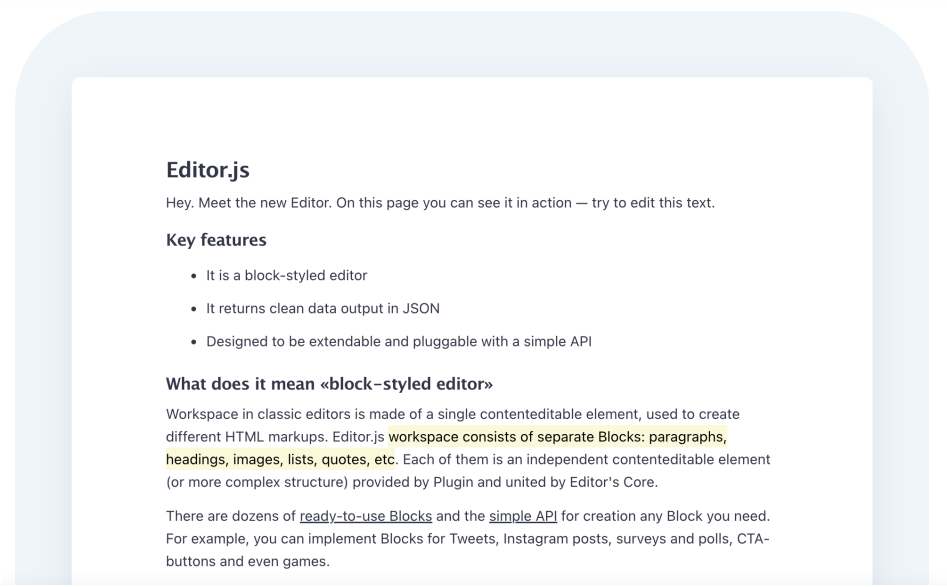
3.5 Rešerše Rich Text editorů

Jsou uvažovány pouze editory, které dávají výstup ve formátu JSON (na základě funkčního požadavku). Pro lepší uživatelský zážitek je dále potřeba si uživatelské rozhraní editoru upravovat a je tedy nutné mít dostatečnou flexibilitu.

3.5.1 Editor.js

Editor.js je volný, otevřený, blokový Rich Text editor. Obsahuje značné množství předpřipravených pluginů a nabízí jednoduché API pro vytvoření vlastních. Pro použití v kombinaci s Reactem nemá přímou podporu a je potřeba si napsat vlastní wrapper [13].

Obrázek 3.8 Ukázka editoru Editor.js



3.5.2 Draft.js

Draft.js je framework pro tvorbu Rich Text editorů v Reactu. Nabízí immutable model (stav nelze měnit, je celý nahrazen). Je možné s ním vytvořit jakýkoliv editor, ať už chceme podporovat jen pár možností editace textu, tak komplexní editor s dlouhými dokumenty. Bohužel v základu nenabízí skoro žádné základní funkcionality. Je potřeba si všechny typy bloků implementovat vlastnoručně [14].

3.5.3 TipTap

TipTap je headless framework pro tvorbu rich text editorů. Neobsahuje žádné CSS styly a vývojář má pod kontrolou jak chování, tak stylování. Není pevně svázán s javascriptovým frameworkem a je tedy možné ho použít jak s Vue.js tak i s Reactem. Celý kód je napsán v TypeScriptu. Výstupem editoru je buď HTML nebo JSON. TipTap je široce rozšiřitelný a zároveň v základu nabízí spoustu pluginů, které je také možné rozšířit [15].

Vlastnosti TipTapu odpovídají našim požadavkům, a proto ho použijeme v naší implementaci Rich Text editoru.

Kapitola 4

Technické zpracování a implementace

V první části kapitoly je popsaná problematika teorií webových API. Konkrétně popisuje, jaké přístupy a technologie existují pro tvorbu webových API. Následuje sekce s popisem vlastního řešení, ve které je vysvětleno celé technické zpracování a důvody, proč bylo takové řešení použito.

4.1 Metodiky a technologie vhodné pro tvorbu webových API

4.1.1 Co je to API

Aplikační programové rozhraní (API) je rozhraní pro programování, které umožňuje interakci s jiným softwarem. Podobně jako uživatelské rozhraní usnadňuje interakci mezi lidmi a počítači. Rozhraní API je implementováno aplikacemi, knihovnami a operačními systémy. Určuje názvy a konvence volání, které by měl programátor používat, aby mohl využívat jejich služby. Mohou zahrnovat specifikace procedur, datových struktur, tříd objektů a protokolů používaných ke komunikaci mezi programovanou aplikací a vývojářem API [16].

4.1.2 Web API

V kontextu vývoje webových aplikací je rozhraní API obvykle definovaná sada požadavků (Requests) protokolu HTTP (Hypertext Transfer Protocol) spolu s definicí struktury odpovědí (Response), obvykle ve formátu XML (Extensible Markup Language) nebo JSON (JavaScript Object Notation). Zatímco "Web API" je prakticky synonymem pro webovou službu, v poslední době se trend (tzv. Web 2.0) odklání od služeb založených na protokolu SOAP (Simple Object Access Protocol) směrem k přímější komunikaci ve stylu REST (Representational State Transfer) [17].

4.1.3 XML

XML (Extensible Markup Language) je značkovací jazyk podobný jazyku HTML, ale bez předem definovaných značek. Místo toho si definujete vlastní značky navržené speciálně pro vaše potřeby. Jedná se o výkonný způsob ukládání dat ve formátu, který lze ukládat, prohledávat a sdílet. A co je nejdůležitější, protože základní formát XML je standardizovaný, pokud sdílíte nebo přenášíte

XML mezi systémy nebo platformami, ať už lokálně nebo přes internet, příjemce může data stále číst a analyzovat díky standardizované syntaxi XML [18].

Existuje mnoho jazyků založených na XML, včetně XHTML, MathML, SVG, XUL, XBL, RSS a RDF. Můžete si také definovat vlastní [18].

4.1.4 JSON

JavaScript Object Notation (JSON) je standardní textový formát pro reprezentaci strukturovaných dat založený na objektové syntaxi jazyka JavaScript. Běžně se používá k přenosu dat ve webových aplikacích (např. k odeslání dat ze serveru na klienta, aby se mohla zobrazit na webové stránce, nebo naopak) [19].

4.1.5 SOAP

SOAP je protokol určený k výměně strukturovaných informací v decentralizovaném a distribuovaném prostředí. K definování rozšiřitelného frameworku pro zasílání zpráv využívá technologii XML. Framework poskytuje konstrukci zpráv, jež lze vyměňovat prostřednictvím různých základních protokolů. Framework byl navržen tak, aby byl nezávislý na konkrétním programovacím modelu a další specifické implementační sémantice [20].

4.1.6 REST API

REST API (známé také jako RESTful API) je aplikační programové rozhraní (API nebo webové API), které splňuje požadavky architektonického stylu REST a umožňuje interakci s RESTful webovými službami. Zkratka REST znamená Representational State Transfer (přenos reprezentativního stavu) a vytvořil ji počítačový vědec Roy Fielding [21].

REST je soubor architektonických požadavků, nikoli protokol nebo standard. Vývojáři API mohou implementovat REST API různými způsoby [21].

Když je klientem zadán požadavek prostřednictvím RESTful API, přenese se reprezentace stavu prostředku žadateli nebo koncovému bodu. Tato informace nebo reprezentace je prostřednictvím protokolu HTTP doručena v jednom z několika formátů: JSON (Javascript Object Notation), HTML, XML, PHP nebo prostý text. JSON je obecně nejoblíbenější formát, protože navzdory svému názvu je jazykově nezávislý a také čitelný pro lidi i stroje [21].

Aby bylo rozhraní API považováno za RESTful, musí dle [21] splňovat tato kritéria:

- Architektura klient-server složená z klientů, serverů a prostředků, přičemž jsou požadavky zpracovávány prostřednictvím protokolu HTTP.
- Bezstavová komunikace klient-server, což znamená, že se mezi požadavky neukládají žádné informace o klientovi a každý požadavek je samostatný a vzájemně nepropojený.
- Data je možné uložit do mezipaměti (cachování) a zefektivnit tak interakci mezi klientem a serverem.
- Jednotné rozhraní mezi komponentami. Informace jsou tedy přenášeny ve standardní podobě. To vyžaduje, aby:
  - požadované zdroje byly identifikovatelné a oddělené od reprezentací zasílaných klientovi.
  - klient mohl manipulovat se zdroji prostřednictvím reprezentace, kterou obdržel, protože obsahuje dostatek informací, které to umožňují.
  - sebepopisné zprávy vrácené klientovi měly dostatek informací k popisu toho, jak je má klient zpracovat.

  - hypertext/hypermédia byly k dispozici, což znamená, že po přístupu ke zdroji by měl být klient schopen pomocí hypertextových odkazů najít všechny další aktuálně dostupné akce, které může provést.

- Vícevrstvý systém, který organizuje jednotlivé typy serverů (ty, které jsou zodpovědné za bezpečnost, vyrovnávání zátěže atd.), se podílí na získávání požadovaných informací do hierarchií, které jsou pro klienta skryty.

- Code-on-demand (volitelné) - možnost odeslat klientovi spustitelný kód ze serveru na vyžádání a rozšířit tak funkčnost klienta.

Přes to, že musí REST API splňovat výše popsaná kritéria, je jeho použití považováno za jednodušší, než použití předepsaného protokolu. Za předepsaný protokol považujeme SOAP (Simple Object Access Protocol), který má specifické požadavky. Těmito požadavky je například zasílání zpráv XML, zabudované zabezpečení a dodržování transakcí, což jej činí pomalejším [21].

REST je proti protokolu SOAP pouze souborem pokynů, které lze implementovat dle potřeby. REST API je tedy rychlejší, lehčí a lépe se škáluje. Často je ideální volbou pro vývoj internetu věcí (IoT) a mobilních aplikací [21].

4.1.7 GraphQL

GraphQL je dotazovací jazyk určený k vytváření klientských aplikací, který poskytuje intuitivní a flexibilní syntaxi a systém pro popis jejich datových požadavků a interakcí [22].

Například požadavek 1 obdrží název příspěvku s id 23 z implementace GraphQL společnosti Bohemia Interactive.

```
{
  post(id: 23) {
    title
  }
}
```

Výpis kódu 1 Příklad GraphlQL požadavku

Ten následně vytvoří výsledná data (ve formátu JSON):

```
{
  "post": {
    "title": "Bohemia Recap - February 2021"
  }
}
```

Výpis kódu 2 Výsledná data ve formátu JSON

GraphQL není programovací jazyk schopný libovolných výpočtů, ale jazyk používaný k zadávání požadavků na aplikační služby. GraphQL nepředepisuje aplikačním službám, které jej implementují, konkrétní programovací jazyk ani systém ukládání. Místo toho aplikační služby přebírají specifikaci GraphQL a mapují je na svůj jazyk, typový systém a filozofii, které GraphQL nabízí. To poskytuje jednotné rozhraní přívětivé pro vývoj produktů a výkonnou platformu pro tvorbu nástrojů [22].

Dle [22] má GraphQL řadu principů návrhu:

- **Zaměření na produkt:** GraphQL je bezvýhradně řízeno požadavky na zobrazovací vrstvu a front-end vývojáře, kteří je píší. GraphQL vychází z jejich způsobu myšlení a požadavků a vytváří jazyk a prostředí, které to umožňují.

- **Hierarchie:** Většina dnešního vývoje produktů zahrnuje vytváření a manipulaci s hierarchiemi zobrazovacích vrstev. Aby bylo dosaženo souladu se strukturou těchto aplikací, je samotný požadavek GraphQL strukturován hierarchicky. Požadavek má stejný tvar jako data v jeho odpovědi. Pro klienty je to přirozený způsob, jak popsat požadavky na data.

- **Striktní datové typy:** Každá služba GraphQL definuje systém typů specifický pro danou aplikaci. Požadavky jsou prováděny v kontextu tohoto typového systému. Při zadání operace GraphQL mohou nástroje před jejím provedením, tj. v době vývoje, zajistit, že je syntakticky správná a platná v rámci tohoto typového systému a služba může poskytnout určité záruky ohledně tvaru a povahy odpovědi.

- **Tvar odpovědi specifikován klientem:** Služba GraphQL prostřednictvím svého typového systému zpřístupňuje možnosti, které mohou její klienti využívat. Za přesnou specifikaci způsobu, jakým budou tyto zpřístupněné možnosti využívány, je odpovědný klient. Tyto požadavky jsou specifikovány až na úrovni položek (fields) jednotlivých objektů. Ve většině aplikací typu klient-server napsaných bez jazyka GraphQL určuje tvar dat vracených z různých koncových bodů služba. Naproti tomu odpověď GraphQL obsahuje přesně to, o co klient žádá a nic víc.

- **Introspektiva:** GraphQL je introspektivní. Systém typů služby GraphQL může být dotazován samotným jazykem GraphQL. Introspekce jazyka GraphQL slouží jako výkonná platforma pro vytváření společných nástrojů a knihoven klientského softwaru.

Díky těmto principům je GraphQL výkonným a produktivním prostředím pro vytváření klientských aplikací. Vývojáři a návrháři produktů, kteří vytvářejí aplikace proti fungujícím službám GraphQL s podporou kvalitních nástrojů, se mohou rychle stát produktivními bez čtení rozsáhlé dokumentace a s malým nebo žádným formálním školením [22].

4.2 Relační vs. NoSQL databáze

Relační a NoSQL jsou dva typy databázových systémů, které se běžně používají v cloudových aplikacích. Jsou různě konstruovány, různě ukládají data a různě se k nim přistupuje [23].

Relační databáze jsou již desítky let rozšířenou technologií. Jsou osvědčené a široce implementované. Konkurenčních databázových produktů, nástrojů a expertních znalostí je mnoho. Relační databáze poskytují úložiště souvisejících datových tabulek. Tyto tabulky mají pevně dané schéma, ke správě dat používají jazyk SQL (Structured Query Language) a podporují ACID [23].

Databáze No-SQL jsou vysoce výkonná nerelační datová úložiště. Vynikají snadným použitím, škálovatelností, odolností a dostupností. Namísto spojování tabulek s normalizovanými daty ukládá NoSQL nestrukturovaná nebo částečně strukturovaná data, často ve formě dvojic klíč-hodnota nebo dokumentů JSON. Databáze NoSQL obvykle neposkytují ACID. Služby s velkým objemem dat, které vyžadují dobu odezvy pod sekundou, upřednostňují datová úložiště NoSQL [23].

Databáze NoSQL zahrnují několik různých modelů přístupu k datům a jejich správy, z nichž každý je vhodný pro konkrétní případy použití. Tabulka 4.1 představuje čtyři běžné modely [23].

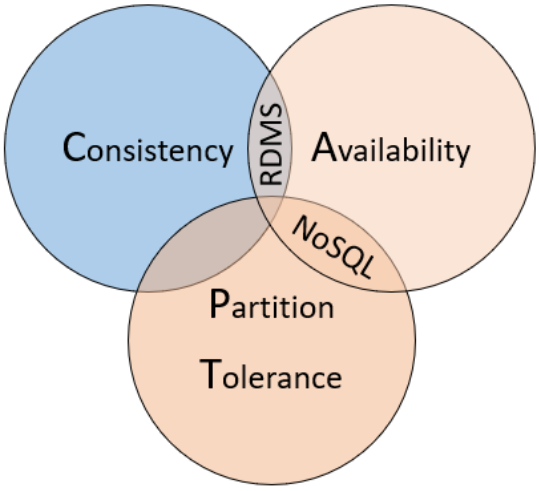
Tabulka 4.1 Datové modely pro NoSQL databáze

Modelování	Vlastnosti
Dokumentové úložiště	Data a metadata jsou hierarchicky uložena v dokumentech JSON uvnitř databáze.
Úložiště typu klíč-hodnota	Nejjednodušší z databází NoSQL, data jsou reprezentována jako kolekce dvojic klíč-hodnota.
Wide-Column Store	Související data jsou uložena jako sada vnořených dvojic klíč-hodnota v rámci jednoho sloupce.
Grafové úložiště	Data jsou uložena ve formě grafové struktury jako uzly, hrany a vlastnosti dat.

4.2.1 CAP teorém

Pro pochopení rozdílů mezi těmito typy databází uveďme větu CAP, což je soubor principů aplikovaných na distribuované systémy, které uchovávají stav [23].

Obrázek 4.1 Tři vlastnosti teorému CAP (Zdroj: Relational vs. NoSQL data)



Teorém uvádí, že distribuované datové systémy nabízejí kompromis mezi konzistencí, dostupností a tolerancí výpadku. A že každá databáze může zaručit pouze dvě z těchto tří vlastností [23]:

- **Konzistence:** Každý uzel v clusteru odpoví nejnovějšími daty, i když systém musí požadavek zablokovat, dokud se neaktualizují všechny repliky. Pokud se "konzistentního systému" zeptáte na položku, která se právě aktualizuje, budete na tuto odpověď čekat, dokud se všechny repliky úspěšně neaktualizují. Obdržíte však nejaktuálnější data.
- **Dostupnost:** Každý uzel vrací okamžitou odpověď, i když tato odpověď neobsahuje nejaktuálnější data. Pokud se zeptáte "dostupného systému" na položku, která se aktualizuje, dostanete nejlepší možnou odpověď, jakou může služba v danou chvíli poskytnout.
- **Tolerance výpadků:** Zaručuje, že systém bude nadále fungovat, i když replika selže nebo ztratí spojení s ostatními replikami.

CAP teorém vysvětluje kompromisy spojené s řízením konzistence a dostupnosti během výpadku sítě. Kompromisy s ohledem na konzistenci a výkonnost však existují i v případě, že k výpadku sítě nedojde [23].

Relační databáze obvykle poskytují konzistenci a dostupnost, ale ne toleranci výpadku. Obvykle jsou zajišťovány na jednom serveru a škálují se vertikálně přidáváním další výpočetní kapacity [23].

Mnoho relačních databázových systémů podporuje integrované funkce replikace, kdy je možné vytvářet kopie primární databáze na jiné instance sekundárních serverů. Operace zápisu se provádějí do primární instance a replikují se do všech sekundárních instancí. Při selhání může primární instance přejít na sekundární a zajistit tak vysokou dostupnost. Sekundární instance lze také použít k distribuci operací čtení. Zatímco operace zápisu jdou vždy proti primární replice, operace čtení mohou být směrovány na kteroukoli sekundární instanci, aby se snížilo zatížení systému [23].

Data lze také horizontálně rozdělit mezi více uzlů, například pomocí shardingu. Sharding však výrazně zvyšuje provozní režii, protože data rozděluje na mnoho částí, které spolu nemohou snadno komunikovat. Jeho správa může být nákladná a časově náročná. Relační funkce, mezi které patří spojování tabulek, transakce a referenční integrita, vyžadují při nasazení shardingu výrazné výkonnostní ztráty [23].

Databáze NoSQL obvykle podporují vysokou dostupnost a toleranci výpadku. Jsou horizontálně škálovatelné, často na běžných a levných serverech. Tento přístup zajišťuje obrovskou dostupnost v rámci geografických regionů i mezi nimi při nižších nákladech. Data se rozdělují a replikují na těchto serverech nebo uzlech, což zajišťuje redundanci a odolnost proti chybám [23].

4.3 Implementace

4.3.1 Použité technologie

4.3.1.1 Monorepo

Monorepo je jediný repozitář obsahující více různých projektů s přesně definovanými vztahy [24].

4.3.1.2 pnpm

pnpm je alternativní správce balíčků pro Node.js. Je to přímá náhrada za npm, ale je rychlejší a efektivnější [25].

Soubory uvnitř složky *node_modules* jsou propojeny z jediného úložiště s adresovatelným obsahem. Je až 2x rychlejší než alternativy (Benchmark). Má vestavěnou podporu monorepozitářů. Ve výchozím nastavení vytváří "neplochou" strukturu *node_modules*, takže kód nemá přístup k libovolným balíčkům [26].

4.3.1.3 Nx

Nx je rychlý a rozšiřitelný build systém s podporou monorepozitářů [27].

Nx má podobnou filozofii návrhu jako Visual Studio Code. Code je výkonný textový editor a můžete s ním být velmi produktivní, i když si nenainstalujete žádná rozšíření. Ekosystém rozšíření VSCode je však to, co může vaši produktivitu skutečně zvýšit [27].

Nx je na tom podobně. Jádro Nx je obecné, jednoduché a nenápadné. Zásuvné moduly Nx, ačkoli jsou pro mnoho projektů velmi užitečné, jsou zcela volitelné [27].

4.3.1.4 JavaScript (Krátká historie)

JavaScript (známý také jako ECMAScript) vznikl jako jednoduchý skriptovací jazyk pro prohlížeče. V době kdy byl vynalezen se předpokládalo, že bude používán pro krátké úryvky kódu vložené do webové stránky - psaní více než několika desítek řádků kódu by bylo poněkud neobvyklé. Z tohoto důvodu první webové prohlížeče prováděly takový kód poměrně pomalu. Postupem času

se však JS stával stále populárnějším a vývojáři webových stránek jej začali používat k vytváření interaktivních zážitků [28].

```
if ("" == 0) {
// Pravda. Ale proc??
}
if (1 < x < 3) {
// Pravda pro jakoukoliv hodnotu promenne x!
}
```

Výpis kódu 3 Operátor rovnosti (==) v jazyce JavaScript vynucuje své argumenty, což vede k neočekávanému chování.

```
const obj = { width: 10, height: 15 };
// Proc toto neni cislo (NaN - not a number)? Hlaskovani je tezke!
const area = obj.width * obj.heigth;
```

Výpis kódu 4 JavaScript také umožňuje přístup k atributům, které nejsou definované.

Většina programovacích jazyků by při výskytu těchto chyb vyhodila chybu. Některé by tak učinily během kompilace - před spuštěním kódu. Při psaní malých programů jsou takové hříčky nepříjemné, ale dají se zvládnout; při psaní aplikací se stovkami nebo tisíci řádků kódu jsou tato neustálá překvapení vážným problémem [28].

4.3.1.5 TypeScript: Kontrola statických typů

Některé jazyky by tyto chybné programy vůbec nedovolily spustit. Zjišťování chyb v kódu bez jeho spuštění se označuje jako statická kontrola. Určení toho, co je chyba a co ne, na základě druhů hodnot, se kterými se pracuje, se nazývá statická typová kontrola [28].

TypeScript kontroluje program na chyby před spuštěním, a to na základě druhů hodnot. Jedná se o statickou kontrolu typů. Například poslední příklad výše obsahuje chybu kvůli datovému typu obj. Na výpisu kódu 5 je chyba, kterou TypeScript našel:

```
const obj = { width: 10, height: 15 };
const area = obj.width * obj.heigth;

Property 'heigth' does not exist on type
'{ width: number; height: number; }'. Did you mean 'height'?
```

Výpis kódu 5 Objevená chyba při kompilaci Typescriptem

4.3.1.6 Node.js

Node.js je asynchronní běhové prostředí JavaScriptu řízené událostmi, které je určeno k vytváření škálovatelných síťových aplikací [29].

Node.js používá V8 JavaScript engine, jádro Google Chrome, mimo prohlížeč [30].

Node.js aplikace běží v jednom procesu, aniž by se pro každý požadavek vytvářelo nové vlákno. Node.js poskytuje ve své standardní knihovně sadu asynchronních I/O primitiv, která

zabraňují blokování kódu JavaScriptu, a obecně jsou knihovny v Node.js psány s využitím neblokujících paradigmat, takže blokování je spíše výjimkou než normou [30].

Když Node.js provádí I/O operaci, jako je čtení ze sítě, přístup k databázi nebo souborovému systému namísto blokování vlákna a plýtvání cykly CPU, bude Node.js při čekání pokračovat v operacích, než se vrátí odpověď [30].

4.3.1.7 NestJS

Nest (NestJS) je framework pro vytváření efektivních a škálovatelných serverových Node.js aplikací. Používá progresivní JavaScript, je vytvořen pomocí jazyka TypeScript a plně jej podporuje (přesto však umožňuje vývojářům kódovat v čistém JavaScriptu). Kombinuje prvky OOP (objektově orientovaného programování), FP (funkčního programování) a FRP (funkčního reaktivního programování) [31].

4.3.1.8 MongoDB

MongoDB je dokumentová NoSQL databáze, která ukládá data ve flexibilních dokumentech typu JSON. Znamená to tedy, že se pole mohou v jednotlivých dokumentech lišit a struktura dat se může v průběhu času měnit. Model dokumentu se mapuje na objekty v kódu aplikace, což usnadňuje práci s daty. MongoDB je ve svém jádru distribuovaná databáze, takže vysoká dostupnost, horizontální škálování a geografická distribuce jsou integrovány a snadno se používají [32].

Byla vybrána jako ideální kandidát na základě nutnosti flexibility schémat objektů v implementovaném API.

4.3.1.9 React

React je Javascriptová knihovna od společnosti Meta (dříve Facebook) pro vytváření uživatelských rozhraní. Zaměřuje se pouze na zobrazovací vrstvu, případně na logiku daného vykreslování. O vykreslení se postarají jednotlivé komponenty, které jsou hierarchicky uspořádány do stromu. Každá komponenta v sobě zapouzdřuje dané funkcionality [33].

Velmi dobrý způsob dělení komponent je na tzv. view komponenty a container komponenty. View komponenty jsou pouze hloupé komponenty, které se starají čistě o vykreslení. Složitější logiku, jako načtení dat ze serveru, uchováváme v container komponentě a předáváme data view komponentám pomocí props.

4.3.1.10 Tailwind CSS

Tailwind CSS je tzv. utility-first CSS framework. Utility-first znamená, že obsahuje atomické classy, kde každá classa aplikuje na element jen jednu věc. Classy jsou generované za běhu (Just In Time Compilation) na základě použití dané classy v kódu. Výsledný soubor se styly je tedy velmi malý a opravdu bude vyžadovat hodně úsilí, aby byl větší než 10 kB [34].

Pomocí Tailwindu lze vytvářet vlastní designy a neobsahuje žádné připravené komponenty. Velmi dobře se doplňuje s knihovnami HeadlessUI a TailwindUI. Tento framework byl zvolen z důvodu možnosti vytvoření vlastního designu a výrazného zvýšení produktivity.

4.3.2 Backend

4.3.2.1 API

Základem pro zpřístupnění obsahu jsou jednotlivá GraphQL API pro každý projekt. Ta umožňují vývojářům webů snadno přistupovat k obsahu pomocí jednotného rozhraní. Pro zrychlení vývoje prototypu byla použita knihovna nestjs-query postavená na frameworku NestJS.

V základu nabízí generování CRUD operací pro GraphQL na základě předepsaného modelu. Podporované knihovny pro připojení a práci s databází jsou TypeORM, Sequelize a Mongoose. Mongoose je ODM (Object Document Mapper), který je použit v prototypu. Dále jsou v základu podporovány relace one to one, one to many, many to one a many to many. Velkou výhodou je i zabudování dataloaderu, který předchází tzv. n+1 problému. Další funkcionalitou je také vestavěné stránkování a filtrace.

Knihovna velice zjednodušuje a urychluje vývoj nových entit. Stačí vytvořit entitu, DTO (Data Transfer Object) a předat je knihovně, která se postará o vygenerování všech operací. Knihovna je flexibilní a funkcionality je možné upravovat podle potřeb.

GraphQL API projektu Bohemia.net je na lokálním vývojovém prostředí zpřístupněno na adrese http://localhost:3010/graphql. Z tohoto endpointu mohou frontend vývojáři získávat potřebná data.

4.3.2.2 Kolize názvů kolekcí v databázi

Názvy kolekcí v databází jsou určovány na základě názvu třídy daného modelu. Může se stát, že ve více API budou stejné objekty (např. Post může být ve více webových projektech). Více objektů stejného názvu by znamenalo stejný název kolekce a tedy kolizi. Data z jednoho API by se míchala s daty z jiného API. Tuto kolizi vyřešíme prefixem kolekce daného projektu.

```
NestjsQueryMongooseModule.forFeature([
  {
    document: Post,
    name: `${environment.entityPrefix}_${Post.name}`,
    schema: PostSchema,
  },
]);
```

Výpis kódu 6 Prefix názvu kolekce daným názvem projektu

4.3.2.3 Autentizace a autorizace

Zabezpečení obsahu proti neoprávněným úpravám či čtení je v takovém systému nutností. Autentizace, autorizace a identita jednotlivých uživatelů využívajících administrační prostředí bude zprostředkována skrze externí systém napojený na CMS. Vzhledem k tomu, že se jedná o několikrát vyřešený problém, není autentizace a autorizace součástí prototypu této práce.

4.3.2.4 Metadata pro admin panel

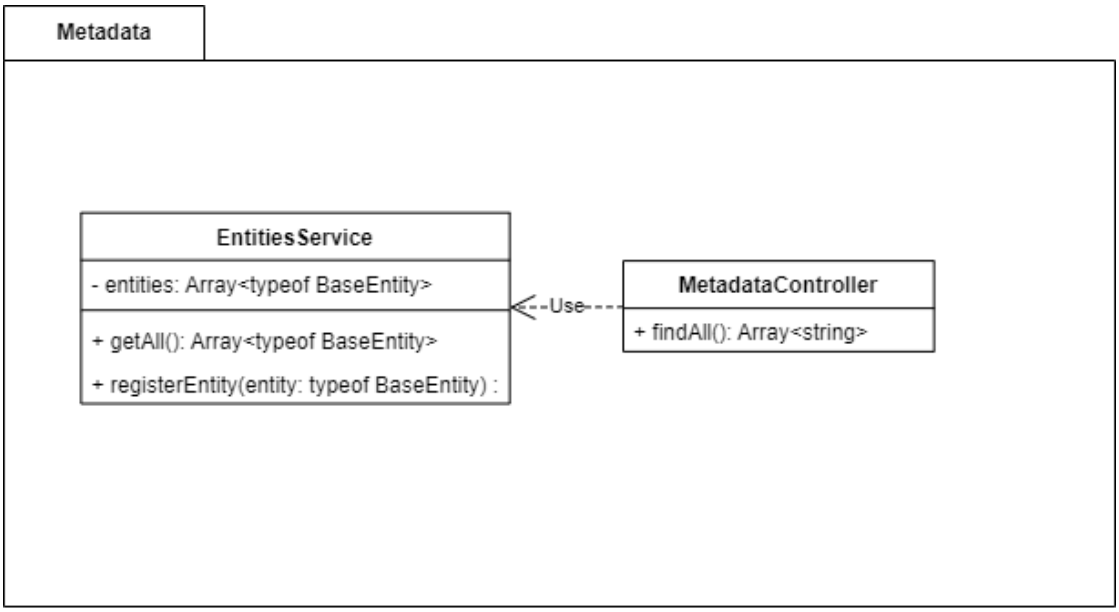
Přístupů, jak poskytovat metadata pro admin panel, je více. Na základě existujících entit je možné generovat další API se kterým bude komunikovat pouze admin panel. Generování takového API (ať už REST nebo GraphQL) je možné například pomocí dekorátorů, které se deklarativně použijí na entity a jejich atributy. Tento přístup nabízí širokou flexibilitu a zpřístupnění mnoha metadat o dané entitě, která by mohla být potřeba.

Další možností je získávat potřebné údaje pomocí systému introspekce, které poskytuje samotné GraphQL. GraphQL API pro jednotlivé webové projekty nebude veřejné, ale bude přístupné pouze z vnitřní sítě, kde jsou umístěny i webové projekty. Z tohoto důvodu je možné introspekci zpřístupnit i na produkci a získávat tak pomocí něho informace o daném schématu.

Je také možné obě varianty kombinovat, což bylo využito v implementaci prototypu této práce. Seznam názvů entit, které jsou určeny k editaci pomocí admin panelu, je zpřístupněn pomocí jednoduchého endpointu, který tento seznam vrátí. Celý modul, který obsluhuje přidávání názvů

entit a jeho zpřístupnění, se nazývá Metadata. V průběhu návrhu nebylo známo, jaký přístup bude nakonec zvolen. Z tohoto důvodu je diagram tříd uveden až v této kapitole.

Obrázek 4.2 Diagram tříd balíčku Metadata



4.3.2.5 Generování operací pro admin panel

Frontend má tedy již připravené informace ohledně kolekcí a jejich položkách, které může vytvářet a editovat. Ještě ale potřebuje příslušné operace, kterým pouze předá příslušné objekty jako parametry a odešle je na server. Generování těchto operací je zajištěno pomocí knihovny gql-generator, která na základě schématu vygeneruje příslušné operace. Výstup tohoto nástroje pošleme do nástroje GraphQL Code Generator, který následně vygeneruje typy pro Typescript a operace pro Apollo klienta.

```
overwrite: true
schema:
  - ./apps/bohemia-net/schema.gql
documents:
  - ./libs/data-access/src/@generated/bohemia-net/graphql/**/*.gql
generates:
  ./libs/data-access/src/lib/@generated/bohemia-net/generated.tsx:
    plugins:
      - typescript
      - typescript-operations
      - typescript-react-apollo
    config:
      withHooks: false
```

Výpis kódu 7 Konfigurace GraphQL Code Generator pro Bohemia.net

```
"generate-operations": {
  "executor": "@nrwl/workspace:run-commands",
  "options": {
    "commands": [
      {
        "command": "gqlg --schemaFilePath ./apps/bohemia-net/schema.gql
        ↪  --destDirPath
        ↪  ./libs/data-access/src/@generated/bohemia-net/graphql --depthLimit
        ↪  3"
      },
      {
        "command": "graphql-codegen -c ./apps/bohemia-net/codegen.yml"
      }
    ]
  }
}
```

Výpis kódu 8 Příkaz pro vygenerování GraphQL operací pro klienta

4.3.2.6 Knihovna médií

Knihovna médií implementuje rozhraní ke komunikaci s admin panelem. Controller obsahuje 5 metod - `findAll`, `findOne`, `updateAlt`, `remove` a `uploadFile`.

Metoda `uploadFile` zároveň zkontroluje podporované formáty souborů. Těmi jsou pouze obrázky, tedy soubory s příponami `jpg`, `jpeg`, `png`, `webp`. Dále se zavolá metoda `uploadImageFile` třídy `ImageService`. V implementaci tato metoda nezajistí uložení daného souboru. Tato funkcionalita bude implementována v budoucnu pomocí externího úložiště typu ObjectStorage, která soubor uloží a vrátí URL na tento veřejný soubor. Do databáze je prozatím uložen náhodný obrázek, místo nahraného.

4.3.2.7 Metadata

Balíček Metadata poskytuje controller pro získání všech entit, které mají být editovatelné v admin panelu.

```
import { Controller, Get } from '@nestjs/common';
import { EntitiesService } from './entities.service';

@Controller('metadata')
export class MetadataController {
  constructor(private entitiesService: EntitiesService) {}

  @Get()
  findAll() {
    return this.entitiesService.getAll().map((item) => item.name);
  }
}
```

Výpis kódu 9 Controller balíčku Metadata

Dále poskytuje jednoduchou třídu se dvěma metodami a polem všech entit. Každá entita, která má být zpřístupněna v admin panelu, se při inicializaci zaregistruje pomocí metody `registerEntity`.

```
import { Injectable } from '@nestjs/common';
import { BaseEntity } from '../base.entity';

@Injectable()
export class EntitiesService {
  private readonly entities: Array<typeof BaseEntity> = [];

  getAll() {
    return this.entities;
  }

  registerEntity(entity: typeof BaseEntity) {
    this.entities.push(entity);
  }
}
```

Výpis kódu 10 Třída EntitiesService

4.3.3 Admin Panel

Celý kód implementace admin panelu lze nalézt v adresáři *apps/admin-panel*. Frontend je napsán v Reactu a vstupním bodem je soubor *app.tsx*, který obsahuje inicializaci hlavních komponent a především routování pomocí knihovny react-router-dom.

Komponenta ErrorBoundary obaluje celou aplikaci a zachycuje všechny případné nepodchycené výjimky. Komponenta SWRConfig konfiguruje knihovnu SWR. Toaster komponenta je hlavní komponenta pro zobrazování toast notifikací knihovny React Hot Toast. BrowserRouter komponenta a její děti definují cesty aplikace a komponenty, které se mají na příslušných cestách zobrazit. Dynamické části adres jsou označeny dvojtečkou před názvem.

4.3.3.1 SWR

Jedná se o knihovnu, která slouží k fetchování a cachování dat pomocí React hooků. Zkratka SWR pochází z názvu `stale-while-revalidate`, což je strategie HTTP cachování. Tato strategie nejprve vrátí data z cache (stale), poté pošle požadavek na aktualizaci dat (revalidate) a nakonec data aktualizuje v cache. To umožňuje uživatelskému rozhraní přijímat stále aktuální data. V implementaci je tato knihovna využita pouze zřídka, a to na získání všech entit, které mohou být v admin panelu editovatelné a k fetchování dat z knihovny médií. Nicméně se do budoucna počítá s rozšířením API a tedy i větším využitím této knihovny.

4.3.3.2 React Hot Toast

React Hot Toast je jednoduchá utility knihovna pro zobrazování notifikací. Umožňuje vlastní design notifikací a nabízí deklarativní API pro zobrazení jednotlivých notifikací.

```
toast.success('Data úspěšně uložena')
```

Výpis kódu 11 Zavoláním funkce success se zobrazí komponenta s notifikací

4.3.3.3 Formik

Formik je knihovna, která výrazně usnadňuje práci s formuláři. Odstraňuje velké množství zbytečně opakujícího se kódu. Postará se o:

- zadávání dat a uchování stavu formuláře,
- validaci a zobrazení chybových hlášek,
- odeslání dat ke zpracování.

4.3.3.4 Apollo Client

Apollo Client je knihovna, která se používá k fetchování, cachování a modifikaci dat pomocí GraphQL. Knihovna se pomocí klienta napojí na GraphQL server se kterým následně komunikuje. Pomocí deklarativních React hooků můžeme posílat na server dotazy na potřebná data a mutace pomocí kterých data modifikujeme.

4.3.3.5 React Table

Jedná se o headless knihovnu pro práci s tabulkami. Knihovna je využita v implementaci k výpisu záznamů. Umožňuje vytvořit vlastní design, přináší deklarativní API k vytváření tabulek a umožňuje velké množství rozšíření.

4.3.3.6 Day.js

Nativní API v Javascriptu není pro práci s datumy moc přívětivé. Je lepší využít externí knihovny jako je Day.js. Velikost celé knihovny je pouze 2kB, všechny operace jsou imutabilní a poskytuje stejné API jako Moment.js

4.3.3.7 Layouty

Implementace nabízí dva layouty - SpaceDetailLayout a ListLayout. SpaceDetailLayout je hlavním layoutem s levým menu a pravou obsahovou oblastí. ListLayout je vnitřní layout, který se změní, pokud je zobrazena stránka Create nebo Edit.

4.3.3.8 List

Stránka výpisu umožňuje na základě parametru v URL adrese zobrazit tabulku se záznamy v dané kolekci. Každá entita obsahuje položky `title`, `status`, `created_at` a `updated_at`. Všechny tyto položky zobrazujeme v tabulce uživateli. Celá tabulka je tvořena komponentou `DataListTable`, která vytváří tabulku pomoc React Table a aplikuje na ní příslušné styly pomocí Tailwindu. Přes jednotlivé řádky tabulky se lze prokliknout do editace daného záznamu, čímž se změní layout.

4.3.3.9 Edit

Stránka pro editaci umožňuje uživateli editovat jeden konkrétní záznam. Pro své fungování potřebuje 3 parametry - název projektu, název entity a id záznamu. Název projektu a entity slouží dále jako klíč pro získání vygenerovaných query a mutací. Komponenta si na základě těchto klíčů získá potřebný query dokument, který poté předá společně s id záznamu `useQuery` hooku.

Dále potřebuje komponenta vědět, které fieldy dané entity jsou editovatelné. To zajistí query `getResourceFields`, které se dotáže introspekce, které fieldy daný `ObjectType` nabízí a které jsou vstupní.

```
query getResourceFields($createNameDTO: String!, $readNameDTO: String!) {
  inputFields: __type(name: $createNameDTO) {
    name
    inputFields {
      name
      type {
        kind
        name
        ofType {
          kind
          name
        }
      }
    }
  }
  allFields: __type(name: $readNameDTO) {
    name
    fields {
      name
      type {
        kind
        name
      }
    }
  }
}
```

Výpis kódu 12 Query, které získá na základě názvu ObjectType jeho fieldy

Odpověď v sobě obsahuje informace o fieldech daného ObjectType, především o tom, jestli je daný field povinný, nebo nepovinný.

Dále jsou tyto fieldy namapovány na příslušné komponenty, které se mají pro daný field zobrazit uživateli. Field `Status` je vyřazen z vykreslování, protože jeho hodnota je kontrolována pomocí komponenty `SubmitFormButton`. Formulář je následně vykreslen pomocí knihovny Formik.

```
export const inputTypeMap: { [key: string]: React.ReactElement } = {
  String: <Field type="text" />,
  Int: <Field type="number" />,
  Float: <Field type="number" />,
  DateTime: <Field type="date" />,
  Boolean: <Field type="checkbox" />,
  JSONObject: <Field component={TipTapEditor} />,
};
```

Výpis kódu 13 Mapa skalárů a jejich komponent k zobrazení

Nyní je možné data zobrazit, chybí ještě možnost data editovat a ukládat. Pro uložení dat je potřeba získat příslušný vygenerovaný dokument s mutací. Ten komponenta získá obdobně pomocí dvojice klíčů - název projektu, název entity. Dokument předáme jako parametr do `useMutation` hooku, který vrací pole, kde první položkou je funkce pro uložení dat.

Před uložením je nutné data validovat již na frontendu. K validaci používáme knihovnu Yup, která se dobře integruje do Formiku. Funkce `composeFieldValidation` zkonstruuje pro daný field jeho validační schéma, které následně předá Formiku. Validace proběhně při pokusu o uložení nebo jednotlivě na každém fieldu při "odkliknutí" ze zadávacího pole.

Může nastat situace, kdy bude mít uživatel vyplňování formuláře rozpracované, neuložené a bude chtít ze stránky odejít. Aby nedošlo ke ztrátě zadaných dat, je uživatel upozorněn hláškou, že má ve formuláři neuložená data a pokud je před odchodem neuloží, budou ztracena. K tomuto slouží kombinace hooků `usePrompt` a `useBeforeunload`.

Samotné uložení změněných dat probíhá pomocí funkce `updateResource`, která přijímá jako parametry id záznamu a objekt se změněnými fieldy. Aby nedocházelo ke zbytečnému posílaní požadavků na uložení záznamu, komponenta kontroluje, zdali byl nějaký field změněn. Pokud nebyl, tlačítko pro uložení není možné stisknout.

Na editační stránce je také možné záznam smazat pomocí tlačítka *Delete*. Obdobně si dané tlačítko najde příslušnou mutaci pro smazání záznamu a zavolá ji. Uživatel je po smazání přesměrován na stránku s výpisem.

4.3.3.10 Create

Stránka pro vytvoření nového záznamu se liší od editační jen minimálně. Výchozími hodnotami v zadávacích polích jsou jejich příslušná prázdná pole a po uložení nebo publikování záznamu dojde k jeho vytvoření.

4.3.3.11 TipTap editor

TipTap je nastylován pomocí Tailwindu a pomocí Tailwind pluginu na typografii. Tento plugin jedinou třídou `prose` přidá základní typografické styly pro HTML, které je generované uživatelem.

```
<article class="prose lg:prose-xl">
  {{ html }}
</article>
```

Výpis kódu 14 Příklad použití třídy prose

Editor je nutné integrovat s knihovnou Formik, protože Rich Text je jednou z položek formuláře. Je tedy potřeba, aby Formik kontroloval zadávání a uložení dat v paměti. Stačí editoru

pomocí props předat daný `field` a samotnou instanci `form`. TipTap zpřístupňuje v nastavení atributy `content` a `onUpdate`, přes které propojíme editor s Formikem.

```
content: field.value,
onUpdate({ editor }) {
  form.setFieldValue(field.name, editor.getJSON());
}
```

Výpis kódu 15 Propojení TipTap editoru s Formikem

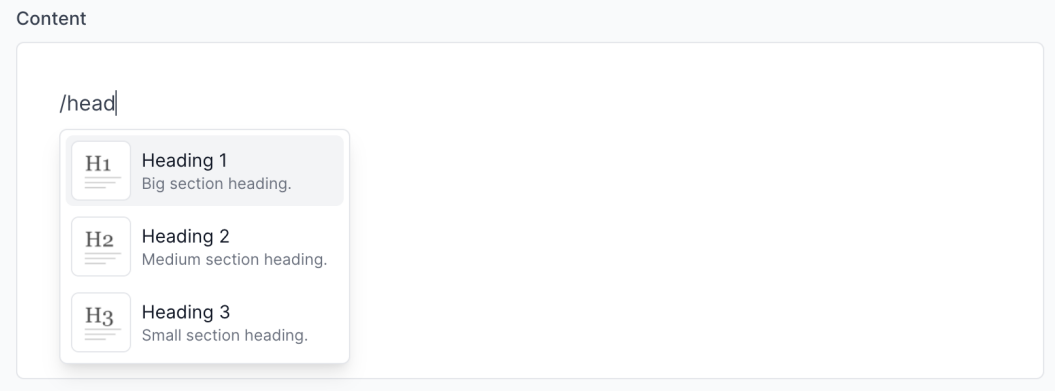
Editor umožňuje inline stylování textu, a to pomocí komponenty `CustomBubbleMenu`, která se zobrazí při označení části textu. Možnosti stylování písma jsou tučné, kurzíva, podtržené, přeškrtnuté, zdrojový kód a odkaz.

Základem editoru je rozšíření StarterKit, které přidává základní bloky jako jsou nadpisy, odstavce, seznamy, horizontální pravítko, zdrojový kód, citát. Tyto bloky jsou doplněny o řadu vlastních nebo upravených bloků jako jsou obrázky nebo příkazy.

Blok s obrázkem bylo nutné upravit pro výběr obrázku z knihovny médií. Pro vložení obrázku do rich textu se uživateli otevře okno pro výběr obrázku z knihovny médií. Je nutné obrázek nejprve vložit do knihovny a až poté do rich textu.

Rozšíření editoru o příkazy je inspirováno aplikací Notion. Toto rozšíření umožňuje uživateli formátovat příslušný text přímo v daném místě, kde se nachází. Pro vyvolání okna s příkazy stačí napsat znak "/". Na výběr máme ze základních bloků a obrázku. Rozšíření umí i nápovědu, pokud tedy za znak lomítka napíšete další text, zobrazí se vám bloky, které danému textu odpovídají.

Obrázek 4.3 Ukázka nápovědy



4.3.3.12 Knihovna médií

Do knihovny médií je možné nahrávat obrázky, a to jak jednotlivě, tak více najednou. Pomocí tlačítka *Add media* se vyvolá kontextové okno pro výběr obrázků ze souborového systému. Pro lepší použitelnost je také možné obrázky přetáhnout přímo do okna prohlížeče a spustit tak jejich nahrávání. Při nahrávání obrázků je uživateli zobrazen aktuální průběh u jednotlivých obrázků.

Po nahrání obrázků je možné se proklinout do jejich detailu, kde je možné zobrazit základní údaje a provést akce smazání obrázku a změnu atributu *alt*. Při změně alternativního textu obrázku dojde k jeho uložení po odkliknutí ze zadávacího pole. Skutečnost, že je text ukládán, je indikována rotujícím kolečkem a úspěšné uložení je indikováno zelenou fajfkou.

4.3.4 Adresářová struktura projektu

```
/
├── .vscode ..... nastavení workspacu pro Visual Studio Code
├── @types ..... adresář s typovými definicemi pro různé moduly
├── apps ..... adresář s projekty aplikace
│   ├── admin-panel ..... zdrojový kód admin panelu
│   └── bohemia-net ..... zdrojový kód Bohemia.net API
├── libs ..... zdrojové kódy knihoven
│   ├── core
│   ├── data
│   ├── data-access
│   └── ui
├── .env.example ..... ukázka obsahu souboru .env
├── .eslintrc.json
├── .prettierrc
├── babel.config.js
├── jest.config.js
├── jest.preset.js
├── nx.json ..... konfigurační soubor NX CLI
├── package.json ..... závislosti
├── pnpm-lock.yaml
├── README.md
├── tsconfig.base.json
└── workspace.json ..... konfigurační soubor s projekty
```

Kapitola 5
Testování

5.1 Automatické testování

Automatizované softwarové testování je považováno za nezbytnou součást každého seriózního vývoje softwaru. Automatizace usnadňuje rychlé a snadné opakování jednotlivých testů nebo sad testů během vývoje. To pomáhá zajistit, aby verze splňovaly cíle v oblasti kvality a výkonu. Automatizace pomáhá zvýšit pokrytí a poskytuje vývojářům rychlejší zpětnou vazbu. Automatizace jednak zvyšuje produktivitu jednotlivých vývojářů, jednak zajišťuje, že testy jsou prováděny v kritických okamžicích životního cyklu vývoje, jako je kontrola zdrojového kódu, integrace funkcí a vydání verze [35].

5.1.1 Unit testy

Unit testy jsou typem testování softwaru, při kterém se testují jednotlivé části nebo komponenty softwaru. Účelem je ověřit, zda jednotlivé části softwarového kódu fungují podle očekávání. Unit testy provádějí vývojáři během vývoje (fáze programování) aplikace. Unit testy izolují část kódu a ověřují jeho správnost. Jednotkou může být jednotlivá funkce, metoda, procedura, modul, nebo objekt [36].

5.1.2 End to end testy

End to end testování je metoda testování softwaru, která ověřuje celý software od začátku až do konce spolu s jeho integrací s externími rozhraními. Účelem end-to-end testování je testování celého softwaru na závislosti, integritu dat a komunikaci s ostatními systémy, rozhraními a databázemi, aby se ověřil kompletní produkční scénář [37].

5.2 Uživatelské testování

Uživatelské testování je proces, při kterém jsou rozhraní a funkce webové stránky nebo aplikace testovány skutečnými uživateli, kteří provádějí konkrétní úkoly v reálných podmínkách. Účelem tohoto procesu je vyhodnotit použitelnost dané webové stránky nebo aplikace a rozhodnout, zda je produkt připraven ke spuštění pro skutečné uživatele. Pro dosažení relevantních výsledků by testující neměli být příliš usměrňováni a mělo by jim být umožněno přirozeně komunikovat s webovou stránkou nebo aplikací, aby se zjistilo, zda je systém dostatečně intuitivní a pohodlný pro používání lidmi, kteří s ním ještě nejsou obeznámeni [38].

Dle Kate Moran se cíle uživatelského testování v jednotlivých studiích liší, ale obvykle zahrnují [39]:

- identifikaci problémů v návrhu webu nebo aplikace,
- odhalení příležitostí ke zlepšení,
- poznání chování a preferencí cílového uživatele.

Existuje mnoho různých typů uživatelského testování, ale základními prvky většiny testů jsou moderátor, úkoly a účastník [39].

Moderátor zadává účastníkovi úkoly. Zatímco účastník tyto úkoly plní, moderátor pozoruje jeho chování a naslouchá zpětné vazbě. Moderátor může také klást doplňující otázky, aby od účastníka zjistil podrobnosti [39].

5.2.1 A/B testování

Při A/B testování jsou do provozu spuštěny dvě různé verze designu. Poté je zjišťováno, která ze dvou verzí se nejlépe osvědčila. Po desetiletí je to klasická metoda v přímém mailingu, kdy společnosti často rozdělují své seznamy adresátů a rozesílají různé verze mailingů různým příjemcům. A/B testování se stává populárním také na webu, kde je snadné zajistit, aby se různé verze stránek zobrazovaly různým návštěvníkům [40].

5.2.2 Pětisekundový test

Pětisekundové testy jsou metodou uživatelského výzkumu, která pomáhá změřit, co si uživatelé odnášejí. Zajímá se především o informace, které si uživatelé z prvních pěti sekund prohlížení designu odnáší. Dále se zabývá také dojmem, který v uživatelích zanechává. Běžně se používají k testování, zda webové stránky efektivně sdělují zamýšlené sdělení [41].

5.2.3 Uživatelské testování Admin Panelu

Celkově proběhla 1 iterace uživatelského testování admin panelu, kde byla testována použitelnost uživatelského rozhraní. Účastníkovi byly zadávány jednotlivé úkoly jako vytvoření nového článku nebo přidání nového obrázku do knihovny médií a jeho použití v článku. Otázky byly pokládány takovým způsobem, aby účastníkovi nebylo napovídáno, jakým způsobem má úkol splnit. Dále byly účastníkovi pokládány dodatečné otázky ohledně jednotlivých prvků, které na obrazovce vidí, co si myslí, že se stane, když na daný prvek klikne atd. Testování pomohlo odhalit 2 nedostatky admin panelu.

Na první obrazovce pro výběr workspacu nebylo zcela jasné, co workspace je, název byl příliš generický. Jako náprava byl název změněn na *Content workspaces*.

Druhým nedostatkem, který testování odhalilo, byla chyba při nahrávání do knihovny médií pomocí tlačítka *Add media*. Po nahrání prvního souboru již nešlo nahrát další soubor.

Kapitola 6

Postup pro nasazení

Pro zajištění stále stejného prostředí při nasazení bude využita platforma Docker. Veškeré naše aplikace budou sestaveny do Docker imagů a následně nasazeny do platformy Kubernetes. Repozitář je uložen na platformě Gitlab, která podporuje i kontinuální integraci, kterou využijeme.

6.1 Sestavení Docker imagů

Každá aplikace v adresáři *apps* bude sestavena do vlastního image. Z tohoto důvodu je potřeba, aby každá aplikace měla vlastní *package.json* soubor s jeho závislostmi. Nx tuto funkcionalitu má zabudovanou a stačí nastavit atribut `generatePackageJson` pro každý projekt v souboru *project.json*. Výsledek sestavení je uložen do složky `dist`.

```
nx run-many --target=build --projects=admin-panel,bohemia-net --parallel
```

Výpis kódu 16 Příkaz pro sestavení aplikací

Na ukázce kódu 17 je příklad, jak by mohl vypadat obsah Dockerfilu pro vytvoření Docker image.

```
FROM node:lts-alpine
WORKDIR /app
COPY ./dist/apps/bohemia-net .
ENV PORT=3010
EXPOSE \${PORT}
RUN pnpm install --prod
# zavislosti, ktere potrebuje NestJS
RUN pnpm install reflect-metadata tslib rxjs @nestjs/platform-express
CMD node ./main.js
```

Výpis kódu 17 Příklad Dockerfilu pro Bohemia.net API

6.2 Gitlab CI

Gitlab nám umožňuje spouštět joby při různých změnách v git repozitáři. Strategie, kterou je vhodné zvolit pro nasazení bude následující. Budou existovat primárně 2 prostředí, na která se bude nasazovat, a to produkční a testovací. Při merge request do `main` větve dojde k sestavení projektů, které byly změnami zasaženy (pomocí `nx affected`) a následně budou automaticky nasazeny na testovací prostředí. K nasazení na produkci dojde ve chvíli, kdy v repozitáři k danému commitu přidáme tag.

Kapitola 7
Závěr

Celá práce se zabývá tvorbou systému pro správu obsahu v rámci organizace. Cílem bylo analyzovat procesy a potřeby klienta v rámci tvorby obsahu, analyzovat stávající řešení, navrhnout nové řešení, implementovat prototyp a ten patřičně otestovat.

Analýza procesů ukázala, jakým způbem organizace s obsahem pracuje. Na základě této analýzy byly sestaveny funkční a nefunkční požadavky na systém pro správu obsahu. Následně byla provedena rešerše existujích řešení.

Na základě požadavků byl vytvořen návrh uživatelského rozhraní a návrh architektury. Součástí návrhu byla i rešerše Rich Text editorů, které by bylo vhodné použít v rámci implementace.

V implementační části byla provedena analýza metodik a technologií vhodných pro tvorbu webových API. Vhodnou technologií pro API bylo zvoleno GraphQL, které vývojářům webů nabízí jednotný způsob přístupu k obsahu. GraphQL API zpřístupňuje obsah pro jeden webový projekt Bohemia.net. Dále API generuje dodatečná metadata pro generování admin panelu. Spolu s API byl implementován prototyp admin panelu, který umožňuje spravovat daný obsah. Admin panel využívá systému introspekce pro získávání fieldů jednotlivých entit, které je možné editovat.

Na konci implementace bylo nutné prototyp otestovat na reálných uživatelích a ověřit tak jeho použitelnost. Proběhla 1 iterace uživatelského testování, které odhalilo 2 nedostatky v uživatelském rozhraní. Zároveň testování potvrdilo, že většina uživatelského rozhraní byla navržena tak, aby bylo použitelné a splňovalo dané cíle.

Celý systém je dále připraven na několik rozšíření. V budoucnu se počítá s širšími možnostmi filtrování, řazení a vyhledávání jednotlivých záznamů v kolekcích. Některé kolekce budou představovat kolekce webových stránek a proto bude nutné pro ně přidat rozšíření o fieldy potřebné pro SEO. Dále se plánuje implementovat vytváření relací mezi záznamy jednotlivých kolekcí, což umožní např. tvorbu kategorií nebo tagů.

Příloha A

Instalace a spuštění projektu

V této sekci jsou shrnuty všechny příkazy nutné k instalaci a spuštění celého projektu po naklonování repozitáře.

A.1 Základní požadavky

- Node.js v16.13.1
- pnpm 6.30.0
- MongoDB

A.2 MongoDB

Je nutné mít běžící instanci databáze MongoDB. Nejjednodušší je stáhnout si image pomocí Dockeru.

```
docker pull mongo
```

Výpis kódu 18 Příkaz pro stažení MongoDB pomocí Dockeru

```
docker run --name some-mongo
```

Výpis kódu 19 Příkaz pro spuštění instance server MongoDB

Dále je potřeba nastavit příslušné proměnné pro dané prostředí. Vytvoříme tedy soubor *.env* a pro lokální prostředí nám stačí zkopírovat obsah souboru *.env.example* do souboru *.env*, kde je uložen i řetězec pro připojení k databázi.

A.3 Instalace závislostí aplikace a spuštění

```
pnpm install
```

Výpis kódu 20 Instalace závislostí

```
pnpm nx run-many --target=seed --all
```

Výpis kódu 21 Naplnění databáze testovacími daty

```
pnpm nx run-many --target=generate-operations --all
```

Výpis kódu 22 Vygenerování GraphQL operací pro admin panel na základě schématu

```
pnpm nx run-many --target=serve --all --maxParallel=10
```

Výpis kódu 23 Spuštění všech aplikací

```
pnpm nx run bohemia-net
```

Výpis kódu 24 Spuštění aplikace samostatně

```
pnpm nx run-many --target=serve --projects=admin-panel,bohemia-net
```

Výpis kódu 25 Spuštění pouze vybraných aplikací

Bibliografie

1. *Headless*. San Francisco: Netlify, 2020. Dostupné také z: https://jamstack.org/glossary/headless-technology/.
2. RADAKOVIC, Nebojsa. *Git-based CMS vs. API-driven CMS: Which Headless CMS Should You Choose?* Wroclaw: Bejamas Group Sp. z o.o., 2021. Dostupné také z: https://bejamas.io/blog/git-based-cms-vs-api-first-cms/.
3. *Self-hosting (web services)*. San Francisco (CA): Wikimedia Foundation, 2001-2022. Dostupné také z: https://en.wikipedia.org/wiki/Self-hosting_(web_services).
4. *What is cloud computing?* Seattle: Amazon Web Services, 2013. Dostupné také z: https://aws.amazon.com/what-is-cloud-computing/.
5. *CI/CD*. San Francisco (CA): Wikimedia Foundation, 2001-2022. Dostupné také z: https://en.wikipedia.org/wiki/CI/CD.
6. *WordPress*. San Francisco (CA): Wikimedia Foundation, 2001-2022. Dostupné také z: https://cs.wikipedia.org/wiki/WordPress.
7. *Strapi*. Paris: Strapi, 2022. Dostupné také z: https://strapi.io/.
8. *KeystoneJS (Node.js CMS & Web Application Platform)*. San Francisco: Medium, 2018. Dostupné také z: https://medium.com/front-end-weekly/keystonejs-node-js-cms-web-application-platform-b069bd5a539f.
9. *Why Keystone*. Sydney: Thinkmill, 2021. Dostupné také z: https://keystonejs.com/why-keystone.
10. *PayloadCMS*. Grand Rapids (Michigan): Payload CMS LLC, 2022. Dostupné také z: https://payloadcms.com/.
11. NORMAN, Don; NIELSEN, Jakob. *The Definition of User Experience (UX)*. California: Nielsen Norman Group, 2013. Dostupné také z: https://www.nngroup.com/articles/definition-user-experience/.
12. *Návrh uživatelského rozhraní webové aplikace*. Praha: Vysoká škola ekonomická v Praze, 2018. Dostupné také z: https://kme.vse.cz/wp-content/uploads/page/534/10.-N%C3%A1vrh-u%C5%BEivatelsk%C3%A9ho-rozhran%C3%AD-webov%C3%A9-aplikace.pdf.
13. *Editor.js*. CodeX, [b.r.]. Dostupné také z: https://editorjs.io/.
14. *Draft.js*. Menlo Park, California: Meta Platforms, Inc., 2022. Dostupné také z: https://draftjs.org/.
15. *Headless WYSIWYG Text Editor – Tiptap Editor*. Berlin: Überdosis GbR, 2022. Dostupné také z: https://tiptap.dev/.

16. *Application Programming Interface*. London: Denis Howe, 1985. Dostupné také z: http://foldoc.org/Application+Programming+Interface.
17. BOOTH, David; VAAS, Hugo; MCCABE, Francis; NEWCOMER, Eric; CHAMPION, Michael; FERRIS, Chris; ORCHARD, David. *Web Services Architecture*. Cambridge (Massachuttes): World Wide Web Consortium, 2004. Dostupné také z: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest.
18. *XML Introduction*. San Francisco: Mozilla Foundation, 1998. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction.
19. *Working with JSON*. San Francisco: Mozilla Foundation, 1998. Dostupné také z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.
20. GUDGIN, Martin; HADLEY, Marc; MENDELSOHN, Noah; MOREAU, Jean-Jacques; NIELSEN, Henrik Frystyk; KARMARKAR, Anish; LAFON, Yves. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Cambridge (Massachuttes): World Wide Web Consortium, 1998. Dostupné také z: https://www.w3.org/TR/soap12-part1/#intro.
21. *What is a REST API*. Raleigh, Severní Karolína: Red Hat, 2022. Dostupné také z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.
22. *GraphQL*. San Francisco: GraphQL Foundation, 2015. Dostupné také z: https://spec.graphql.org/October2021/.
23. *Relational vs. NoSQL data*. Redmond, Washington: Microsoft, 2022. Dostupné také z: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data.
24. *Monorepo Explained*. Gilbert, Arizona: Narwhal Technologies Inc., 2022. Dostupné také z: https://monorepo.tools/#what-is-a-monorepo.
25. KOCHAN, Zoltan. *Why should we use pnpm?* San Francisco: Medium, 2018. Dostupné také z: https://medium.com/pnpm/why-should-we-use-pnpm-75ca4bfe7d93.
26. *Pnpm*. Pnpm, 2015. Dostupné také z: https://pnpm.io/.
27. *Intro to Nx*. Gilbert, Arizona: Narwhal Technologies Inc., 2022. Dostupné také z: https://nx.dev/getting-started/intro.
28. *TypeScript for the New Programmer*. Redmond, Washington: Microsoft, 2012. Dostupné také z: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.
29. *About Node.js*. San Francisco, California: OpenJS Foundation, 2019. Dostupné také z: https://nodejs.org/en/about/.
30. *Introduction to Node.js*. San Francisco, California: OpenJS Foundation, 2019. Dostupné také z: https://nodejs.dev/learn/introduction-to-nodejs.
31. *NestJS Documentation*. Kamil Mysliwiec, 2017. Dostupné také z: https://docs.nestjs.com/.
32. *What Is MongoDB?* New York: MongoDB, Inc., 2022. Dostupné také z: https://www.mongodb.com/what-is-mongodb.
33. *React*. Menlo Park, California: Meta Platforms, Inc., 2022. Dostupné také z: https://reactjs.org/.
34. *Tailwind CSS*. Tailwind Labs Inc., 2022. Dostupné také z: https://tailwindcss.com/.
35. *Testing*. Kamil Mysliwiec, 2017. Dostupné také z: https://docs.nestjs.com/fundamentals/testing.
36. HAMILTON, Thomas. *Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE*. Ahmedabad, India: Guru99 Tech Pvt Ltd, 2022. Dostupné také z: https://www.guru99.com/unit-testing-guide.html.

37. HAMILTON, Thomas. *END-To-END Testing Tutorial: What is E2E Testing with Example*. Ahmedabad, India: Guru99 Tech Pvt Ltd, 2022. Dostupné také z: https://www.guru99.com/end-to-end-testing.html.

38. *What is User testing*. Romania: Omniconvert, 2022. Dostupné také z: https://www.omniconvert.com/what-is/user-testing/.

39. MORAN, Kate. *Usability Testing 101*. Fremont, California: Nielsen Norman Group, 1998. Dostupné také z: https://www.nngroup.com/articles/usability-testing-101/.

40. NIELSEN, Jakob. *Putting A/B Testing in Its Place*. Fremont, California: Nielsen Norman Group, 1998. Dostupné také z: https://www.nngroup.com/articles/putting-ab-testing-in-its-place/.

41. *Five Second Test*. Australia: UsabilityHub, 2022. Dostupné také z: https://fivesecondtest.com/.

Obsah přiloženého média

```
/
├── readme.txt ..... stručný popis obsahu média
├── src
│   ├── impl ..... zdrojové kódy implementace
│   └── thesis ..... zdrojová forma práce ve formátu LaTeX
└── text ..... text práce
    └── thesis.pdf ..... text práce ve formátu PDF
```