**PANEURÓPSKA VYSOKÁ ŠKOLA**
**FAKULTA INFORMATIKY**

**FI- 184954-20862**

# NÁVRH A IMPLEMENTÁCIA CMS SYSTÉMU S VYUŽITÍM SERVISNE ORIENTOVANEJ ARCHITEKTÚRY

**Bakalárska práca**

**2024**

**Roman Zemaník**

**PANEURÓPSKA VYSOKÁ ŠKOLA**
**FAKULTA INFORMATIKY**

# NÁVRH A IMPLEMENTÁCIA CMS SYSTÉMU S VYUŽITÍM SERVISNE ORIENTOVANEJ ARCHITEKTÚRY

Bakalárska práca

**Roman Zemaník**

| | |
|---|---|
| Študijný program: | Aplikovaná informatika |
| Študijný odbor: | Informatika |
| Školiace pracovisko: | Ústav aplikovanej informatiky |
| Školiteľ: | Ing. Július Hlaváč, PhD. |

**Bratislava 2024**

**Cieľom práce je vytvorenie Content management systému na správu obsahu. CMS bude využívať servisne orientovanú architektúru s využitím rozhrania na báze REST/API s front-endom aplikácie.**

**V teoretickej časti je potrebné zanalyzovať dostupné technológie, vybrať si niektoré z nich a odôvodniť svoj výber. Praktická časť bude spočívať v implementácií systému s vybranými technológiami.**

**1. Popis SOA architektúry**

**2. Popis Content management systémov príklady ich využitia výhody a nevýhody**

**3. Popis technológií Potrebných pre implementáciu praktickej časti**

**3. Implementácia CMS systému**

**Poďakovanie**

Pri vypracovaní tejto práce by som sa rád poďakoval za odbornú pomoc pánovi prof./doc. X Y, ktorý mi poskytol mnoho cenných rád a konzultácií.

Poďakovanie môžete napísať podľa seba

**Čestné prehlásenie**

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

.....podpis

**Abstrakt**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer eu lacus leo. Nulla egestas purus non dignissim tincidunt. In sit amet tellus bibendum, lobortis magna ut, sodales mi. Etiam vel eros efficitur purus ultrices vulputate et quis justo. Nunc ultrices tellus a dui mattis, eget laoreet arcu tempor. Donec vestibulum, magna ac fringilla lacinia, libero risus fringilla arcu, nec consectetur nibh justo vel sem. Quisque gravida sit amet elit ut aliquam.

Abstrakt obsahuje informáciu o cieľoch práce, jej stručnom obsahu a v závere abstraktu sa charakterizuje splnenie cieľa, výsledky a význam celej práce. Súčasťou abstraktu je 3 - 5 kľúčových slov.

**Kľúčové slová:** CMS, Headless CMS, REST, API, NestJS, Back-End, HTTP, Databáza, PostgreSQL

**Abstract**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer eu lacus leo. Nulla egestas purus non dignissim tincidunt. In sit amet tellus bibendum, lobortis magna ut, sodales mi. Etiam vel eros efficitur purus ultrices vulputate et quis justo. Nunc ultrices tellus a dui mattis, eget laoreet arcu tempor. Donec vestibulum, magna ac fringilla lacinia, libero risus fringilla arcu, nec consectetur nibh justo vel sem. Quisque gravida sit amet elit ut aliquam.

**Keywords**: CMS, Headless CMS, REST, API, NestJS, Back-End, HTTP, Databáza, PostgreSQL

# Obsah

## Zoznam ilustrácií

## Zoznam skratiek a značiek

| | |
|---|---|
| **REST** | Representational State Transfer |
| **API** | Aplication Programming Interface |
| **CMS** | Content Management System |
| **JWT** | Jason Web Token |
| **HTTP** | Hyper Text Transfer Protokol |
| **URL** | Universal Resource Locator |

# Úvod

TODO

# 1 Analýza problematiky

V tejto kapitole sa pozrieme na účel CMS systémov, na problémy, ktoré CMS systémy pomáhaju riešiť, na ich rôzne typy a na už existujúce implementácie týchto systémov.

## 1.1 Čo je to CMS?

CMS po slovensky znamená systém na správu obsahu (Content Management System). Tento systém je webová aplikácia, cez ktorú sa dá spravovať obsah inéj webovej stránky alebo aplikácie. Obsah môže tvoriť text, obrázky, rôzne číselné hodnoty alebo metadata jednotlivých webových stránok ako napríklad popis a titulka, ktoré používajú vyhľadávače ako google alebo bing na vytvorenie výsledkov vyhľadávania. Tento obsah sa väčšinou nachádza v kombinácií vhodnej pre určtý druh objektu alebo sekcie, ktorú na stránke chceme zobraziť. Môže ísť o sekciu, ktorý potrebuje nadpis, text a obrázok alebo o produkt, ktorý potrebuje názov, obrázok, popis a cenu.

## 1.2 Prečo CMS?

Hlavným účelom CMS systémov je pomôcť ľudom jednoduchšie spravovať obsah na webových stránkach alebo aplikáciach. CMS vytvára príležitosť pre ľudí bez znalostí v oblasti programovania s webovými technológiami vytvárať, upravovať alebo mazať obsah. Úžitok v nich nájdu aj technicky zdatní ľudia, lebo im pomáha obsah spravovať podstatne rýchlejšie ako keby ho nemali. Bez CMS by človek musel ísť priamo do zdrojového kódu stránky, na to aby čokoľvek upravil alebo zmenil. Takéto systémy sa najčastejšie používajú na dynamické stránky, ktoré majú obsah, ktorý sa časom mení. Príkladom takýchto webov je e-shop, kde sa cez CMS môžu pridávať produkty, meniť ich ceny alebo upravovať ich popisy. Ďalším príkladom sú blogy, kde sa pravidelne pridávajú alebo menia články. CMS sa ale nehodí na každú web stránku. Napríklad jednoduché statické stránky, ktorých obsah sa nemení, z takéhoto systému nezískajú veľa úžitku.

## 1.3 Výhody a nevýhody CMS

Každý systém ponúka výhody a nevýhody a CMS systémy nie sú výnimkou.

### 1.3.1 Výhody CMS

- Rýchlejšia a jednoduchšia úprava obsahu
- Stránku dokáže upravovať aj človek bez technických znalostí
- CMS za náš vyrieši všetku serverovú logiku
- Užívateľ sa môže sústrediť čisto na obsah
- Optimalizáca SEO (Search Engine Optimization)

- Ušetrenie penazí - s CMS systémom nemúsime platiť vývojárov, za každú zmenu, ktorú chceme na stránke vykonať

### 1.3.2 Nevýhody CMS

- Obmedzené možnosti - ak technické znalosti máme a chceme na stránku pridať niečo, čo CMS nepodporuje, tak sa buď musíme spoliehať na to, že existuje rozšírenie, ktorý náš problém rieši alebo si ho musíme napísať sami. Napísanie vlastného rozšírenia, ale nemusí byť vôbec jednoduché a môže zabrať o dosť viac času ako by nám trvalo priame upravenie kódu stránky

- Údržba - pri používani CMS systému sa spoliehame na to, že vývojári, ktorý CMS systém vyvíjajú, sa starajú o pravidelné aktualizácie systému na to, aby držal krok s novými technológiami alebo s novo objavenými internetovými hrozbami

- Cena - niektoré CMS systémý alebo ich rozšírenia stoja peniaze

- Optimalizácia - CMS systémy sú vyvíjane tak, aby splnili požiadavky, čo najväčšieho množstva zákazníkov a to môže spôsobiť, že pre náš konkrétny prípad bude stránka, ktorá používa CMS horšie optimalizovaná ako vlastné riešenie

## 1.4 Typické funkcie CMS

Každý CMS systém je unikátny, ale veľa z nich ponúka rovnaké základné funkcie, ktoré si predstavíme v tejto časti.

### 1.4.1 WYSIWYG Editor

Jednou typickou súčasťou každého CMS je WYSIWYG editor, cez ktorý sa dá napísať text rovnako ako napr. vo Worde a tento text CMS premení na HTML elementy, ktoré sa dajú zobraziť na stránke.

### 1.4.2 Správa súborov

Ďalšou funkciu, ktorú ponúka každé CMS je správa súborov. Táto funkcia nám umožnuje cez CMS vkladať rôzne súbory na server ako napr. obrázky alebo dokumenty. Veľa CMS systémov taktiež ponúka možnosť na tieto súbory odkazovať. Príkladom by bolo odkazovanie na obrázok, ktorý použijeme v článku.

### 1.4.3 Správa užívateľov

Správa užívateľov je dôležitá funkcia, ktorú využívajú hlavne stránky, na ktorých pracuje tím ľudí. Tento systém umožnuje napr. vytváranie účtu pre nového človeka v tíme a určenie jeho právomocí. Takto vieme nastaviť kolegovi zodpovednému za správu článkov právomoci na úpravu len článkov a kolegovi, ktorý má na starosti e-shop len úpravu produktov. Vieme taktiež nastaviť, aby niektoré nastavenie v CMS boli dostupné len pre jedného administrátora, aby sa nestalo, že niekto omylom CMS nastaví zle a spôsobí napr. nedostupnosť stránky.

### 1.4.4 Jazyky

CMS nám vie pomôct, keď máme stránku, ktorá má byť dostupná v rôznych jazykoch. Väčšina CMS poskytuje možnosť ukladať text v rôznych jazykoch a nasledovne ho zobrazovať podľa výberu návštevníka stránky.

### 1.4.5 Analytika

Veľa CMS systémov poskytuje nástroje na sledovanie analytiky. Analytika nám poskytuje rôzne informácie o stránke. Medzi niektoré z týchto informácií patrí počet návštevníkov stránky, počet ľudí, ktorí vyplnili určitý formulár alebo priemerný čas strávený na určitých podstránkach. Túto funkciu vie využiť napr. marketingové oddelenie na to, aby zistilo ako sa darí určitým produktov alebo kampaniam.

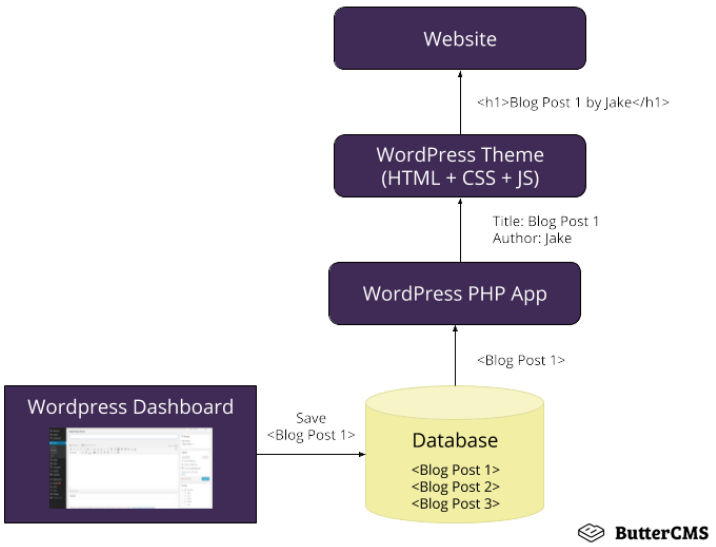
## 1.5 Typy CMS

Existuje veľa rôznych implementácií systémov na správu obsahu, ale všetky sa dajú zredukovať do jednej z dvoch kategórií, ktoré si v tejto kapitole predstavíme.

https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms

### 1.5.1 Klasické CMS

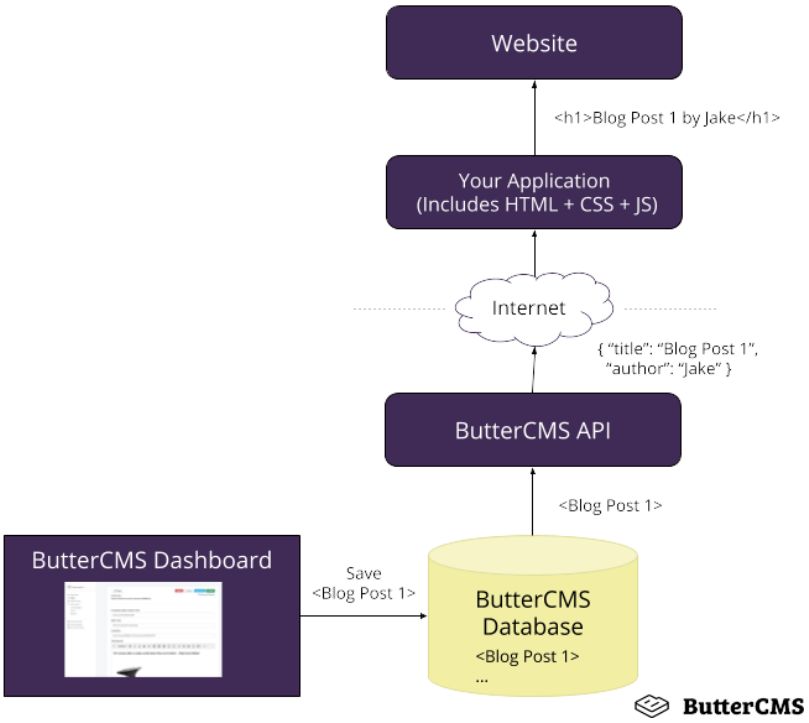
Klasicky sa CMS používalo len na správu webových stránok. V posledných rokoch sa spolu s rastúcim počtom zariadení, na ktoré chceme zobraziť svoj obsah, zvyšuje aj popularita rie Toto riešenie ale nemúsi byť postačujúce pre firmy, ktoré chcú svoj obsah dostať na mnoho rôznych a stále pribúdajúcich zariadení, ktor




Obrázok 1, Architektúra klasického CMS

### 1.5.2 Headless CMS

Headless CMS je navrhnutý tak, aby poskytoval obsah bez akýchkoľvek návrhových alebo prezentáčných vrstiev. To znamená, že headless CMS sa zameriava výlučne na správu obsahu a poskytuje ho cez API (bez pripojeného front-endu). Tento prístup dáva vývojárom a obsahovým tvorcom väčšiu flexibilitu pri implementácii rôznych typov prezentácií, ako sú webové stránky, mobilné aplikácie, alebo iné digitálne prostredia. Použitie headless CMS môže tiež zjednodušiť proces aktualizácie a distribúcie obsahu, pretože sa obsah ukladá na jednom mieste a môže byť ľahko dostupný pre viacero kanálov.




Obrázok 2, Architektúra Headless CMS

# 2 Použité technológie

V tejto kapitole si predstavíme všetky relevantné technológie, postupy alebo štandardy potrebné na implementáciu môjho CMS systému.

## 2.1 URL

URL je spolu s Hypertextom a HTTP jeden z kľúčových konceptov URL slúži na lokalizáciu zdrojov na internete. Vďaka URL vieme, kde sa hľadaný zdroj nachádza a ako k nemu pristupovať. URL sa dá chápať ako adresa určitého unikátneho zdroja nachádzajúceho sa na internete. Tento unikátny zdroj môže byť HTML dokument, obrázok alebo štruktúrované data v jednom z používaných formátov ako napríklad JSON alebo XML. V mojej práci sa URL používa hlavne v kontexte s dátami formátovanými v JSON formáte, ktorý si vysvetlíme v neskoršej kapitole.

URL sa skladá z nasledujúcich troch povinných komponentov a z jedného nepovinného:

1. Schéma - identifikuje protokol použitý pri získani zdroja z internetu.
2. Host (autorita) - identifikuje server, ktorý hostuje zdroj. Za názvom hosta sa môže ale nemusí nachádzať aj číslo portu (napr. www.priklad.sk:80).
3. Cesta - identifikuje špecifický zdroj, ku ktorému chce klient pristúpiť na hostovom servery.
4. Dopytový reťazec (Query string) - obsahuje parametre, ktoré chceme hostitelovi poslať. Tieto parametre sa nachádzajú za cestou, za znakom ? a sú v pároch názov-hodnota. Viacero hodnôt vieme spojiť znakom ampersand (&). Parametre vie využiť server na vykonanie rôznych operácií. Tento komponent URL adresy nie je povinný.

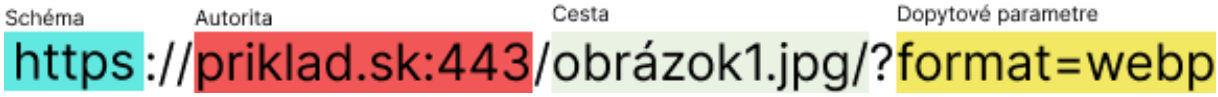
Konkrétny príklad uvádzam na obrázku (Obrázok 3), na ktorom môžeme vidieť všetky spomenuté komponenty.

Ako **schéma** sa uvádza zabezpečený HTTP protokol, čo nášmu zariadeniu hovorí, že chceme vykonať HTTP požiadavku. Iné možnosti by boli napríklad ftp:// na komunikáciu pomocou FTP (Protokol na prenos súborov) alebo mailto: na otvorenie emailového klienta.

**Autorita** je príklad.sk, čo sa počas požiadavky preloží na IP adresu, pomocou ktorej vieme nájsť hladaný server. Druhou častou autority môže byť port. Port je často určený implicitne podľa vybranej schémy. Napríklad schéma HTTPS implicitne doplní port 443 alebo nezabezpečený HTTP protokol doplní port 80.

**Cesta** nám hovorí, že hľadaný obrázok sa na servery nenachádza v žiadnom priečinku a volá sa obrázok1.jpg. Keby sa obrázok nachádza v priečinku obrázky, tak by cesta bola /obrázky/obrázok.jpg. [1]

V tomto prípade **dopytové parametre** určujú kľúč *formát* s hodnotou *webp*, vďaka čomu vie server zistiť, že chceme obrázok konvertovať do formátu webp. K URL by sme vedeli pridať aj ďalší parameter ako napríklad sirka=250 na to, aby zmenil šírku obrázka na 250 pixelov. Výsledne parametre by vyzerali nasledovne format=webp&sirka=250.




Obrázok 1, Príklad URL adresy

## 2.2 HTTP

Hypertextový prenosový protokol (angl. hypertext transfer protokol) je protokol pôvodne slúžiaci na prenos HTML dokumentov cez internet medzi klientom a serverom. Dnes sa používa na prenos rôznych typov súborov ako napr. obrázkov, videí, fontov a mnoho ďalších. Prvá verzia tohto protokolu bola navrhnutá v roku 1989 Timom Berners-Leeom [2]. V nasledujúcich

podkapitolách si prejdeme hlavné koncepty HTTP protokolu, ktoré sú relevantné pre implementáciu CMS. [3]

### 2.2.1 Klient-server model

HTTP sa riadi klasickým modelom klient-server, ktorý je všadeprítomný vo svete webových technológií. Klient-server model funguje na základe požiadaviek a odpovedí. Klient pošle požiadavku na server, ktorý požiadavku spracuje a pošle odpoveď. Väčšinou keď hovoríme o klient-server modeli v súvislosti s HTTP, tak klientom sa myslí webový prehliadač a serverom je kód, ktorý beží na počítači pripojenom na internet. [3]

### 2.2.2 Zdroje

Web servery sú hostitelmi rôznych typov dát, ku ktorým môže klient pristupovať. Tieto dáta sa v súvislosti s HTTP nazývajú tiež zdroje (resources). Najednoduchším typom zdroja je statický súbor uložený v súborovom systeme servera. Toto môže byť napr. HTML súbor, PDF, obrázok alebo mnoho iných. [3]

### 2.2.3 Požiadavka

Požiadavku skoro vždy posiela klient na server. Cieľom požiadavky je prístup k zdroju, ktorý sa nachádza na serveri. Na to, aby sme určili, ku ktorému zdroju chceme pristúpiť, musíme v požiadavke definovať cestu k zdroju, HTTP metódu a hlavičky a prípadné telo požiadavky. [3]

### 2.2.4 HTTP metódy

HTTP protokol poskytuje mnoho rôznych metód, ktoré sa využívajú pri tvorení požiadaviek. Metódy si vyberáme podľa požadovanej akcie, ktorú chceme vykonať pre daný zdroj. [3]

#### 2.2.4.1 GET

Metóda GET sa používa na získanie zdrojov zo servera. Táto metóda je bezpečná, idempotentá a cachovatelná. To že je bezpečná znamená, že táto metóda nevyvolá žiadnu zmenu na strane servera. Idempotentnosť znamená, že opakované požiadavky budú mať vždy rovnaký výsledok ako jedna požiadavka a to že je cachovatelná označuje možnosť ukladania odpovedí do cache. [3]

#### 2.2.4.2 POST

Používa sa na uloženie zdrojov na server. Keďže mení stav zdrojov na servery, tak nie je bezpečná a ani idempotentá. Pri určitej konfigurácií môže byť cachovatelná, ale toto je zriedkavo implementované kvôli nedostatočnej podpore prehliadačov.

https://developer.mozilla.org/en-US/docs/Glossary/Cacheable

#### 2.2.4.3 DELETE

Účelom tejto metódy je vymazanie zdrojov zo servera. Nie je bezpečná, lebo odstráni zdroj zo servera. Je idempotentná, lebo prvá požiadavka zdroj vymaže a pri nasledovných požiadavkach je už zdroj vymazaný, takže stav na servery sa nemení. Odpoveď na DELETE požiadavku sa nedá uložiť do cache.

#### 2.2.4.4 PUT

Posledná metóda, ktorú použijem v implementácií CMS systém je metóda PUT, ktorá slúži na zmenu stavu existujúcich zdrojov. Môže ísť napríklad o zmenu textu v existujúcom komponente. Nie je bezpečná a cachovatelná, ale je idempotentná.

### 2.2.5 Stavové kódy

HTTP stavové kódy sa posielajú spolu s odpoveďou na požiadavku a ich účelom je poskytnúť informáciu o výsledku operácie. Delia sa na 5 kategórií.

#### 2.2.5.1 Informačné kódy (100 - 199)

Poskytujú informáciu o stave servera. Napríklad kód 100 (Continue) sa môže použiť, keď klient chce poslať veľký súbor a pred poslaním súboru chce vedieť, či má server dostatočnú kapacitu.

#### 2.2.5.2 Úspech (200 - 299)

Označuje, že žiadaná operácia prebehla úspešne. Patrí sem napríklad kód 201 (Created), ktorý označuje úspešné vytvorenie zdroja na servery.

#### 2.2.5.3 Presmerovanie (300 - 399)

Odpovede s týmto kódom podávajú informáciu o tom, že naša požiadavka bola presmerovaná na inú adresu. Príkladom je kód 302 (Found), čo znamená, že požadovaný zdroj bol dočasne presunutý na inú adresu.

#### 2.2.5.4 Chyba na strane klienta (400 - 499)

Indikujú chybu na strane klienta. Najčastejšie sa jedná o chyby s kódom 404 (Not Found), čo nám hovorí o tom, že zdroj, ku ktorému chceme pristupovať neexistuje na danej adrese alebo 401 (Unauthorized), keď chceme pristupvoať ku zdroju bez potrebných právomocí.

#### 2.2.5.5 Chyba na strane servera (500 - 599)

Posledná kategória HTTP stavových kódov nám hovorí o chybe, ktorá nastala na strane servera. Väčšinou sa posiela kód 500 (Internal Server Error), čo je všeobecný kód, ktorý sa používa, keď sa iný špecifickejší kód nehodí na označenie chyby.

## 2.3 JSON (JavaScript Object Notation)

JSON je jednoduchý dátový formát odvodený od spôsobu definovania objektov v JavaScripte. Používa sa pri prenose dát medzi rôznymi systémami.

Príklad odpovede webovej služby na získanie informácií o počasí formátovanej v JSON formáte.

```
{
    "zemepisna_sirka": 48.1439,
    "zemepisna_dlzka": 17.1097,
    "predpovede": [
        {
            "datum": "09.03.2024",
            "popis": "Zamračené",
            "nebezpečie": false
        },
        {
            "datum": "10.03.2024",
            "popis": "Slnečno",
            "nebezpečie": false
        }
    ]
}
```

## 2.4 REST

REST je akronym pre **RE**presentational **S**tate **T**ransfer prvý krát definovaný počítačovým vedcom Dr. Roy Fieldlingom v jeho dizertačnej práci [4]. REST ustanovuje architektektonický štýl používaný pri implementácií webových služieb alebo inak povedané webové API. Webové služby, ktoré su implementované podľa REST architektúry sa taktiež nazývajú RESTful APIs. Príkladmi webových služieb, ktoré implementujú REST princípy sú Twilio, ktoré poskytuje službu na posielane SMS správ cez ich RESTful API alebo Stripe, ktorý poskytuje RESTful API na vykonávanie platieb cez internet. [5]

### 2.4.1 Princípy REST

Na to aby sa webová služba mohla nazývať RESTful, tak musí spĺnať 5 princípov, ktoré si teraz priblížime.

#### 2.4.1.1 Jednotné rozhranie

Tento princíp hovorí o tom, že server poskytuje klientovy všetky dostupné zdroje jednotným a predvídatelným spôsobom. Toto sa dá dosiahnuť pomocou jednotného názvoslovia pri tvorení zdrojov a využívania HTTP metód. Naša API

#### 2.4.1.2 Bezstavovosť

Bezstavovosť vyžaduje, aby každá požiadavka od klienta na server obsahovala všetky potrebné informácie na úspešné vykonanie požadovanej akcie.

#### 2.4.1.3 Cachovatelnosť

Tento princíp požaduje, aby odpovede z API boli označené ako cachovatelné alebo necachovatelné. Ak je odpoveď cachovatelná, tak klient má možnosť po určitý čas použiť dáta, ktoré boli odpovedou na tú špecifickú požiadavku.

#### 2.4.1.4 Klient-server

Preto aby API bola RESTful musia byť rozdelené vrstvy na uživateľské rozhranie (klient) a

#### 2.4.1.5 Vrstvovaný systém

Server musí byť navrhnutý vo vrstvách. Bežný príklad vrstvovaného systému je MVC architektúra, kde sa kód delí na View (užívateľské rozhranie/prezenčná vrstva), Model (dátová vrstva) a Controller (vrstva, ktorá prepája View a Model).

## 2.5 JavaScript

JavaScript je interpretovaný programovací jazyk, ktorý vznikol za účelom pridania reaktivity na webové stránky. Bol vytvorený Brendanom Eichom v spoločnosti Netscape Communications pod prvotným názvom Mocha v roku 1995. Napriek jeho názvu neexistuje žiadny vzťah medzi programovacím jazykom Java a JavaScriptom. Aj keď bol jazyk pôvodne vyvinutý ako intepretovaný jazyk, tak v roku 2008 Google vytvoril V8 engine, ktorý dokáže JavaScript skompilovať do bytecodu a nasledovne ho

## 2.6 TypeScript

TypeScript je nadstavba jazyka JavaScript, ktorá pridáva statické typovanie a ďalšie pokročilé funkcie k pôvodnému JavaScriptu. To znamená, že TypeScript umožňuje definovať typy pre premenné, parametre funkcií a návratové hodnoty, čím pomáha vývojárom zachytávať chyby a zlepšovať ľahkú čitateľnosť a údržbu kódu. Kód napísaný v TypeScripte sa kompiluje do kódu JavaScript, čo umožňuje jeho použitie vo všetkých moderných webových a aplikáciách. TypeScript sa často používa v projektoch so zložitejšou logikou alebo veľkým počtom vývojárov, aby sa zlepšila prehľadnosť a predvídateľnosť kódu.

## 2.7 NestJS

NestJS je Node.js framework s účelom uľahčenia vývoja škálovateľných serverových aplikácií. Plne podporuje TypeScript a je v ňom aj implementovaný, ale dovoľuje používanie klasického

JavaScriptu. Kombinuje aspekty Objektovo-Orientovaného Programovania a Funkcionálneho Programovania. [6]

## 2.8 PostgreSQL

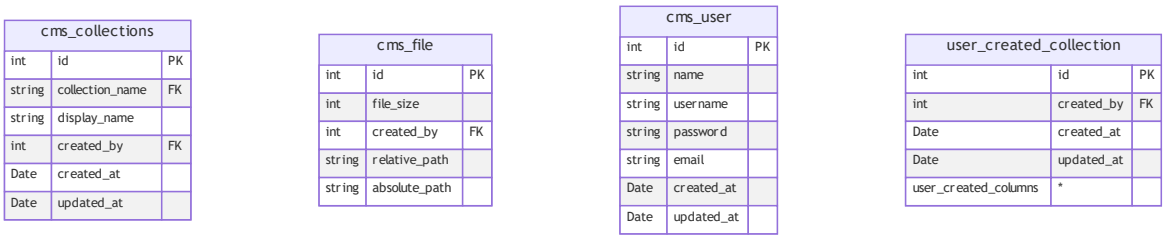
PostgreSQL je open-source relačný databázový systém, ktorý sa vyznačuje robustnosťou, flexibilitou a rozšírenými možnosťami. Je navrhnutý na podporu veľkých objemov dát a zároveň poskytuje pokročilé funkcie, ako sú transakcie s ACID vlastnosťami, rôzne typy indexov, podpora pre geografické dáta a mnoho ďalších rozšírení. PostgreSQL je známy svojou schopnosťou integrovať sa s rôznymi programovacími jazykmi a technológiami a je často využívaný v širokom spektre aplikácií od malých projektov až po veľké podnikové systémy.

## 2.9 JWT (JSON Web Token)

JSON Web Token (JWT) je open standard (RFC 7519), ktorý definuje kompaktný a zabezpečený spôsob prenášania informácií medzi stranami vo forme JSON objektu. Tieto informácie môžu byť ľubovoľné dáta, ktoré je možné digitálne podpísať alebo zašifrovať. JWT sa často používa na autentifikáciu a overovanie identity medzi klientom a serverom, ako aj na zdieľanie údajov o užívateľovi medzi rôznymi službami v decentralizovaných systémoch. Je zložený zo troch častí: hlavičky (header), payloadu a podpisu (signature), ktoré sú oddelené bodkočiarkami. Hlavička obsahuje informácie o type tokenu a použitom algoritme šifrovania, payload obsahuje samotné dáta tokenu a podpis sa používa na overenie integritu a autentickosť tokenu. JWT môže byť podpísaný (signed) alebo zašifrovaný (encrypted) na zvýšenie bezpečnosti.

# 3 Návrh systému

## 3.1 Fyzický model databázy




## 3.2 Koncové body

Koncové body definované v mojej impmentácií Headless CMS systému.

### 3.2.1 Kolekcie

/collection (POST)

/collection/all (GET)

/collection/:collection (DELETE)

### 3.2.2 Položky

/item/:collection (POST)

/item/:collection/:id (DELETE)

### 3.2.3 Atribúty

/attribute/:collection (POST)

/attribute/:collection/:columnName (DELETE)

### 3.2.4 Autentikácia

/auth/login (POST)

/auth/register (POST)

/auth/profile (GET)

### 3.2.5 Súbory

/file/upload (POST)

# 4 Implementácia systému

# Záver

Nezabudnite na Záver a zhodnotenie,

Cieľ bakalárskej práce bol splnený.

# Použitá literatúra


[1] Mozilla Corporation, „developer.mozilla.org,“ [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL.

[2] T. Berners-Lee. [Online]. Available: https://www.w3.org/History/1989/proposal.html.

[3] D. G. &. B. Totty, HTTP: The Definitive Guide, O'Reilly Media, 2002.

[4] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures,“ UNIVERSITY OF CALIFORNIA, CALIFORNIA, IRVINE, 2000.

[5] L. R. a. M. Amundsen, RESTful Web APIs, O’Reilly Media, Inc.,, 2013.

[6] K. Mysliwiec, „www.nestjs.com,“ [Online]. Available: https://docs.nestjs.com/.

[7] Kontent.ai, „State of the Headless CMS Market,“ 2023.