Stredná priemyselná škola elektrotechnická  
Hálova 16, 851 01 Bratislava

**AUTOBAHN.SK - BACKEND WEBOVEJ APLIKÁCIE**

KOMPLEXNÁ ODBORNÁ MATURITNÁ PRÁCA

Bratislava, 2024 4.D Martin Mojžiš

Stredná priemyselná škola elektrotechnická  
Hálova 16, 851 01 Bratislava

**AUTOBAHN.SK - BACKEND WEBOVEJ APLIKÁCIE**

KOMPLEXNÁ ODBORNÁ MATURITNÁ PRÁCA

Štúdijný odbor: 2573M programovanie digitálnych technológií  
Konzultant: Ing. Dominik Zatkalík, PhD.

Bratislava, 2024 4.D Martin Mojžiš

<SEM VOLZITE ZADANIE, PODPISANE>

**Čestné vyhlásenie**

Ja, dolupodpísaný, študent 4. D triedy Strednej priemyselnej školy elektrotechnickej Hálova 16, 851 01 Bratislava, týmto vyhlasujem, že som túto prácu vyhotovil sám, s použitím uvedenej literatúry a podľa rád môjho konzultanta.

.........................................................

V Bratislave, <dd. mm. rrrr> Martin Mojžiš

**Poďakovanie**

Rád by som sa touto cestou poďakoval svojmu <školiteľovi> za prístup a odborné rady. Tiež by som sa rád poďakoval <spoločnosti> za finančnú podporu pri realizácii praktickej časti mojej práce. PODĽA VÁŠHO UVÁŽENIA

**Abstrakt:**

**Kľúčové slová:**

**Abstract:**

**Keywords:**

**Obsah**

[ÚVOD 6](#_Toc188258470)

[1 Teoretické východiská 7](#_Toc188258471)

[1.1 Štruktúra webových aplikácií 7](#_Toc188258472)

[1.1.1 Frontend – Užívateľská vrstva 7](#_Toc188258473)

[1.1.2 Backend – Logická vrstva 8](#_Toc188258474)

[1.1.3 Databáza – Úložisko dát 8](#_Toc188258475)

[1.1.4 API – Prepojenie vrstiev 8](#_Toc188258476)

[1.2 Vývojový plán webovej aplikácie 9](#_Toc188258477)

[1.2.1 Brainstorming nápadov 9](#_Toc188258478)

[1.2.2 Tvorba funkčnej špecifikácie 9](#_Toc188258479)

[1.2.3 Tvorba wireframes 10](#_Toc188258480)

[1.2.4 Plánovanie roboty 10](#_Toc188258481)

[1.3 Analýza backendových technológií vhodných na tvorbu webových aplikácií 10](#_Toc188258482)

[1.3.1 ASP .NET (C#) 10](#_Toc188258483)

[1.3.2 Spring Boot (Java) 11](#_Toc188258484)

[1.3.3 Django (Python) 11](#_Toc188258485)

[1.3.4 Ruby on Rails (Ruby) 11](#_Toc188258486)

[1.3.5 Express (Node.js) 12](#_Toc188258487)

[1.3.6 Laravel (PHP) 12](#_Toc188258488)

[1.4 Typy databáz a ich využitie pri vývoji webových aplikácií 12](#_Toc188258489)

[1.4.1 Relačné databázy (SQL) 13](#_Toc188258490)

[1.4.2 Nerelačné databázy (NoSQL) 13](#_Toc188258491)

[1.5 Backendové technológie použité pri tvorbe webovej aplikácie 14](#_Toc188258492)

[1.5.1 OctoberCMS: Developer-friendly framework 14](#_Toc188258493)

[1.5.2 MySQL: Relačná databáza 15](#_Toc188258494)

[1.5.3 BugSnag: Analytika a logovanie chýb 15](#_Toc188258495)

[2 Návrh riešenia 16](#_Toc188258496)

[2.1 Vytvorenie backendovej časti webovej aplikácie 16](#_Toc188258497)

[2.1.1 Setup prostredia 17](#_Toc188258498)

[2.1.2 Setup úložiska 19](#_Toc188258499)

[2.1.3 Vytvorenie pluginov 19](#_Toc188258500)

[2.1.4 Vytvorenie modelov 20](#_Toc188258501)

[2.1.5 Vytvorenie HTTP ovládačov 20](#_Toc188258502)

[2.1.6 Užívatelia a autentifikácia 20](#_Toc188258503)

[2.2 Technická architektúra backendovej časti webovej aplikácie 20](#_Toc188258504)

[2.3 Databázu webovej aplikácie 21](#_Toc188258505)

[2.4 Zverejnenie webovej aplikácie na doménu a jej nasadenie na server 21](#_Toc188258506)

[2.5 Podnikateľský plán webovej aplikácie 21](#_Toc188258507)

[2.6 Marketingový plán a stratégia webovej aplikácie 21](#_Toc188258508)

[3 Záver 23](#_Toc188258509)

[Zoznam použitej literatúry 24](#_Toc188258510)

[Prílohy <volitelne> 7](#_Toc188258511)

[Príloha A – Zdrojový kód I](#_Toc188258512)

**Zoznam skratiek, značiek a symbolov**

API - Application Programming Interface

WWW - World Wide Web

IP - Internet Protocol

PHP - Hypertext Preprocessor

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

CMS - Content Management System

MVP - Minimum Viable Product

CLI - Command-line interface

IoT - Internet of Things

JSON - JavaScript Object Notation

CORS - Cross-origin resource sharing

AWS – Amazon Web Services

**Zoznam tabuliek, grafov a ilustrácií**

<Zoznam skratiek, značiek a symbolov>

# ÚVOD

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.

Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

# Teoretické východiská

Webová aplikácia sa najčastejšie chápe ako HTML stránka, ktorá sa zobrazuje používateľom vo webových prehliadačoch na zariadeniach pripojených k internetu. Tieto stránky sú uložené na vzdialených serveroch, ktoré sú prostredníctvom internetovej siete prepojené s konečnými užívateľmi. Každý server, rovnako ako akékoľvek iné zariadenie v sieti, má svoju vlastnú **IP adresu**. Väčšinou sú však tieto servery identifikované prostredníctvom domén (napríklad google.com) alebo subdomén (napríklad docs.google.com).

V bežnej reči sa však pod pojmom webová stránka často myslí celý systém, ktorý zabezpečuje jej fungovanie. Vizuálna časť, ktorú vidí používateľ, je len malou časťou celého systému. Na pozadí prebieha väčšinou komplexná logika, ktorá pracuje s dátami, ktoré užívatelia buď poskytujú, alebo ich na stránke potrebujú. Okrem toho existujú aj jednoduchšie prezentačné stránky, ktoré slúžia na prezentáciu organizácie alebo produktu. Takéto stránky sú zvyčajne statické, pretože ich obsah sa často nemení.

Na druhej strane sú dynamické webové stránky, ktoré svoj obsah získavajú z backendu. Tento backend spracúva dáta a odosiela ich na vykreslenie. Typickými príkladmi dynamických stránok sú internetové obchody alebo stránky novín. V oboch prípadoch je základom stránky obsah, ktorý spravujú administrátori. Na to, aby mohli obsah ľahko upravovať, sa dnes často používajú systémy na správu obsahu (CMS). Administrátori tak môžu prostredníctvom formulárov jednoducho vytvárať blogové príspevky, produkty a ďalší obsah. Navyše majú možnosť nastaviť titulné obrázky či iné detaily.

## Štruktúra webových aplikácií

Webové aplikácie sú komplexné systémy, ktoré pozostávajú z viacerých prepojených častí. Ich štruktúra je navrhnutá tak, aby umožnila plynulú komunikáciu medzi jednotlivými komponentmi a zabezpečila požadovanú funkcionalitu. Táto štruktúra sa zvyčajne skladá z troch hlavných vrstiev: frontend, backend a databáza.

### Frontend – Užívateľská vrstva

Frontend je časť aplikácie, ktorú vidí a s ktorou interaguje užívateľ. Zodpovedá za vizuálne prvky, ako sú rozloženie, farby, texty, tlačidlá a formuláre. Táto vrstva je zvyčajne naprogramovaná pomocou technológií ako HTML, CSS a JavaScript. Moderné frontend frameworky ako React, Vue.js alebo Angular umožňujú vývojárom vytvárať rýchle, responzívne a dynamické používateľské rozhrania.

Frontend komunikuje s backendom prostredníctvom API (Application Programming Interface), aby získal dáta alebo poslal požiadavky. Napríklad po kliknutí na tlačidlo na odoslanie formulára sa tieto údaje odosielajú na backend, kde sú ďalej spracované.

### Backend – Logická vrstva

Backend je motorom každej webovej aplikácie. Ide o časť, ktorá spracúva požiadavky od frontendovej vrstvy, vykonáva zložité výpočty alebo logiku, a zabezpečuje komunikáciu s databázou, preto sa backend často nazýva aj ako business logika aplikácie. Backendové systémy sú programované v rôznych jazykoch, ako sú PHP, Python, Node.js, C# alebo Java.

Na prehľadnú organizáciu backendového kódu sa často využívajú frameworky, ako napríklad Laravel, Django, Express.js, Spring alebo .NET Core. Tieto nástroje pomáhajú vývojárom implementovať funkcionalitu rýchlejšie a s menším rizikom chýb.

Backend tiež zabezpečuje bezpečnosť aplikácie, autentifikáciu užívateľov a ochranu dát. Napríklad pri prihlasovaní do aplikácie backend overí zadané údaje a vygeneruje token, ktorý užívateľovi umožní prístup k ďalším funkcionalitám.

### Databáza – Úložisko dát

Databáza je miesto, kde sú uložené všetky dáta potrebné pre fungovanie aplikácie. Môže ísť o údaje o užívateľoch, produkty, transakcie alebo akýkoľvek iný obsah, ktorý aplikácia spravuje. Najčastejšie sa používajú dva druhy databáz:

1. Relačné databázy – ako napríklad MySQL, PostgreSQL alebo Microsoft SQL Server, ktoré organizujú dáta do tabuliek a umožňujú ich prepojenie pomocou vzťahov.
2. NoSQL databázy – ako MongoDB alebo Cassandra, ktoré sú flexibilnejšie a používajú sa najmä pri práci s veľkým množstvom nestruktúrovaných dát.

Databáza komunikuje s backendom, ktorý od nej žiada dáta alebo do nej zapisuje nové informácie.

### API – Prepojenie vrstiev

Komunikácia medzi frontendom, backendom a databázou prebieha pomocou API. API definuje pravidlá, ako si jednotlivé vrstvy môžu vymieňať dáta. Najčastejšie sa používajú REST API alebo GraphQL, ktoré umožňujú rýchlu a efektívnu výmenu informácií medzi systémami.

## Vývojový plán webovej aplikácie

Pri vývoji webovej aplikácie je dôležité venovať dostatok času plánovaniu, ktoré zabezpečí hladký priebeh celého procesu. Kľúčovým krokom je rozdelenie celkového plánu na jednotlivé fázy, ktoré umožnia postupný vývoj aplikácie, minimalizujú riziko nejasností a problémov počas implementácie. Tieto fázy zahŕňajú brainstorming nápadov, tvorbu funkčnej špecifikácie, vytvorenie wireframes a podrobný plán práce.

### Brainstorming nápadov

Prvým krokom vo vývoji aplikácie je brainstorming, počas ktorého sa zameriavame na zhromažďovanie a analyzovanie nápadov. Ide o veľmi dôležitú fázu, kde je potrebné zamyslieť sa nad cieľom aplikácie, jej používateľmi, požiadavkami a funkciami. Zameriavame sa na všetky možnosti a snažíme sa nájsť najefektívnejšie riešenia. Môžeme využiť rôzne techniky ako diskusie, analýzu konkurencie alebo spätnú väzbu od potenciálnych používateľov, aby sme pochopili ich potreby a očakávania.

Tento proces nám pomáha definovať hlavné ciele aplikácie a určiť, aké funkcie a vlastnosti by mala mať, aby bola úspešná. Na základe týchto nápadov a postrehov vytvoríme zoznam funkcií, ktoré sa následne stanú základom pre ďalšie fázy vývoja.

### Tvorba funkčnej špecifikácie

Po úspešnom brainstormingu je ďalším krokom vytvorenie funkčnej špecifikácie, ktorá podrobne popisuje, čo aplikácia musí robiť a ako bude vyzerať jej funkcionalita. Funkčná špecifikácia sa zvyčajne delí do niekoľkých častí:

1. Základná funkcionalita – určenie, čo všetko aplikácia musí umožniť. To môže zahŕňať registráciu užívateľov, správu účtov, zobrazenie produktov alebo spracovanie platieb.
2. Rozdelenie na stránky a sekcie – určenie, ako bude aplikácia organizovaná, aké stránky alebo sekcie budú potrebné a aké funkcie budú priradené k jednotlivým stránkam.
3. Prioritizácia funkcií – stanovenie, ktoré funkcie sú nevyhnutné a ktoré môžu byť implementované neskôr. Tento krok pomáha sústrediť sa na kľúčové vlastnosti aplikácie a minimalizovať riziko zdržania projektu.

Funkčná špecifikácia je nevyhnutná na zabezpečenie, že všetky požiadavky sú jasne definované ešte pred začiatkom vývoja a umožňuje tímu lepšie pochopiť, čo je potrebné na dosiahnutie úspechu.

### Tvorba wireframes

Ďalšou dôležitou fázou je vytvorenie wireframes – vizuálnych náhľadov alebo skíc, ktoré zobrazujú rozloženie jednotlivých stránok aplikácie. Wireframes slúžia ako základný nástroj pre dizajnérov a vývojárov, aby pochopili, ako bude aplikácia vyzerať a ako budú usporiadané jej jednotlivé prvky.

Wireframes zvyčajne nezahŕňajú detailný dizajn, ale ukazujú základnú štruktúru stránky – kde budú umiestnené tlačidlá, navigačné menu, obrázky a texty. Vytvorením wireframes sa zabezpečí, že všetky dôležité prvky budú na správnych miestach a že používateľská skúsenosť bude intuitívna.

### Plánovanie roboty

Posledným krokom pred samotným začiatkom vývoja je plánovanie roboty. Tento krok zahŕňa rozdelenie celého vývojového procesu na menšie úlohy a stanovenie termínov, kedy by mali byť jednotlivé časti aplikácie hotové. Je dôležité, aby vývojový tím mal jasný časový harmonogram a vedel, ktoré úlohy sú najurgentnejšie.

Plánovanie roboty tiež zahŕňa identifikáciu potrebných technológií, nástrojov a rôznych frameworkov, ktoré budú použité pri vývoji. Každý člen tímu by mal mať presne definovanú úlohu a vedieť, na čo sa má počas vývoja sústrediť.

## Analýza backendových technológií vhodných na tvorbu webových aplikácií

Pri vývoji webových aplikácií je výber správnej backendovej technológie kľúčový pre zabezpečenie rozšíriteľnosti a udržateľnosti celého systému. Ako som už spomínal, backend zodpovedá za spracovanie logiky aplikácie, komunikáciu s databázami a poskytovanie dát pre frontend. V tomto texte sa zameriame na niektoré populárne backendové frameworky, ktoré sú vhodné pre tvorbu webových aplikácií, pričom pri ich výbere som sa zameral aj možnosti CMS nadstavieb, ktoré môžu výrazne uľahčiť správu obsahu.

### ASP .NET (C#)

ASP .NET je silný a flexibilný framework od Microsoftu, ktorý sa často používa na vývoj webových aplikácií s využitím jazyka C#. Tento framework je veľmi populárny v prostredí podnikových aplikácií a umožňuje vývoj aplikácií s vysokým výkonom a bezpečnosťou. Pre vývojárov, ktorí preferujú C# a .NET ekosystém, je ASP .NET skvelou voľbou.

Pre ASP .NET existujú rôzne CMS nadstavby, ktoré umožňujú jednoduchú správu obsahu bez potreby rozsiahleho vlastného vývoja. Umbraco CMS je jedným z najpopulárnejších CMS riešení pre tento framework, ktorý ponúka užívateľsky prívetivé rozhranie a flexibilitu pri správe obsahu. Ďalšími možnosťami sú Butter CMS a Strapi CMS, ktoré poskytujú headless (bez vlastného alebo frontendu, ktorý by zobrazoval obsah) CMS riešenia s API na správu obsahu.

### Spring Boot (Java)

Spring Boot je moderný framework pre jazyk Java, ktorý zjednodušuje vývoj komplexnejších a škálovateľných webových aplikácií. Spring Boot je obľúbený medzi vývojármi Java, pretože ponúka množstvo nástrojov na rýchle vytváranie aplikácií a ich následné nasadenie. Tento framework je veľmi užitočný, pokiaľ ide o prácu s mikroservisami a enterprise aplikáciami.

Pre tento framework existujú viaceré CMS riešenia, ktoré umožňujú jednoduchú integráciu do aplikácie. Crafter CMS je flexibilné a výkonné CMS, ktoré je navrhnuté špeciálne pre Java ekosystém. Rovnako ako pri ASP .NET, aj tu sú k dispozícii ďalšie headless CMS riešenia, ako napríklad Butter CMS a Strapi CMS, ktoré poskytujú jednoduchú správu obsahu cez API.

### Django (Python)

Django je najrozšírenejší webový framework pre Python, ktorý je známy svojím plne vybaveným ekosystémom. Django poskytuje množstvo vstavaných funkcií, ako je autentifikácia, správa databáz alebo admin rozhranie, čo výrazne urýchľuje vývoj webových aplikácií. Tento framework je ideálny pre projekty, kde je potrebná vysoká úroveň organizácie a bezpečnosti.

V prípade Django existuje viacero kvalitných CMS riešení, medzi ktoré patrí Django CMS a Wagtail, ktoré ponúkajú pokročilú správu obsahu s rôznymi rozšíriteľnými možnosťami. Okrem toho sú tu aj možnosti ako Butter CMS a Strapi CMS, ktoré sú platformovo nezávislé a umožňujú flexibilné API riešenia na správu obsahu.

### Ruby on Rails (Ruby)

Ruby on Rails je veľmi obľúbený framework pre jazyk Ruby, ktorý sa vyznačuje vysokou produktivitou a konvenciami (množstvo vstavaných funkcií), ktoré uľahčujú vývoj aplikácií. Rails je ideálny pre rýchly vývoj MVP aplikácií a startupy, ktoré potrebujú rýchlo uviesť aplikáciu do prevádzky.

Pre Ruby on Rails existujú CMS riešenia ako Alchemy CMS a Refinery CMS, ktoré sa výborne integrujú s týmto frameworkom. Tieto CMS ponúkajú flexibilitu pri správe obsahu a jednoduché rozšírenie funkcionality. Rovnako ako pri iných frameworkoch, aj pre Rails sú dostupné headless CMS ako Butter CMS a Strapi CMS, ktoré sa ľahko implementujú do aplikácie cez API.

### Express (Node.js)

Express je minimalistický framework pre Node.js, ktorý sa často používa pri tvorbe moderných webových aplikácií a mikroservisov. Express poskytuje flexibilitu a rýchlosť pri vývoji, čo ho robí ideálnym pre startupy a projekty, ktoré vyžadujú rýchle iterácie a zmeny.

Pre Express existuje niekoľko CMS riešení, ako napríklad Payload CMS a Keystone, ktoré sú plne prispôsobiteľné a ľahko integrované s aplikáciami postavenými na Express. Pre headless CMS riešenia sa môžu použiť aj Butter CMS a Strapi CMS, ktoré umožňujú správu obsahu cez API.

### Laravel (PHP)

Laravel je veľmi populárny PHP framework, ktorý je známy svojou elegantnosťou a jednoduchosťou. Laravel poskytuje množstvo nástrojov na správu databáz, autentifikáciu a vytváranie RESTful API, čo ho robí ideálnym pre tvorbu komplexných webových aplikácií.

Pre Laravel existujú kvalitné CMS riešenia ako October CMS a Winter CMS, ktoré umožňujú jednoduchú správu obsahu a sú veľmi flexibilné, čo je ideálne pre rôzne typy webových aplikácií. Tieto CMS sú špecifické pre Laravel, ale pre weby, ktoré potrebujú API-first prístup, môžu byť použité aj Butter CMS a Strapi CMS.

## Typy databáz a ich využitie pri vývoji webových aplikácií

Pri vývoji webových aplikácií je výber vhodnej databázy kľúčovým rozhodnutím, ktoré môže ovplyvniť výkon, škálovateľnosť a flexibilitu aplikácie. Databázy sa delia do rôznych typov na základe spôsobu ukladania a spracovania dát. Dve hlavné kategórie databáz sú relačné databázy a nerelačné databázy. Každý typ má svoje výhody a nevýhody, a preto je dôležité vybrať správny typ podľa konkrétnych požiadaviek aplikácie.

### Relačné databázy (SQL)

Relačné databázy sú najbežnejším typom databáz, ktoré udržiavajú dáta v štruktúre tabuliek, pričom medzi jednotlivými tabuľkami sú definované vzťahy (relácie). Dajú sa veľmi efektívne používať v aplikáciách, ktoré vyžadujú silnú integritu dát a komplexné dotazy.

Príklady relačných databáz:

1. MySQL – najpopulárnejšia a rýchla relačná databáza, vhodná pre širokú škálu aplikácií.
2. PostgreSQL – pokročilá open-source databáza s komplexnými funkciami, ideálna pre aplikácie s veľkými dátami alebo zložitými transakciami.
3. Microsoft SQL Server – relačná databáza od Microsoftu, obľúbená v podnikovom prostredí.
4. Oracle Database – silná databáza, ktorá sa využíva v náročných podnikových aplikáciách.

Relačné databázy sú ideálne pre aplikácie, ktoré potrebujú transakcie, zložité dotazy, spoľahlivosť a štruktúrované dáta (napríklad e-shopy, bankové systémy alebo aplikácie spravujúce inventár).

### Nerelačné databázy (NoSQL)

Nerelačné databázy, alebo NoSQL databázy, sú flexibilnejšie a umožňujú ukladanie dát v rôznych formátoch, ako sú dokumenty, kľúč-hodnota páry, grafy alebo stĺpcové modely. Tieto databázy sú vhodné pre aplikácie, ktoré potrebujú rýchlo spracovávať veľké množstvo nestruktúrovaných dát alebo dát s dynamickými schémami.

Príklady nerelačných databáz:

1. MongoDB – populárna dokumentová databáza, ktorá ukladá dáta vo formáte JSON. Vhodná pre aplikácie, ktoré musia spracovávať veľké objemy dát s meniacimi sa štruktúrami (napríklad sociálne siete, analytické nástroje).
2. Cassandra – distribuovaná stĺpcová databáza, ideálna pre aplikácie, ktoré potrebujú vysokú dostupnosť a horizontálnu škálovateľnosť (napríklad sledovanie logov v reálnom čase).
3. Redis – databáza kľúč-hodnota, ktorá je extrémne rýchla a často sa používa na cacheovanie alebo správu relácií (napríklad v systémoch, ktoré potrebujú rýchle čítanie/zápis dát).
4. CouchDB – databáza dokumentov, ktorá umožňuje flexibilné ukladanie dát a vysokú dostupnosť.

Nerelačné databázy sú vhodné pre aplikácie, ktoré potrebujú flexibilitu pri ukladaní rôznych druhov dát (napríklad mobilné aplikácie, real-time aplikácie, IoT aplikácie).

## Backendové technológie použité pri tvorbe webovej aplikácie

Pri tvorbe backendovej časti mojej webovej aplikácie som sa rozhodol pre technológie, ktoré sú robustné, flexibilné a overené v praxi. Jedným z kľúčových rozhodnutí bolo vybrať si OctoberCMS, postavený na frameworku Laravel (PHP), ktorý mi poskytol základ pre tvorbu aplikácie. Mám s ním viac ako 3 ročné skúsenosti, čo bol jeden z rozhodujúcich faktorov pri jeho výbere. Okrem toho, aplikácia využíva MySQL ako databázu a rôzne ďalšie technológie na ukladanie dát, logovanie a správu súborov, vrátane AWS S3 na správu obrázkov a BugSnag na analytiku a logovanie chýb.

### OctoberCMS: Developer-friendly framework

OctoberCMS je open-source CMS postavený na veľmi populárnom PHP frameworku Laravel. Jeho hlavnou výhodou je modularita a prehľadná štruktúra kódu, vďaka ktorej je vývoj aplikácie jednoduchší a efektívnejší. Tento systém je založený na pluginovej architektúre, čo znamená, že je možné pridávať rôzne moduly a funkčnosti podľa potreby bez zásahu do základného kódu. To je ideálne pre aplikácie, ktoré sa môžu v priebehu času rozširovať alebo potrebujú pravidelnú úpravu, pretože umožňuje rýchle prispôsobenie a modifikáciu aplikácie.

Jednou z ďalších veľkých výhod OCMS je integrovaný CMS systém, ktorý umožňuje správu obsahu priamo z administratívneho rozhrania. Tento systém je veľmi užívateľsky prívetivý a umožňuje rýchle nastavenie stránok, spravovanie obsahu a ich publikovanie bez potreby zásahu do kódu, avšak pre potreby našej aplikácie využijeme iba CMS rozhranie (listy a formuláre) pre entity našej našej aplikácie, teda nie frontendovú časť ktorou OCMS tiež disponuje. Pre webové aplikácie, ktoré potrebujú ľahkú správu obsahu bez zásahu do kódu počas chodu aplikácie, ako v našom prípade, je OCMS výborným riešením.

Vďaka tomu, že je postavený na Laravel frameworku, ktorý je jedným z najpopulárnejších a PHP frameworkov, môžeme využívať množstvo existujúcich knižníc a nástrojov, ktoré Laravel ponúka, pomocou Composer Package Manager. Rýchly vývoj, podpora pre RESTful API, automatické generovanie migračných skriptov pre databázu, jednoduchá autentifikácia a ďalšie vlastnosti robia Laravel aj v kombinácii s OCMS skvelou voľbou pre vytváranie moderných webových aplikácií.

### MySQL: Relačná databáza

Pre správu dát v mojej webovej aplikácii som sa rozhodol pre MySQL, čo je jedna z najrozšírenejších a najspoľahlivejších relačných databáz. MySQL je známa svojou rýchlosťou, škálovateľnosťou a vysokou dostupnosťou, čo ju robí ideálnou voľbou pre aplikácie s vysokými požiadavkami na výkon a spracovanie veľkého objemu dát. S MySQL mám skúsenosti, ktoré mi umožňujú efektívne optimalizovať dotazy, navrhovať komplexné schémy databáz a implementovať transakcie s dôrazom na integritu dát.

Relačné databázy, ako je MySQL, sú ideálne pre aplikácie, ktoré vyžadujú komplexné dotazy, transakcie a vzťahy medzi dátami. Pre aplikáciu, ktorá pracuje s rôznymi dátovými entitami a požaduje vysokú spoľahlivosť a transakčnú integritu, je MySQL perfektnou voľbou. S jej podporou pre ACID vlastnosti (Atomicity, Consistency, Isolation, Durability) môžem byť istý, že dáta v aplikácii sú vždy v súlade a správne spravované.

### BugSnag: Analytika a logovanie chýb

Aby sme mali prehľad o výkone aplikácie a mohli efektívne riešiť chyby, využívame BugSnag. BugSnag je služba na logovanie chýb a analytiku, ktorá poskytuje real-time monitoring a podrobný prehľad o chybách, ktoré sa v aplikácii vyskytnú. Pomáha nám rýchlo identifikovať a opraviť problémy, ktoré by mohli ovplyvniť používateľov, a to ešte predtým, ako sa stanú vážnymi problémami.

BugSnag umožňuje sledovať chybové hlásenia v aplikácii, detailne zobrazuje kontext chyby, vrátane informácií o prostredí, používateľoch a ďalších faktoroch. Tieto informácie nám pomáhajú analyzovať príčiny problémov a rýchlo ich vyriešiť, čo zvyšuje kvalitu aplikácie a zlepšuje používateľský zážitok.

# Návrh riešenia

## Vytvorenie backendovej časti webovej aplikácie

Ako som už vyššie spomenul, na vývoj backendovej časti mojej aplikácie som použil OCMS postavený na frameworku Laravel v kombinácii s relačnou databázou MySQL. Programovanie v OCMS je pomerne jednoduché, keďže už obsahuje množstvo hotových funkcionalít vo forme modulov, ktoré som si mohol jednoducho integrovať do svojho projektu. Aplikácia Autobahn.sk je moderný online autobazár, ktorý sa špecializuje výhradne na nemecké autá. Jej unikátnosť spočíva v integrácii technológií umelej inteligencie, ktoré napomáhajú analýze a spracovaniu údajov. Tieto technológie som integroval priamo na úrovni backendu.

Jednou z hlavných funkcií mojej aplikácie je možnosť prehliadať širokú databázu inzerátov. Na tento účel som vytvoril API rozhranie pre zoznam inzerátov, ktoré poskytuje dáta vo formáte JSON pre frontend. Toto API umožňuje nielen načítanie zoznamu všetkých inzerátov, ale aj pokročilé vyhľadávanie a filtrovanie podľa rôznych parametrov, ako sú značka, model, rok výroby, cena, stav vozidla či počet najazdených kilometrov.

Okrem základných filtrov API podporuje aj radenie výsledkov podľa preferencií používateľa, napríklad podľa ceny či najnovších inzerátov. Vďaka optimalizácii dotazov do databázy a využitiu cache mechanizmov v rámci OCMS je načítanie dát rýchle a efektívne, aj keď počet inzerátov rastie.

Pre každý inzerát som vytvoril API rozhranie, ktoré poskytuje všetky podrobnosti potrebné pre jeho zobrazenie na frontendovej strane. Toto API pre detail inzerátu zahŕňa údaje o vozidle, ako sú technické špecifikácie, fotografie a históriu ceny. Navyše obsahuje funkcie, ktoré umožňujú používateľovi inzerát označiť ako obľúbený, čím si ho môže jednoducho uložiť na neskoršie prehliadanie.

Ďalšou kľúčovou funkciou detailu inzerátu je sledovanie vývoja ceny. Backend analyzuje zmeny v cenách inzerátov a tieto informácie sprístupňuje používateľom vo forme dát, ktoré sa dajú reprezentovať na frontende vo forme prehľadných grafov. Taktiež som implementoval možnosť hodnotiť predajcov, čo zvyšuje dôveryhodnosť platformy a umožňuje kupujúcim lepšie sa rozhodnúť.

Na strane predajcov som implementoval API pre vytváranie, úpravu a mazanie inzerátov. Toto rozhranie umožňuje registrovaným používateľom jednoducho spravovať svoje inzeráty. Prostredníctvom jednoduchých POST requestov môžu pridávať nové inzeráty vrátane fotografií, upravovať existujúce inzeráty a podľa potreby ich mazať.

Pri vývoji tohto rozhrania som dbal na validáciu vstupov, aby som zabezpečil konzistenciu dát v databáze. Na validáciu som využil nástroje poskytované OCMS, pričom som implementoval aj vlastné pravidlá validácie pre špecifické polia, ako sú VIN čísla alebo dátumy prvej registrácie.

Okrem inzerátov na predaj áut aplikácia ponúka aj funkcionalitu servisu vozidiel a vyhľadávania náhradných dielov. Na tento účel som vytvoril špecializované API rozhranie, ktoré umožňuje používateľom zadávať svoje požiadavky na servisné služby a vyhľadávať kompatibilné diely podľa značky a modelu vozidla.

Toto rozhranie využíva prepojenie s externými databázami náhradných dielov, ktoré sa pravidelne aktualizujú. Používateľ tak môže rýchlo nájsť potrebný diel a skontrolovať jeho dostupnosť u rôznych predajcov. V budúcnosti plánujem rozšíriť túto funkcionalitu o možnosť priameho objednávania dielov cez platformu.

Ďalšou dôležitou časťou backendu je API pre správu používateľov, ktoré zahŕňa všetky základné operácie, ako sú registrácia, prihlasovanie, odhlasovanie a obnova hesla. Na autentifikáciu používateľov som implementoval **JWT** autentifikáciu, ktorú som integroval cez vlastný plugin v OCMS.

Okrem základných operácií som vytvoril aj API na zobrazenie profilu používateľa, ktoré poskytuje iba inzeráty od daného používateľa. Toto API je zabezpečené middleware-om, ktorý overuje, či je používateľ prihlásený a oprávnený pristupovať k týmto údajom.

### Setup prostredia

Vývoj backendu aplikácie začal prípravou vývojového prostredia. Používam MacBook Air M1, na ktorom som využil aplikáciu **MAMP**, čo je skratka pre macOS (operačný systém), Apache (webový server), MySQL a PHP. Táto aplikácia poskytuje lokálny server s podporou PHP a MySQL, čo umožňuje pohodlný vývoj a testovanie aplikácie. Pre projekt môj projekt bola použitá verzia **PHP 8.2**, ktorá je plne kompatibilná s OCMS vo **verzii 3**.

Po nainštalovaní potrebného softvéru som vytvoril novú lokálnu MySQL databázu. Tento proces zahŕňal nasledovné kroky:

1. Spustenie MAMP - Aplikácia MAMP umožňuje správu MySQL databáz cez integrovaný nástroj **phpMyAdmin**. Po spustení MAMP som sa prihlásil do phpMyAdmin na adrese http://localhost:8888/phpmyadmin.
2. Vytvorenie novej databázy - V phpMyAdmin som vytvoril novú databázu pre vývoj aplikácie s názvom **autobahn\_local**. Nastavenie kódovania bolo zvolené ako utf8mb4\_general\_ci, aby bola zaistená podpora rôznych jazykov a špeciálnych znakov.
3. Konfigurácia pripojenia v **.env** súbore - Po vytvorení databázy som aktualizoval konfiguračný súbor .env, kde som definoval prístupové údaje, ako názov databázy, port, používateľské meno (root) a heslo (root), ktoré MAMP predvolene poskytuje.

Po úspešnom nastavení databázy som pokračoval vytvorením nového OCMS projektu pomocou nástroja Composer s príkazom:

**composer create-project october/october api**

Po spustení tohto príkazu sa vygenerovala štruktúra projektu, ktorá je základom pre vývoj aplikácie. Štruktúra obsahuje niekoľko kľúčových priečinkov:

1. plugins/ - Obsahuje všetky pluginy aplikácie. Pluginy predstavujú modulárne časti kódu, ktoré sa starajú o konkrétnu funkcionalitu.
2. storage/ - Slúži na ukladanie dočasných dát, ako sú cache, sessions, a logy.
3. vendor/ - Obsahuje všetky balíčky a knižnice nainštalované prostredníctvom Composeru.
4. config/ - Obsahuje konfiguračné súbory aplikácie.
5. bootstrap/ - Obsahuje inicializačné skripty, ktoré sa spúšťajú pri každom requeste.

Medzi dôležité súbory patrí:

1. composer.json - Definuje závislosti projektu.
2. artisan - CLI nástroj na správu aplikácie, podobný nástroju v Laraveli.
3. index.php - Vstupný bod aplikácie.

Po vytvorení štruktúry som spustil migrácie databázových tabuliek, ktoré boli súčasťou východiskového projektu, príkazom:

**php artisan october:up**

Tento krok zabezpečil, že základné tabuľky aplikácie (napr. tabuľka užívateľov) boli pripravené na používanie. Celý tento proces pripravil prostredie pre ďalší vývoj aplikácie, pričom všetky dáta boli ukladané do lokálnej databázy autobahn\_local, ktorá bola pripojená prostredníctvom MAMP.

### Setup úložiska

Pre ukladanie obrázkov a iných médií som využil cloudové úložisko AWS S3. Integrácia AWS S3 do OCMS si vyžaduje úpravu konfigurácií v súboroch **filesystems.php** a **cms.php** ktoré nájdeme v projekte v priečinku /config, kde som definoval disk typu na s3. Tento disk bol nastavený na použitie prístupových údajov (Access Key ID a Secret Access Key), ktoré som vytvoril v konzole AWS.

Na strane AWS bolo potrebné vytvoriť nový bucket, nastaviť správne oprávnenia a regionálne parametre (eu-central-1). Po dokončení konfigurácie sa aplikácia úspešne prepojila s úložiskom S3, čím sa zabezpečilo, že všetky médiá sú bezpečne uložené a dostupné v cloude.

### Vytvorenie pluginov

Pluginy sú základným stavebným kameňom aplikácie v OCMS. Umožňujú modulárny vývoj a jednoduchú správu kódu. Pre môj projekt som vytvoril viacero vlastných pluginov, ktoré som pre lepšiu prehľadnosť logicky zoskupil do spoločných „namespaces“. Napríklad **AppUser** ktorý obsahuje pluginy pre správu používateľov alebo **AppAd** ktorý zas obsahuje pluginy súvisiace s inzerátmi.

Nový plugin som vytvoril pomocou scaffolding (stavebného) commandu:

**php artisan create:plugin Namespace.Name**

Tento príkaz automaticky vygeneruje štruktúru pluginu v priečinku plugins/, ktorá obsahuje napríklad priečinky models/, controllers/ a súbor **Plugin.php** na registráciu pluginu. Okrem vlastných pluginov som použil aj oficiálne existujúce pluginy od autorov OCMS, ako **RainLab.User**, ktorý poskytuje základnú funkcionalitu pre správu používateľov vrátane modelu User, a **October.Drivers** plugin, ktorý obsahuje okrem iného aj ovládače pre AWS S3.

### Vytvorenie modelov

Modely v OCMS fungujú rovnako ako v Laraveli. Modely umožňujú komunikáciu s databázou a každý model predstavuje jednu tabulku a jej dáta v databáze. Modely som vytváral pomocou príkazu:

**php artisan create:model Namespace.Name ModelName**

Tento príkaz okrem vytvorenie triedy „ModelName“ vygeneruje taktiež v priečinku updates/ nový súbor pre tzv. migráciu, teda kód ktorý nejakým spôsobom upravuje databázu, pričom v tomto prípade sa jedná o migráciu na vytvorenie novej tabuľky pre daný model.

Ako príklad, model **Ad** predstavuje jednotlivé inzeráty. Tento model obsahuje definície atribútov (napr. title, description, price), relácie (napr. vzťah k modelu User) a metódy na manipuláciu s dátami. Ďalšie modely v mojom projekte zahŕňajú napríklad AdEnum, AdPrice, User a UserFlag.

### Vytvorenie HTTP ovládačov

Kontrolery, známe aj ako ovládače, zohrávajú kľúčovú úlohu v projekte Autobazár Autobahn.sk, keďže v nich definujeme logiku pre spracovanie požiadaviek od frontendu na backend. Ovládače spravujú všetky akcie, ktoré sú vykonávané pri volaní jednotlivých endpointov.

Tieto požiadavky môžu byť dvoch typov: **GET** a **POST**. **GET** požiadavky slúžia na získanie dát, napríklad načítanie detailov konkrétneho inzerátu alebo zoznamu dostupných áut. Na druhej strane, **POST** požiadavky umožňujú vykonávať zložitejšie akcie, ako je pridávanie nového inzerátu alebo registrácia nového používateľa, pričom tieto dáta zapisujú do databázy.

### Užívatelia a autentifikácia

Pre autentifikáciu som použil knižnicu **tymon/jwt-auth**, ktorú som implementoval prostredníctvom vlastného pluginu **AppUser.UserApi**. Tento plugin obsahuje všetku logiku pre autentifikáciu používateľov, vrátane HTTP controllerov na rôzne operácie ako prihlásenie alebo registrácia a middleware **Authenticate**, ktorý používam naprieč celou aplikáciou na sprístupnenie endpointov len pre authentifikovaných používateľov.

## Technická architektúra backendovej časti webovej aplikácie

Pri vývoji mojej aplikácie som kládol dôraz na dobre premyslenú technickú architektúru. Táto architektúra bola navrhnutá tak, aby umožňovala jednoduchú správu, rozširovanie a údržbu aplikácie počas celého vývoja, pričom zároveň spĺňala špecifické požiadavky projektu. V úvodných fázach som sa zameral na plánovanie architektúry a jej postupné dolaďovanie, čo sa ukázalo ako kľúčové najmä v situáciách, keď bolo potrebné realizovať nové požiadavky alebo vykonávať väčšie zmeny.

Architektúra backendu aplikácie nebola statická, ale bola priebežne upravovaná a optimalizovaná podľa konkrétnych potrieb projektu. Už pri návrhu som si uvedomil, že kvalitná technická architektúra zabezpečuje nielen lepšiu škálovateľnosť aplikácie, ale aj prehľadnosť kódu a ľahšiu spoluprácu pri budúcom rozširovaní aplikácie.

### Pluginová architektúra

Na začiatku projektu som si pripravil pluginovú architektúru v programe Google Sheets. Tento dokument slúžil ako nástroj na vizualizáciu jednotlivých častí aplikácie a na lepšie pochopenie komplexnosti celého riešenia. Zároveň som si vďaka nemu mohol vopred definovať, aké funkcie bude aplikácia potrebovať a ako ich logicky rozdeliť do samostatných modulov. V rámci plánovania pluginovej architektúry som definoval, aké pluginy budú v aplikácii prítomné, aké sú medzi nimi vzťahy a ako budú zoskupené do spoločných namespacov, aby sa zabezpečila ich lepšia organizácia.

Pod pluginovou architektúrou rozumiem koncept, v ktorom som funkcie aplikácie rozdelil do menších, izolovaných častí – pluginov. Každý plugin sa stará o konkrétnu funkcionalitu a komunikuje s ostatnými modulmi prostredníctvom jasne definovaných rozhraní (API).

Pri návrhu pluginovej architektúry pre moju aplikáciu som jednotlivé pluginy rozdelil do nasledujúcich namespacov:

1. AppDev: Tento namespace obsahuje plugin pre seedovanie databázy (AppDev.Seed). Tento plugin som použil na uloženie testovacích dát, ktoré sa naimportujú do databázy počas vývoja pomocou CLI príkazu.
2. AppAd: Namespace pre pluginy, ktoré súvisia priamo s inzerátmi. Zahŕňajú logiku pre zoznam inzerátov, detaily inzerátov a API na ich správu. Tieto pluginy pokrývajú všetky aspekty súvisiace s inzerátmi, kategóriami, cenami, vrátane filtrovania, triedenia a interakcie s používateľmi.
3. AppService: V tomto namespaci som umiestnil pluginy, ktoré súvisia so servisom vozidiel. Tieto pluginy sa starajú o správu údajov o servise, vyhľadávanie náhradných dielov a komunikáciu s externými službami pre aktualizáciu týchto informácií.
4. AppQna: Tento namespace obsahuje plugin pre často kladené otázky (FAQ). Plugin spravuje logiku pre ukladanie otázok a odpovedí, ich kategorizáciu a poskytovanie dát cez API rozhranie.
5. AppApi: Tento namespace obsahuje všeobecné pluginy súvisiace s API rozhraním. Napr. plugin **AppApi.ApiResponse** definuje štandard pri posielaní HTTP odpovedí, plugin **AppApi.ApiException** má zase na starosti riešenie a správne interpretovanie rôznych výnimiek ktoré môžu počas chodu aplikácie nastať. Plugin **AppApi.CORS** má na starosti CORS nastavenia ktoré sú pri webových stránkach dôležitá záležitosť z hladiska bezpečnosti.

Táto architektúra umožňuje rozdelenie zodpovedností jednotlivých častí aplikácie, čo prispieva k prehľadnosti a jednoduchšiemu vývoju. Nižšie je možné vidieť schému pluginovej architektúry vytvorenú v Google Sheets (obr. XY).

### Architektúra OCMS aplikácie

Keď používateľ odošle požiadavku (napríklad cez prehliadač alebo API), prebehne v aplikácii nasledujúci proces:

1. Prijatie požiadavky: HTTP požiadavka je zachytená vstupným bodom aplikácie, ktorým je súbor **index.php** v root adresári OCMS aplikácie. Tento súbor inicializuje jadro systému v priečinku bootstrap/ a zavádza potrebné súčasti, ako je konfigurácia a autoloading tried.
2. Routing: Požiadavka je nasmerovaná na správny route definovaný v routing súboroch aplikácie. V prípade mojej aplikácie sú všetky endpointy definované v jednotlivých pluginoch v súboroch **routes.php**.
3. Middleware: Po spracovaní routingu sa požiadavka prechádza cez middleware. Middleware môže vykonávať rôzne činnosti, ako je autentifikácia, logovanie alebo validácia údajov.
4. Kontroler: Po prechode middleware-om sa požiadavka dostane k príslušnému kontroleru, ktorý obsahuje business logiku potrebnú na spracovanie požiadavky. Tu sa vykonávajú rôzne operácie, ako je čítanie údajov z databázy, validácia alebo príprava odpovede.
5. Odpoveď: Nakoniec je výsledok spracovania požiadavky vrátený používateľovi vo forme HTTP odpovede. V našom prípade sa jedná o odpoveď vo formáte JSON.

V rámci OCMS je architektúra aplikácie rozšírená o podporu pluginov, ktoré umožňujú jednoducho rozširovať funkcionality bez nutnosti zasahovať do jadra systému. Každý plugin má vlastné routing súbory, controllery, modely a ďalšie komponenty, čo umožňuje vývojárom pracovať na jednotlivých častiach aplikácie nezávisle.

## Databáza webovej aplikácie

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

### Dátová štruktúra

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

## Zverejnenie webovej aplikácie na doménu a jej nasadenie na server

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

## Podnikateľský plán webovej aplikácie

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

## Marketingový plán a stratégia webovej aplikácie

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

# Záver

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem.

In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla. Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien.

Donec ut est in lectus consequat consequat. Etiam eget dui. Aliquam erat volutpat. Sed at lorem in nunc porta tristique. Proin nec augue. Quisque aliquam tempor magna. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nunc ac magna. Maecenas odio dolor, vulputate vel, auctor ac, accumsan id, felis. Pellentesque cursus sagittis felis.

# Zoznam použitej literatúry

[1] <https://www.geeksforgeeks.org/web-technology/>

[2] <https://www.geeksforgeeks.org/backend-development/>

[3] <https://scopicsoftware.com/blog/backend-technologies/>

[4] <https://octobercms.com/blog/post/putting-octobercms-words>

[5] <https://docs.octobercms.com/>

[6] <https://jwt-auth.readthedocs.io/en/develop/>

[7] <https://www.w3schools.com/MySQL/default.asp>

# Prílohy <volitelne>

## Príloha A – Zdrojový kód

**príloha B - Fotodokumentácia**