Stredná priemyselná škola elektrotechnická  
Hálova 16, 851 01 Bratislava

**AUTOBAHN.SK - BACKEND WEBOVEJ APLIKÁCIE**

KOMPLEXNÁ ODBORNÁ MATURITNÁ PRÁCA

Bratislava, 2025 IV.D Martin Mojžiš

Stredná priemyselná škola elektrotechnická  
Hálova 16, 851 01 Bratislava

**AUTOBAHN.SK - BACKEND WEBOVEJ APLIKÁCIE**

KOMPLEXNÁ ODBORNÁ MATURITNÁ PRÁCA

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

Bratislava, 2025 IV.D Martin Mojžiš

<SEM VOLZITE ZADANIE, PODPISANE>

**Čestné vyhlásenie**

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

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

V Bratislave, 20. 2. 2025 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](#_Toc190302482)

[1 Teoretické východiská 7](#_Toc190302483)

[1.1 Štruktúra webových aplikácií 7](#_Toc190302484)

[1.1.1 Frontend – Užívateľská vrstva 7](#_Toc190302485)

[1.1.2 Backend – Logická vrstva 8](#_Toc190302486)

[1.1.3 Databáza – Úložisko dát 8](#_Toc190302487)

[1.1.4 API – Prepojenie vrstiev 8](#_Toc190302488)

[1.2 Vývojový plán webovej aplikácie 10](#_Toc190302489)

[1.2.1 Brainstorming nápadov 10](#_Toc190302490)

[1.2.2 Tvorba funkčnej špecifikácie 10](#_Toc190302491)

[1.2.3 Tvorba wireframes 11](#_Toc190302492)

[1.2.4 Plánovanie práce 11](#_Toc190302493)

[1.3 Analýza backendových technológií vhodných na tvorbu webových aplikácií 11](#_Toc190302494)

[1.3.1 ASP .NET (C#) 11](#_Toc190302495)

[1.3.2 Spring Boot (Java) 12](#_Toc190302496)

[1.3.3 Django (Python) 12](#_Toc190302497)

[1.3.4 Ruby on Rails (Ruby) 12](#_Toc190302498)

[1.3.5 Express (Node.js) 13](#_Toc190302499)

[1.3.6 Laravel (PHP) 13](#_Toc190302500)

[1.4 Typy databáz a ich využitie pri vývoji webových aplikácií 13](#_Toc190302501)

[1.4.1 Relačné databázy (SQL) 14](#_Toc190302502)

[1.4.2 Nerelačné databázy (NoSQL) 14](#_Toc190302503)

[2 Návrh riešenia 15](#_Toc190302504)

[2.1 Backendové technológie použité pri tvorbe webovej aplikácie 15](#_Toc190302505)

[2.1.1 OctoberCMS: Developer-friendly framework 15](#_Toc190302506)

[2.1.2 MySQL: Relačná databáza 16](#_Toc190302507)

[2.2 Vytvorenie backendovej časti webovej aplikácie 16](#_Toc190302508)

[2.2.1 Nastavenie prostredia 18](#_Toc190302509)

[2.2.2 Nastavenie cloudového úložiska 19](#_Toc190302510)

[2.2.3 Vytvorenie pluginov 21](#_Toc190302511)

[2.2.4 Vytvorenie modelov 21](#_Toc190302512)

[2.2.5 Vytvorenie HTTP ovládačov 22](#_Toc190302513)

[2.2.6 Užívatelia a autentifikácia 22](#_Toc190302514)

[2.2.7 Testovanie API aplikácie 22](#_Toc190302515)

[2.3 Integrácie použité pri tvorbe webovej aplikácie 23](#_Toc190302516)

[2.3.1 BugSnag: Analytika a logovanie chýb 24](#_Toc190302517)

[2.3.2 Algolia: Vyhľadávanie pomocou umelej inteligencie 24](#_Toc190302518)

[2.3.3 OpenAI: Univerzálny model umelej inteligencie 26](#_Toc190302519)

[2.3.4 Google API: Google autentifikácia a Google vyhľadávanie lokality 27](#_Toc190302520)

[2.3.5 Mailjet API: Newsletter integrácia 28](#_Toc190302521)

[2.4 Technická architektúra backendovej časti webovej aplikácie 29](#_Toc190302522)

[2.4.1 Pluginová architektúra 30](#_Toc190302523)

[2.4.2 Architektúra OctoberCMS aplikácie 31](#_Toc190302524)

[2.5 Databáza webovej aplikácie 32](#_Toc190302525)

[2.5.1 Dátová štruktúra 32](#_Toc190302526)

[2.5.2 Cacheovanie databázy 33](#_Toc190302527)

[2.6 Zverejnenie webovej aplikácie na doménu a jej nasadenie na server 34](#_Toc190302528)

[2.6.1 DevOps 34](#_Toc190302529)

[2.6.2 Prípravy na nasadenie aplikácie 34](#_Toc190302530)

[2.6.3 Registrácia webovej domény 35](#_Toc190302531)

[2.6.4 Vytvorenie a nastavenie virtuálneho servera 35](#_Toc190302532)

[2.6.5 Prepojenie webovej domény s virtuálnym serverom 35](#_Toc190302533)

[2.6.6 Nasadenie a testovanie aplikácie na serveri 36](#_Toc190302534)

[2.7 Podnikateľský plán webovej aplikácie 36](#_Toc190302535)

[2.8 Marketingový plán a stratégia webovej aplikácie 36](#_Toc190302536)

[3 Záver 37](#_Toc190302537)

[Zoznam použitej literatúry 38](#_Toc190302538)

[Prílohy <volitelne> 7](#_Toc190302539)

[Príloha A – Zdrojový kód I](#_Toc190302540)

**Zoznam skratiek, značiek a symbolov**

**API** Application Programming Interface (Rozhranie pre programovanie aplikácií)

**AI** Artificial Intelligence (Umelá inteligencia)

**WWW** World Wide Web (Svetová sieť)

**IP** Internet Protocol (Internetový protokol)

**HTTP** Hypertext Transfer Protocol (Hypertextový prenosový protokol)

**HTTPS** Hypertext Transfer Protocol Secure (Protokol ktorý zabezpečuje komunikáciu medzi webovým serverom a prehliadačom)

**SSL** Secure Socket Layer (Protokol ktorý umožnuje šifrované pripojenie)

**PHP** Hypertext Preprocessor

**HTML** HyperText Markup Language (Značkovací jazyk pre hypertext)

**CSS** Cascading Style Sheets (Kaskádové štýly)

**CMS** Content Management System (Systém na správu obsahu)

**MVP** Minimum Viable Product (Minimálne životaschopný produkt)

**CLI** Command-line interface (Rozhranie príkazového riadka)

**IoT** Internet of Things (Internet vecí)

**JSON** JavaScript Object Notation (Formát zápisu objektov v JavaScripte)

**ACID** Atomicity, Consistency, Isolation, Durability (Atomicita, Konzistentnosť, Izolácia, Trvácnosť)

**CORS** Cross-Origin Resource Sharing (Zdieľanie zdrojov medzi doménami)

**AWS** Amazon Web Services

**VPS** Virtual Private Server (Virtuálny súkromný server)

**S3** Simple Storage Service

**REST** Representational State Transfer (Súbor architektonických obmedzení)

**DNS** Domain Name System (Systém názvov domén)

**GUI** Graphical User Interface (Grafické používateľské rozhranie)

**CDN** Content Delivery Network (Sieť pre doručovanie obsahu)

**ORM** Object–Relational Mapping (Objektovo-relačné mapovanie)

**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 a manažovať 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 zo štyroch hlavných vrstiev: frontend, backend, databáza a API.

### 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 (SQL)** – ako napríklad MySQL, PostgreSQL alebo Microsoft SQL Server, ktoré organizujú dáta do tabuliek a umožňujú ich prepojenie pomocou vzťahov.
2. **Nerelačné databázy (NoSQL)** – ako MongoDB alebo Cassandra, ktoré sú flexibilnejšie a používajú sa najmä pri práci s veľkým množstvom neštruktúrovaných 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 prostredníctvom **API (Application Programming Interface)**, ktoré definuje **pravidlá pre výmenu dát medzi jednotlivými časťami systému**. API slúži ako most medzi používateľským rozhraním a serverovou logikou, čím umožňuje efektívne získavanie a spracovanie informácií. Medzi najčastejšie používané prístupy patrí **REST API** a **GraphQL**, ktoré umožňujú rýchlu a spoľahlivú komunikáciu medzi komponentmi aplikácie.

**REST (Representational State Transfer)** je architektonický štýl pre distribuované hypermediálne systémy, ktorý bol prvýkrát formálne popísaný Royom Fieldingom v roku 2000. REST nie je konkrétny protokol ani štandard, ale súbor odporúčaní, ktoré zabezpečujú **jednoduchosť, škálovateľnosť a nezávislosť** medzi klientom a serverom. Webové API, ktoré sa riadi týmito zásadami, sa označuje ako **REST API** alebo **RESTful API**.

Hlavné princípy REST architektúry zahŕňajú:

1. **Jednotné rozhranie** – Každý zdroj musí byť jednoznačne identifikovateľný prostredníctvom URI (Uniform Resource Identifier). Interakcia so zdrojmi prebieha prostredníctvom reprezentácií v štandardizovaných formátoch, ako sú JSON alebo XML.
2. **Klient-server architektúra** – REST striktne oddeľuje klienta (používateľské rozhranie) od servera (dátová vrstva), čo umožňuje nezávislý vývoj a vylepšovanie jednotlivých komponentov.
3. **Bezstavovosť (statelessness)** – Každá požiadavka od klienta musí obsahovať všetky potrebné informácie na jej spracovanie. Server si neukladá žiadne informácie o predchádzajúcich požiadavkách klienta.
4. **Podpora cacheovania** – Odpovede zo servera môžu byť označené ako „cacheable“, čo umožňuje klientovi opätovne použiť uložené dáta bez potreby opakovaného kontaktovania servera.
5. **Vrstvená architektúra** – API môže byť rozdelené do viacerých vrstiev, kde každá vrstva vykonáva špecifickú funkciu (napr. load balancer, autentifikácia, aplikačná logika). Klient nikdy nevidí viac ako len vrstvu, s ktorou priamo komunikuje.

REST API využíva **štandardné HTTP metódy** na interakciu so zdrojmi ako napríklad:

1. **GET** – Získanie dát zo servera (napr. zoznam inzerátov).
2. **POST** – Vytvorenie nového záznamu (napr. pridanie nového inzerátu).
3. **PUT** – Aktualizácia existujúceho záznamu (napr. úprava detailov inzerátu).
4. **DELETE** – Odstránenie záznamu (napr. vymazanie inzerátu).

## 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, čím sa docieli lepší používateľský zážitok.

### Plánovanie práce

Posledným krokom pred samotným začiatkom vývoja je plánovanie práce. 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 výkonný 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 kódu 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 architektonickou jednoduchosťou. Laravel poskytuje množstvo nástrojov na správu databáz, autentifikáciu a vytváranie **REST API**, čo ho robí ideálnym pre tvorbu komplexných webových aplikácií.

Pre Laravel existujú kvalitné CMS riešenia ako **OctoberCMS** a **WinterCMS** (ktoré vzniklo odlúčením od OctoberCMS), 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.

## 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** – robustná 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).

# Návrh riešenia

## Backendové technológie použité pri tvorbe webovej aplikácie

Pri tvorbe backendovej časti mojej webovej aplikácie som sa rozhodol pre technológie, ktoré sú spoľahlivé, flexibilné a overené v praxi. Jedným z kľúčových rozhodnutí bolo vybrať si OctoberCMS, postavený na frameworku Laravel, 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 integrácie technológií ktoré sa nižšie spomenieme.

### 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 OctoberCMS 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 mojej aplikácie využijeme iba CMS rozhranie (listy a formuláre) pre jednotlivé entity v mojej aplikácii, teda nie frontendovú časť ktorou OctoberCMS tiež disponuje. Pre webové aplikácie, ktoré potrebujú ľahkú správu obsahu bez zásahu do kódu počas chodu aplikácie, ako v našom prípade, je OctoberCMS 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 **REST API**, automatické generovanie migračných skriptov pre databázu, jednoduchá autentifikácia a ďalšie vlastnosti robia Laravel aj v kombinácii s OctoberCMS 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.

## Vytvorenie backendovej časti webovej aplikácie

Ako som už vyššie spomenul, na vývoj backendovej časti mojej aplikácie som použil OctoberCMS postavený na frameworku Laravel v kombinácii s relačnou databázou MySQL. Programovanie v OctoberCMS 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 (AI), ktoré napomáhajú vyhľadávaniu, personalizovaniu a spracovaniu údajov. Tieto technológie som integroval priamo na úrovni backendu pomocou technológií spomenutých v **2.1** - **Algolia AI search** a **OpenAI**.

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. Toto vyhľadávanie je obohatené o integráciu na Algolia AI search, čo je vyhľadávaci engine ktorý je naučený na porozumenie ľudskej reči pri dopytovaní.

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 OctoberCMS 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** požiadaviek 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 OctoberCMS, 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 OctoberCMS.

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.

### Nastavenie prostredia

Vývoj backendu aplikácie začal prípravou vývojového prostredia. Používam MacBook Air M1 2020, 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. Pre projekt môj projekt bola použitá verzia **PHP 8.2**, ktorá je plne kompatibilná s OctoberCMS vo verzii **v3**.

Na vývoj aplikácie využívam integrované vývojové prostredie (IDE) **PhpStorm** od spoločnosti JetBrains, ktoré poskytuje širokú škálu nástrojov na efektívnu prácu s PHP. PhpStorm ponúka pokročilé funkcie, ako je automatické dopĺňanie kódu, integrovaná podpora pre databázy, debugging a inteligentná analýza kódu, čo výrazne zjednodušuje proces vývoja.

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 GUI**. 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, s ktorými sa v automobilovom priemysle môžeme stretnúť.
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 OctoberCMS 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 dôležitých 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 každej požiadavke.

Medzi dôležité súbory patrí:

1. **composer.json** - Definuje závislosti projektu (knižnice, ktoré aplikácia využíva) pre Composer.
2. **artisan** - **CLI** nástroj na správu aplikácie, podobný nástroju v Laraveli.
3. **index.php** - Vstupný bod aplikácie, na tento súbor odkazujú všetky požiadavky.
4. **.htaccess** - Textový súbor určený na úpravu niektorých vlastností Apache servera.

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 - **users**) 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 ako už bolo spomenuté vyššie.

### Nastavenie cloudového úložiska

Pri návrhu systému pre ukladanie obrázkov a iných médií som zvažoval dve možnosti – **klasické lokálne úložisko na serveri** a **cloudové úložisko**. Hlavným dôvodom, prečo som sa rozhodol pre cloudové riešenie, bolo škálovateľnosť a dostupnosť dát. Lokálne úložisko by síce mohlo byť jednoduchšie na implementáciu, avšak v prípade rastu aplikácie by mohlo narážať na **výkonnostné a kapacitné limity servera**.

Cloudové úložisko poskytuje **vyššiu spoľahlivosť**, pretože dáta sú distribuované v dátových centrách s **automatickou replikáciou** a **zálohovaním**. Ďalším faktorom bolo **zníženie záťaže na aplikačný server**, pretože statické súbory sa pri použití cloudu servírujú priamo z **CDN**, čím sa zlepšuje **rýchlosť načítania stránky**.

Pre ukladanie obrázkov a iných médií som využil **cloudové úložisko AWS S3 (Simple Storage Service)**. **Integrácia AWS S3 do** OctoberCMS si vyžaduje úpravu konfigurácie v súbore **filesystems.php**, ktorý sa nachádza v priečinku **config/ v root priečinku aplikácie**. V tomto súbore som definoval disk typu „**s3**“, pričom bolo nutné zabezpečiť autentifikáciu voči **S3 bucketu** pomocou AWS IAM.

Najskôr som v **AWS IAM (Identity and Access Management)** vytvoril nového používateľa s názvom **„autobahn“.** Tento používateľ dostal **iba oprávnenie „AmazonS3FullAccess"**, čím som zabezpečil, že má prístup len k S3 úložisku, no nemá nadbytočné oprávnenia v rámci celého AWS účtu. Po vytvorení používateľa som vygeneroval **Access Key ID** a **Secret Access Key**, ktoré sú nevyhnutné na autentifikáciu.

Ďalej bolo potrebné nakonfigurovať tieto údaje v súbore **.env**, ktorý obsahuje environmentálne premenné pre aplikáciu. Do tohto súboru som pridal nasledujúce konfigurácie:

* FILESYSTEM\_DRIVER=s3
* AWS\_ACCESS\_KEY\_ID=\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* AWS\_SECRET\_ACCESS\_KEY=\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* AWS\_DEFAULT\_REGION=eu-central-1
* AWS\_BUCKET=autobahn.sk
* AWS\_USE\_PATH\_STYLE\_ENDPOINT=false
* AWS\_URL=https://autobahn.sk.s3.amazonaws.com/
* AWS\_ENDPOINT=https://s3.eu-central-1.amazonaws.com

Na strane AWS bolo potrebné vytvoriť nový **bucket** s názvom **„autobahn.sk“**, nastaviť **správne oprávnenia** a **regionálne parametre**. Ako **región** som zvolil **Frankfurt (eu-central-1)**, ktorý je geograficky najbližší a poskytuje nízku latenciu.

OctoberCMS štandardne pracuje **s troma diskami súčasne**, pričom každý disk je určený na iný typ obsahu:

1. **uploads** – sem sa ukladajú nahrané médiá, napríklad obrázky v inzerátoch.
2. **resized** – obsahuje automaticky zmenšené verzie obrázkov pre lepší výkon a optimalizáciu načítania.
3. **media** – zahŕňa médiá nahrané z prostredia CMS, napríklad obrázky použité v stránkach alebo blogoch. Tento disk sa konkrétne v mojej aplikácii ale nepoužíva.

Každý z týchto diskov má v mojom **AWS S3 buckete** vytvorený samostatný **priečinok** („uploads“, „resized“ a „media“), do ktorého OctoberCMS automaticky ukladá súbory podľa ich účelu. Tento systém umožňuje **lepšiu organizáciu médií** a **efektívnejšiu správu obsahu**.

### Vytvorenie pluginov

Pluginy sú základným stavebným kameňom aplikácie v OctoberCMS. 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) príkazu:

**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 OctoberCMS, 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. Ďalej sa po spustení tohto príkazu vytvorí v danom plugine priečinok **updates/** a v ňom súbor **version.yaml**. V priečinku updates/ sa neskôr budú nachádzať jednotlivé migračné súbory a v súbore version.yaml sú tieto migrácie verzionované, aby sme vedeli .

### Vytvorenie modelov

Modely v OctoberCMS 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 mojom projekte, 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 priamo v aplikácii som použil knižnicu **tymon/jwt-auth**, ktorú som implementoval prostredníctvom vlastného pluginu **AppUser.UserApi**. Tento plugin obsahuje všetku logiku pre autentifikáciu používateľov, vrátane HTTP kontrolerov na rôzne operácie ako prihlásenie, registrácia alebo odhlásenie a middleware **Authenticate**, ktorý používam naprieč celou aplikáciou na sprístupnenie endpointov len pre autentifikovaných používateľov.

Okrem vlastného riešenia autentifikácie v aplikácii používame aj prihlásenie prostredníctvom Google účtu spomenuté v **2.1.6**, pričom používateľ nepotrebuje mať najskôr vytvorený účet prostredníctvom mojej registrácie, pretože sa mu účet automaticky vytvorí pri pokuse o autentifikáciu (v prípade že účet ešte nemá vytvorený).

### Testovanie API aplikácie

Testovanie API je nevyhnutnou súčasťou vývoja softvéru, pretože umožňuje overiť správnosť komunikácie medzi klientom a serverom. Pri vývoji mojej aplikácie bolo potrebné priebežne testovať jednotlivé API endpointy a overovať ich odozvu na rôzne typy požiadaviek. Na tento účel bola využitá aplikácia **Postman**, ktorá predstavuje jedno z najrozšírenejších riešení na testovanie **RESTful API**.

Postman je nástroj, ktorý umožňuje **odosielanie HTTP požiadaviek**, sledovanie odpovedí servera a automatizované testovanie API. Jeho hlavnou výhodou je jednoduché grafické rozhranie, ktoré uľahčuje prácu s rôznymi typmi požiadaviek ako **GET, POST, PUT, DELETE** a ďalšími. Okrem manuálneho testovania umožňuje **automatizované testovanie**, kde je možné definovať testovacie skripty pomocou jazyku **JavaScript**, ktoré overujú správnosť odpovedí. Ďalšou výhodou je možnosť **správy kolekcií požiadaviek**, čo umožňuje efektívnejšiu organizáciu testov podľa funkcionalít aplikácie.

Počas testovania API boli jednotlivé požiadavky rozdelené do **kolekcií** podľa ich funkcionality. Napríklad, pre správu inzerátov som vytvoril kolekciu **„Ad“**, ktorá obsahovala požiadavky súvisiace s operáciami nad inzerátmi, ako je ich vytváranie, úprava, mazanie a získavanie údajov. Kolekcia **„Auth“** bola venovaná autentifikácii a obsahovala požiadavky na **registráciu, prihlasovanie, odhlasovanie a získavanie tokenu** na autorizáciu ďalších API volaní.

Pri testovaní jednotlivých požiadaviek bolo potrebné simulovať rôzne scenáre, napríklad **odosielanie platných aj neplatných údajov**, testovanie **autorizácie** pomocou **JWT tokenov** a validáciu správnych odpovedí zo servera. Postman umožňuje ukladanie a opakované odosielanie požiadaviek, čo výrazne uľahčuje testovanie API počas celého vývojového procesu. Každá požiadavka mohla obsahovať **parametre v URL, hlavičky (Headers) alebo telo požiadavky (Body)**, pričom bolo možné definovať **variabilné hodnoty**, čo umožnilo testovať rôzne vstupy bez nutnosti manuálneho prepísania údajov.

Okrem bežného testovania poskytuje Postman aj **možnosť automatizovaných testov**, ktoré sa definujú v záložke **Tests**. Pomocou **JavaScriptu** je možné napríklad overiť, či odpoveď API obsahuje očakávaný HTTP status kód, správnu štruktúru JSON objektu alebo konkrétnu hodnotu v odpovedi.

## Integrácie použité pri tvorbe webovej aplikácie

Pri vývoji moderných webových aplikácií je nevyhnutné využívať externé služby a API, ktoré rozširujú funkcionalitu systému a umožňujú efektívnejšie spracovanie dát. Integrácie s rôznymi technológiami zabezpečujú vyšší výkon, spoľahlivosť a automatizáciu procesov, čo vedie k lepšiemu používateľskému zážitku. V mojej aplikácii som implementoval viacero technologických integrácií, ktoré pokrývajú rôzne aspekty fungovania systému.

V nasledujúcich odsekoch sú detailne popísané jednotlivé integrácie: **BugSnag** (analytika a logovanie chýb), **Algolia** (vyhľadávanie pomocou umelej inteligencie), **OpenAI** (univerzálny model umelej inteligencie), **Google API** (Google autentifikácia a Google vyhľadávanie lokalít) a **Mailjet API** (newsletter integrácia).

### 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 ponúka webové rozhranie (dashboard) dostupné na adrese [https://app.bugsnag.com](https://app.bugsnag.com/), kde je možné sledovať chybové hlásenia v aplikácii, detailne zobrazuje kontext chyby, vrátane informácií o prostredí, používateľoch a ďalších faktoroch, takže vidíme detailne aký „**use-case**“ nastal, kedy nastal a konkrétne akému používateľovi. Tieto informácie nám pomáhajú analyzovať príčiny problémov a rýchlo ich vyriešiť, čo zvyšuje kvalitu aplikácie a zlepšuje používateľský zážitok.

### Algolia: Vyhľadávanie pomocou umelej inteligencie

Vyhľadávanie je jednou z kľúčových funkcionalít v mojej aplikácii, pretože umožňuje používateľom rýchlo a presne nájsť vozidlá podľa ich preferencií. Pri výbere technológie na vyhľadávanie som sa rozhodol pre **Algolia**, ktorá poskytuje výkonné fulltextové vyhľadávanie poháňané umelou inteligenciou (AI). Algolia umožňuje **indexovanie** dát v reálnom čase, optimalizáciu výsledkov na základe relevantnosti a **personalizované vyhľadávanie** podľa potrieb používateľa.

Aby bolo možné vyhľadávať inzeráty v aplikácii, bolo potrebné zabezpečiť **pravidelnú synchronizáciu** údajov z databázy do Algolie. Každý inzerát v systéme obsahuje množstvo atribútov, ako je značka, model, rok výroby, cena, stav vozidla a ďalšie parametre, ktoré sú dôležité pri vyhľadávaní. Tieto údaje sa synchronizujú do Algolia indexu pri vytvorení, úprave alebo odstránení inzerátu.

Na synchronizáciu som využil **Algolia PHP klienta**, ktorý umožňuje jednoducho komunikovať s Algolia API. Pri každej zmene inzerátu sa automaticky vykoná aktualizácia záznamu v Algolia indexe. Proces synchronizácie prebieha v troch krokoch:

1. **Pridanie alebo aktualizácia inzerátu** – Pri každej zmene sa inzerát odošle do Algolia API cez metódu **saveObject()**. Algolia následne aktualizuje index v reálnom čase.
2. **Odstránenie inzerátu** – Ak používateľ vymaže inzerát, systém zavolá metódu **deleteObject()**, ktorá odstráni záznam z Algolia indexu.
3. **Hromadná aktualizácia** – Pri väčších zmenách, ako je import nových inzerátov, sa používa **batch()**, ktorá umožňuje synchronizovať viacero záznamov naraz.

Samotné vyhľadávanie v Algolia funguje na princípe indexov, čo sú **databázové štruktúry** optimalizované na rýchle vyhľadávanie. Pre aplikáciu som vytvoril hlavný index ads, ktorý obsahuje všetky relevantné informácie o inzerátoch. Tento index je nakonfigurovaný tak, aby vyhľadával nielen podľa hlavných parametrov, ale aj podľa **synonymických výrazov** a **pravopisných odchýlok**, čím sa zvyšuje presnosť výsledkov.

Vyhľadávanie v mojej aplikácii je dostupné priamo z domovskej obrazovky, kde sa nachádza hlavné vyhľadávacie pole. Toto pole je napojené na API endpoint, ktorý komunikuje s Algolia službou a vracia relevantné výsledky **v reálnom čase**.

Na backendovej strane som vytvoril API kontroler, ktorý prijíma vyhľadávací dopyt a odosiela ho do Algolia indexu. Po spracovaní požiadavky sa výsledky vrátia vo formáte JSON a frontend ich zobrazí používateľovi.

Vyhľadávanie podporuje **auto-doplňovanie**, vďaka čomu používatelia vidia návrhy výsledkov už pri zadávaní textu. Okrem základného fulltextového vyhľadávania Algolia umožňuje aj **filtrovanie výsledkov**, napríklad podľa značky, ceny alebo počtu najazdených kilometrov, čím sa ešte viac zlepšuje používateľský zážitok.

Webové rozhranie Algolia, dostupné na adrese <https://dashboard.algolia.com/>, poskytuje vizuálny prehľad a správu indexov, ktoré sa používajú pri vyhľadávaní. V rámci tohto rozhrania je možné **monitorovať** stav indexov, **analyzovať** vyhľadávacie dopyty a **upravovať** nastavenia relevancie výsledkov. Používateľ môže manuálne spravovať jednotlivé záznamy, overovať správnosť uložených údajov a vykonávať testovatacie vyhľadávania bez nutnosti úprav v kóde aplikácie.

V kontexte mojej aplikácie Algolia dashboard umožňuje kontrolovať správnu synchronizáciu inzerátov s indexom „**ads“**, nastavovať pravidlá pre vyhľadávanie a prispôsobovať spôsob radenia výsledkov. Okrem toho je možné analyzovať metriky, ako sú najčastejšie vyhľadávané výrazy, miera úspešnosti vyhľadávania a dopyty, ktoré nevrátili žiadne výsledky. Tieto dáta môžu slúžiť na optimalizáciu vyhľadávacieho procesu, napríklad pridávaním synonymických výrazov alebo úpravou priorít v algoritme relevancie.

### OpenAI: Univerzálny model umelej inteligencie

Integrácia umelej inteligencie do webových aplikácií sa v posledných rokoch stala čoraz populárnejšou, najmä v oblastiach spracovania prirodzeného jazyka. Rozhodol som sa do tejto aplikácie implementovať **OpenAI ChatGPT API**, konkrétne model **gpt-3.5-turbo**, ktorý umožňuje generovanie textu na základe zadaných vstupných údajov. Tento model som využil na **automatické generovanie popisov inzerátov**, čím som výrazne uľahčil a zrýchlil proces vytvárania inzerátov pre používateľov.

Napojenie na OpenAI API v rámci OctoberCMS prebieha prostredníctvom **REST HTTP požiadaviek**. OpenAI poskytuje **Chat Completion API endpoint** (*POST https://api.openai.com/v1/chat/completions*), ktorý umožňuje interakciu s jazykovým modelom na báze kontextovej konverzácie.

Každý request odosielaný na tento endpoint musí obsahovať autentifikačný token, ktorý je súčasťou HTTP hlavičky (*Authorization: Bearer {API\_KEY}*). V samotnom request body je potrebné definovať hlavné parametre určujúce správanie modelu:

1. **model** – názov modelu, v mojom prípade “**gpt-3.5-turbo**”.
2. **messages** – pole objektov reprezentujúcich konverzačný kontext. Obsahuje správy s rôznymi rolami:
   1. **system** – definuje inštrukcie pre model, ako má odpovedať.
   2. **user** – obsahuje konkrétny vstup od používateľa.
3. **temperature** – hodnota medzi 0 a 1, určuje mieru kreativity odpovedí (nižšie hodnoty generujú presnejšie odpovede, vyššie kreatívnejšie).
4. **max\_tokens** – maximálny počet tokenov (slovných jednotiek) v odpovedi.
5. **top\_p** – alternatíva k temperature, ovplyvňuje výber pravdepodobných odpovedí.

Ako som už naznačil, v rámci mojej aplikácie som konkrétne využil OpenAI API na **automatické generovanie popisov inzerátov**. Používateľ pri vytváraní nového inzerátu nemusí manuálne písať detailný popis vozidla – stačí, aby zadal základné technické údaje, ako sú značka, model, rok výroby, palivo, prevodovka a počet najazdených kilometrov.

Na základe týchto údajov backendová aplikácia odosiela request na OpenAI API, pričom systémový „prompt“ zabezpečuje, že vygenerovaný text bude **formálny, informatívny a marketingovo vhodný**. Napríklad pri zadaní vstupných údajov: „*BMW 320d, 2019, diesel, 120 000 km, automatická prevodovka“,* sa vygeneruje nasledovný popis inzerátu: „*Ponúkam na predaj BMW 320d z roku 2019 s úsporným dieselovým motorom a automatickou prevodovkou. Vozidlo má najazdených 120 000 km a je vo vynikajúcom technickom stave. Ponúka dynamickú jazdu a nízku spotrebu paliva. K dispozícii sú kompletné servisné záznamy. V prípade záujmu ma neváhajte kontaktovať.“*

Tento proces nielenže šetrí čas používateľov, ale zároveň zabezpečuje, že všetky inzeráty majú **konzistentnú kvalitu popisu**. Model ChatGPT je možné ďalej optimalizovať prispôsobením systémového „promptu“ tak, aby bol popis viac technický alebo viac orientovaný na predajné aspekty.

### Google API: Google autentifikácia a Google vyhľadávanie lokality

V rámci aplikácie som sa rozhodol integrovať **Google API** na dve hlavné funkcionality: autentifikáciu používateľov prostredníctvom Google účtu a vyhľadávanie lokalít pomocou **Google Maps API**. Pre napojenie na tieto služby som v rámci projektu vytvoril samostatný namespace **AppGoogle**, v ktorom sú implementované dva pluginy: **AppGoogle.GoogleUserAuth** a **AppGoogle.GoogleMapsFindPlace**.

Na implementáciu autentifikácie používateľov prostredníctvom Google účtu som využil knižnicu „**google/apiclient**“, ktorá poskytuje kompletné rozhranie na prácu s **Google OAuth 2.0** autentifikáciou. Táto knižnica umožňuje jednoduché spracovanie OAuth postupov, validáciu prístupových tokenov a získanie informácií o prihlásenom používateľovi.

Pre integráciu Google autentifikácie bolo potrebné dynamicky pridať nový autentifikačný provider do súboru **config/services.php** pod názvom „**google**“. Tento provider vyžaduje nastavenie troch hlavných premenných prostredia: **GOOGLE\_CLIENT\_ID**, **GOOGLE\_CLIENT\_SECRET** a **GOOGLE\_REDIRECT**, ktoré sa získavajú **z Google Cloud konzoly** a zároveň sú uložené v súbore **.env**.

V Google Cloud konzole (<https://console.cloud.google.com>) som vytvoril nový projekt s názvom „**Autobahn**“, v rámci ktorého som nastavil **OAuth consent screen**. Tento proces umožňuje definovať aplikáciu, ktorá bude žiadať o autentifikáciu, a povolenia, ktoré si bude nárokovať. Následne som v sekcii **Credentials** vygeneroval nové **OAuth Client ID**, ktoré predstavuje jedinečný identifikátor pre moju aplikáciu a slúži na autorizáciu požiadaviek na Google API.

V aplikácii autentifikácia prebieha nasledovne: pri pokuse o prihlásenie sa v **LoginApiController** vyšle udalosť **appuser.userapi.beforeLogin**, ktorú následne zachytáva plugin **AppGoogle.GoogleUserAuth**. Toto sa označuje ako „*hooknutie na event*“. Ak je v requeste parameter „**social**“ s hodnotou „**google**“, aplikácia očakáva aj parameter „**code**“, ktorý frontend získal od Google po úspešnej autentifikácii. Po validácii kódu je možné extrahovať informácie o používateľovi, ako napríklad meno, email a unikátne **Google ID**, a následne ich uložiť do databázy alebo spárovať s existujúcim účtom.

Druhou implementovanou funkcionalitou v rámci Google API je vyhľadávanie lokalít prostredníctvom **Google Maps API**. Táto funkcionalita je implementovaná v plugine **AppGoogle.GoogleMapsFindPlace**. Na rozdiel od autentifikácie, kde sa využíva trieda **Google\_Client**, komunikácia s Google Maps API prebieha priamo cez HTTP požiadavky pomocou triedy **GuzzleHttp\Client**. Táto knižnica umožňuje jednoduché odosielanie HTTP REST požiadaviek a spracovanie odpovedí v JSON formáte.

Na prístup k **Google Maps API** bolo potrebné vygenerovať **GOOGLE\_MAPS\_API\_KEY**, ktorý bol uložený do súboru **.env**. Tento kľúč sa používa na všetky požiadavky smerujúce do Google Maps API.

V aplikácii je vyhľadávanie lokalít implementované pomocou **Google Place API**, konkrétne pomocou endpointu **Find Place from Text**. Táto služba umožňuje získať najrelevantnejšiu lokalitu na základe textového vstupu. Pre tento účel som vytvoril HTTP endpoint „**api/v1/google-maps/find-place**”, ktorý môže frontend využiť napríklad pri dynamickom vyhľadávaní miesta inzerátu vo formulári na vytvorenie inzerátu.

Pre ochranu pred nadmerným počtom požiadaviek bol na tento endpoint nastavený „**throttle:60,1**“, čo znamená, že používateľ môže vykonať maximálne **60 požiadaviek za minútu.** Toto opatrenie chráni API pred zneužitím a nadmerným zaťažením.

Okrem vyhľadávania miest na základe textového vstupu aplikácia implementuje aj získavanie podrobností o konkrétnej lokalite na úrovni backendu. Táto funkcionalita umožňuje spracovať podrobnosti o vybranej lokalite, ako sú presná adresa, súradnice a ďalšie meta informácie, ktoré môžem využiť pri zobrazovaní podrobností o inzeráte.

### Mailjet API: Newsletter integrácia

Pre hromadné odosielanie e-mailov a správu kontaktov v rámci newsletterov bola do aplikácie integrovaná služba **Mailjet API**. Mailjet je cloudová e-mailová platforma, ktorá umožňuje jednoduché odosielanie transakčných aj marketingových e-mailov, správu kontaktov a personalizované kampane. Vzhľadom na potrebu efektívnej správy e-mailových adries používateľov bola využitá funkcionalita **Contact lists API**, ktorá umožňuje dynamické pridávanie a odoberanie kontaktov zo zoznamu newsletterov.

Na integráciu s Mailjet API bola použitá **PHP knižnica** „**mailjet/mailjet-apiv3-php**“, ktorá poskytuje jednoduché rozhranie na komunikáciu so službou Mailjet. Prvým krokom bolo **vytvorenie kontakt listu** v rozhraní Mailjet na adrese <https://app.mailjet.com/contacts> kde bol definovaný zoznam s názvom **„Autobahn.sk“**. Každý kontakt, ktorý sa má prihlásiť na odber newsletteru, musí byť následne pridaný do tohto zoznamu.

Pre správne fungovanie API bolo potrebné v aplikácii nakonfigurovať autentifikačné údaje, ktoré sa generujú v sekcii **API Key Management** na platforme Mailjet. Vygenerované kľúče sa následne uložili do konfiguračného súboru **.env**. Kľúče **MJ\_APIKEY\_PUBLIC** a **MJ\_APIKEY\_PRIVATE** slúžia na autentifikáciu pri volaniach API, zatiaľ čo hodnota **MJ\_CONTACT\_LIST\_ID** obsahuje jedinečný identifikátor zoznamu kontaktov, do ktorého sa budú pridávať nové e-mailové adresy.

Na komunikáciu so službou Mailjet API bola použitá trieda **\Mailjet\Client**, ktorá sa pri inicializácii vytvorí s použitím vygenerovaných API kľúčov. Pridanie nového kontaktu do zoznamu sa realizuje pomocou metódy **\Mailjet\Resources::$ContactslistManagecontact**, pričom sa v požiadavke odosiela e-mailová adresa používateľa spolu s typom operácie (pridanie alebo odstránenie kontaktu). Každý nový odberateľ je teda automaticky uložený do zoznamu "**Autobahn.sk**" a môže byť využitý v budúcich e-mailových kampaniach. Táto integrácia umožňuje efektívne spravovať používateľské e-mailové adresy a automatizovať proces registrácie na newsletter priamo z aplikácie.

## 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 tabuľkovom editore. 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 na 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 **AppQna.Qna** 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 (trieda **ApiResource** obsahuje viaceré metódy ako napr. **success()**, **error()** alebo **info()** ktoré reprezentujú rôzne typy odpovede), 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 OctoberCMS 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 OctoberCMS 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 správnu cestu definovanú v smerovacích (routing) súboroch aplikácie. V prípade mojej aplikácie sú všetky endpointy definované v jednotlivých pluginoch v súboroch **routes.php**.
3. **Middleware** – Po spracovaní smerovania sa požiadavka prechádza cez tzv. middleware. Middleware môže vykonávať rôzne činnosti, ako je autentifikácia, logovanie alebo validácia údajov. V mojej aplikácii sa môžeme stretnúť napríklad s middleware-om **IsSellerMiddleware** ktorý zabezpečuje aby sa na endpointy ktoré používajú daný middleware dostali len používateľa označení ako predajcovia.
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 OctoberCMS je architektúra aplikácie rozšírená o podporu pluginov, ktoré umožňujú jednoducho rozširovať funkcionality bez nutnosti zasahovať do jadra systému. Každý plugin má vlastné smerovacie (routing) súbory, kontrolery, modely a ďalšie komponenty, čo umožňuje vývojárom pracovať na jednotlivých častiach aplikácie nezávisle.

## Databáza webovej aplikácie

Každá webová aplikácia potrebuje spoľahlivý systém na ukladanie a správu dát. V mojom projekte som sa rozhodol použiť MySQL, relačnú databázovú platformu, ktorá je široko využívaná pre svoju stabilitu, výkon a širokú podporu nástrojov. MySQL podporuje transakcie, referenčnú integritu a optimalizované vyhľadávanie pomocou indexov, čo sú dôležité vlastnosti pre efektívne fungovanie aplikácie. V aplikácii je **databázový engine** MySQL využívaný v kombinácii s **ORM** systémom, ktorý poskytuje OctoberCMS. Tento framework umožňuje jednoduchú správu databázových entít a automatickú synchronizáciu modelov so štruktúrou databázy.

Okrem phpMyAdmin, ktorý je často používaný na správu databáz cez webové rozhranie, využívam aj desktopovú aplikáciu **TablePlus**. TablePlus je grafický nástroj na správu databáz, ktorý ponúka intuitívne rozhranie, možnosť vykonávania SQL dotazov a editáciu tabuliek v reálnom čase. Výhodou je tiež podpora viacerých databázových systémov, čo umožňuje jednoduché prepínanie medzi rôznymi databázami a ich konfiguráciami. V rámci vývoja je TablePlus veľmi užitočný pri ladení dát, sledovaní zmien a optimalizácii dotazov.

### Dátová štruktúra

Štruktúra databázy je navrhnutá tak, aby bola prehľadná a umožňovala efektívne vzťahy medzi jednotlivými entitami. V OctoberCMS sa dodržiava konvencia pomenovania tabuliek v **snake\_case** formáte, pričom názov tabuľky pozostáva z **namespace\_plugin\_plurálmodelu**. Napríklad, ak sa v plugine **AppAd.Ad** nachádza model **Ad**, tabuľka v databáze sa bude volať **appad\_ad\_ads**. Táto konvencia zabezpečuje konzistentnosť a jednoduchšiu orientáciu v databázovej štruktúre.

MySQL engine rozlišuje viaceré dátové typy, medzi ktoré patria:

1. **Číselné typy** – napr. INT, BIGINT, DECIMAL
2. **Textové typy** – napr. VARCHAR, TEXT
3. **Dátumové a časové typy** – napr. DATETIME, TIMESTAMP
4. **Logické hodnoty** – BOOLEAN (interné reprezentované ako TINYINT(1))

Pri definovaní stĺpcov sa v MySQL používajú atribúty ako:

1. **is\_nullable** – určuje, či môže byť hodnota v stĺpci NULL
2. **foreign\_key** – definuje vzťah medzi tabuľkami
3. **column\_default** – nastavuje predvolenú hodnotu pre daný stĺpec
4. **comment** – umožňuje pridať popis k jednotlivým stĺpcom pre lepšiu čitateľnosť

V relačných databázach je dôležité správne definovanie vzťahov medzi tabuľkami. V mojej aplikácii existuje napríklad vzťah medzi tabuľkou **users** a tabuľkou **appad\_ad\_ads**, kde každý inzerát patrí konkrétnemu používateľovi. Tento vzťah je realizovaný cez cudzí kľúč - **user\_id**, ktorý zabezpečuje, že každý inzerát patrí existujúcemu používateľovi v tabuľke users. Ak sa používateľ vymaže, všetky jeho inzeráty budú automaticky odstránené vďaka pravidlu **ON DELETE CASCADE**. Táto štruktúra zaisťuje referenčnú integritu a minimalizuje nekonzistenciu v databáze.

### Cacheovanie databázy

Cacheovanie databázy je technika, ktorá umožňuje uchovávať často používané dáta v dočasnom úložisku, čím sa minimalizuje počet priamych dotazov do databázy. Tento proces výrazne zrýchľuje načítanie stránok a znižuje zaťaženie databázového servera. Cache funguje ako rýchla pamäťová vrstva medzi backendovým systémom a klientom, ktorá uchováva odpovede na opakujúce sa dotazy. Typické cache mechanizmy zahŕňajú:

1. **Cache-aside** – aplikácia najprv skontroluje cache, a ak dáta nie sú dostupné, načíta ich z databázy a uloží do cache.
2. **Read-through** – cache sama komunikuje s databázou a automaticky sa aktualizuje.
3. **Write-through** – pri zápise údajov sa najskôr aktualizuje cache a následne databáza.
4. **Write-back** – dáta sa najprv ukladajú do cache a až po určitom čase sa synchronizujú s databázou.

Použitie cacheovania databázy prináša viaceré výhody. Znižuje záťaž na databázový server, keďže menší počet dotazov znamená nižšiu spotrebu zdrojov. Zároveň zrýchľuje odozvu aplikácie, pretože načítanie dát z cache je podstatne rýchlejšie ako ich získavanie priamo z databázy. Cacheovanie tiež zlepšuje škálovateľnosť systému, umožňujúc zvládnuť väčší počet používateľov bez výrazného spomalenia. Ďalšou výhodou je nižšia latencia, ktorá minimalizuje oneskorenie pri získavaní často používaných údajov, čím sa zvyšuje celková efektivita aplikácie.

V rámci OctoberCMS sa cacheovanie využíva na zrýchlenie načítania stránok a optimalizáciu databázových operácií. OctoberCMS podporuje rôzne typy cacheovania, pričom medzi najčastejšie patrí:

1. **Súborová cache** – údaje sa ukladajú do súborov na disku, čo je jednoduché a efektívne riešenie pre menšie projekty.
2. **Pamäťová cache (napr. Redis, Memcached)** – využíva sa pri vyššom zaťažení systému na optimalizáciu výkonu.

V mojom projekte sa momentálne využíva klasická súborová cache, ktorá je postačujúca pre aktuálnu úroveň zaťaženia. Avšak, v prípade zvýšenia počtu používateľov a rastu náročnosti aplikácie plánujem prechod na **Redis**. Tento krok by umožnil rýchlejšie spracovanie dotazov, efektívnejšiu správu cache a lepšiu škálovateľnosť systému.

Redis ako distribuovaný cache poskytuje lepšie možnosti, ako sú podpora perzistentného ukladania dát, automatická expirácia a vysoká rýchlosť prístupu k údajom.

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

Proces zverejnenia webovej aplikácie na internete zahŕňa niekoľko krokov, od prípravy samotnej aplikácie cez výber vhodného **hostingu** a registráciu domény až po jej prepojenie so serverom a následné nasadenie. Dôležitou súčasťou celého procesu je aj konfigurácia bezpečnostných opatrení, ktoré zabezpečia, že aplikácia bude dostupná, spoľahlivá a chránená pred neoprávneným prístupom. Pre úspešné nasadenie webovej aplikácie je nevyhnutné venovať pozornosť nielen technickým aspektom, ale aj optimalizácii jej výkonu a správnemu nastaveniu serverového prostredia.

### DevOps

DevOps predstavuje súbor postupov, nástrojov a filozofií, ktoré spájajú vývojový (Development) a prevádzkový (Operations) tím s cieľom zabezpečiť efektívne nasadenie a správu aplikácií. DevOps umožňuje automatizáciu vývoja, testovania a nasadzovania softvéru, čo vedie k rýchlejšiemu a spoľahlivejšiemu dodaniu aplikácií. Medzi základné nástroje DevOps patria verzovacie systémy, ako Git, ktoré umožňujú spoluprácu viacerých vývojárov, CI/CD nástroje ako Jenkins alebo GitHub Actions, ktoré automatizujú nasadzovanie, a kontajnerizačné technológie ako Docker, ktoré uľahčujú nasadzovanie aplikácií na rôznych infraštruktúrach.

### Prípravy na nasadenie aplikácie

Pred samotným nasadením aplikácie do produkčného prostredia je potrebné vykonať niekoľko prípravných krokov. Prvým krokom je kontrola, či všetky komponenty aplikácie fungujú správne a či neobsahuje chyby, ktoré by mohli ovplyvniť jej stabilitu. Následne sa aplikácia optimalizuje, pričom sa odstraňujú nepotrebné súbory a knižnice, aby sa znížila jej veľkosť a zvýšila rýchlosť načítania. V rámci príprav je dôležité zabezpečiť aplikáciu pred potenciálnymi útokmi, ako sú SQL injekcie alebo neoprávnený prístup k citlivým údajom. Po vykonaní týchto krokov je aplikácia pripravená na nasadenie.

### Registrácia webovej domény

Registrácia webovej domény je kľúčovým krokom pri zverejnení webovej aplikácie. Doména slúži ako unikátna adresa, cez ktorú budú môcť používatelia pristupovať k aplikácii. Pre tento projekt som si zakúpil doménu **autobahn.sk** prostredníctvom služby **WebSupport**, ktorá poskytuje správu DNS záznamov a umožňuje jednoduché prepojenie domény so serverom. Proces registrácie domény zahŕňa jej zakúpenie a následnú každoročnú obnovu, aby sa predišlo jej exspirácii. SSL certifikát, ktorý umožní bezpečnú komunikáciu cez HTTPS, je v mojom prípade riešený až na úrovni servera.

### Vytvorenie a nastavenie virtuálneho servera

Pre správne fungovanie webovej aplikácie je nevyhnutné zabezpečiť vhodný hosting. V tomto prípade som sa rozhodol pre **AWS Lightsail**, čo je cloudová služba poskytovaná Amazon Web Services (AWS). AWS je momentálne najpoužívanejšia cloudová platforma s podielom 32 % na trhu. AWS Lightsail umožňuje jednoducho vytvárať a spravovať virtuálne súkromné servery (VPS) s pevne definovanými parametrami. Pre tento projekt som vytvoril VPS s názvom **AutobahnServer** v regióne **Frankfurt (eu-central-1)** s nasledujúcimi parametrami:

* 512 MB RAM
* 2 vCPU
* 20 GB SSD

Po vytvorení servera som mu priradil statickú IP adresu, aby sa nemenila pri reštarte inštancie. Táto IP adresa bude použitá pri konfigurácii DNS záznamov. Nastavil som Networking type na **IPv4-only**, čo je postačujúce pre moje potreby. Na zabezpečenie prístupu som nakonfiguroval **IPv4 Firewall** pravidlá a povolil nasledujúce porty:

* **SSH (22)** – vzdialený prístup na server
* **HTTP (80)** – nešifrovaný webový prenos
* **HTTPS (443)** – šifrovaný webový prenos
* **MySQL/Aurora (3306)** – databázové pripojenie

### Prepojenie webovej domény s virtuálnym serverom

DNS (Domain Name System) je systém, ktorý prekladá doménové názvy na IP adresy, čím umožňuje prístup k webovým stránkam prostredníctvom čitateľných adries namiesto číselných IP adries.

Vďaka systému DNS si používateľ nemusí pamätať IP adresu webovej stránky, stačí mu poznať jej názov domény. DNS sa postará o správne nasmerovanie automaticky. Keď používateľ zadá adresu stránky do prehliadača, ten zistí, ktorý DNS server obsahuje potrebné záznamy pre danú doménu, a vyžiada si jej IP adresu. DNS záznam domény obsahuje rôzne typy údajov, pričom každý z nich plní špecifickú úlohu. Medzi základné DNS záznamy patria:

* **A** – preklad domény na IPv4 adresu
* **AAAA** – preklad domény na IPv6 adresu
* **CNAME** – alias pre iný doménový názov

V konzole WebSupport som nastavil A záznamy:

* **www.autobahn.sk** – statická IP adresa servera pre podporu www domény
* **autobahn.sk** – statická IP adresa servera pre podporu www domény
* **\*.autobahn.sk** – statická IP adresa servera pre podporu subdomén

### Nasadenie a testovanie aplikácie na virtuálnom serveri

Po úspešnom prepojení domény so serverom som nakonfiguroval VPS ako webový server a nainštaloval **Bitnami LAMP stack**, ktorý obsahuje:

* **Linux** – operačný systém
* **Apache** – webový server
* **MySQL** – databázový server
* **PHP** – skriptovací jazyk

Verziu PHP som vybral tak, aby bola kompatibilná s OctoberCMS, ktorý používam pre aplikáciu. Na zabezpečenie **HTTPS** som použil službu **Let's Encrypt**, ktorá poskytuje bezplatné **SSL** certifikáty. Na generovanie certifikátov som použil ich CLI nástroj, ktorý som automatizoval pomocou shell scriptov a nastavil som na serveri **CRON** úlohu na automatické obnovovanie certifikátov.

Databázový server som spustil v **Docker** kontajneri, kde beží MySQL databáza. Aplikáciu som stiahol na server pomocou **Git**, umiestnil ju do priečinka „**htdocs**“ a nakonfiguroval **.env** súbor pre správne pripojenie k databáze a ďalšie parametre aplikácie.

Na záver som otestoval, či všetky funkcionality aplikácie pracujú správne, a overil som, že web je dostupný cez HTTPS s platným SSL certifikátom.

## 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] GEEKSFORGEEKS, 2024. *„Web Technology“.* [online]. [cit. 2024-12-26]. Dostupné na internete: <https://www.geeksforgeeks.org/web-technology/>.

[2] GEEKSFORGEEKS, 2025. *„Backend Development“.* [online]. [cit. 2024-12-26]. Dostupné na internete: <https://www.geeksforgeeks.org/backend-development/>.

[3] RAMSEY, B., 2024. *„Top 10 backend technologies and frameworks to use in 2024“.* [online]. [cit. 2024-12-27]. Dostupné na internete: <https://scopicsoftware.com/blog/backend-technologies/>.

[4] BOBKOV, A. a GEORGES, S., 2015. *„Putting October CMS into words“.* [online]. [cit. 2024-12-27]. Dostupné na internete: <https://octobercms.com/blog/post/putting-octobercms-words>.

[5] BOBKOV, A. a GEORGES, S., 2025. *OctoberCMS Documentation.* [online]. [cit. 2025-01-03]. Dostupné na internete: <https://docs.octobercms.com/>.

[6] OTWELL, T., 2025. *Laravel Documentation.* [online]. [cit. 2025-01-03]. Dostupné na internete: <https://laravel.com/docs/9.x>.

[7] LERDORF, R., 2025. *PHP Manual.* [online]. [cit. 2025-01-03]. Dostupné na internete: <https://www.php.net/manual/en/index.php>.

[8] ADERMANN, N. a BOGGIANO, J., 2025. *Composer docs.* [online]. [cit. 2025-01-04]. Dostupné na internete: <https://getcomposer.org/doc/>.

[9] TYMON, S., 2025. *JSON Web Token Authentication for Laravel & Lumen.* [online]. [cit. 2025-01-04]. Dostupné na internete: <https://jwt-auth.readthedocs.io/en/develop/>.

[10] W3SCHOOLS, 2025. *MySQL Tutorial.* [online]. [cit. 2025-01-10]. Dostupné na internete: <https://www.w3schools.com/MySQL/default.asp>.

[11] AMAZON WEB SERVICES, 2025. *Amazon Simple Storage Service Documentation.* [online]. [cit. 2025-01-19]. Dostupné na internete: <https://docs.aws.amazon.com/s3/>.

[12] MARTIN, R.C., 2017. *Clean architecture: A craftsman’s guide to software structure and Design.* Boston: Pearson Education. 432 s. ISBN 978-0-13-449416-6.

[13] GUPTA, L., 2023. *„What is REST?: REST API Tutorial“.* [online]. [cit. 2025-02-01]. Dostupné na internete: <https://restfulapi.net/>.

[14] DESSAIGNE, N., 2025. *Algolia Documentation.* [online]. [cit. 2025-02-01]. Dostupné na internete: <https://www.algolia.com/doc/>.

[15] ALTMAN, S., 2025. *OpenAI API Reference.* [online]. [cit. 2025-02-01]. Dostupné na internete: <https://platform.openai.com/docs>.

[16] ASTHANA, A., 2025. *Postman Docs.* [online]. [cit. 2025-02-01]. Dostupné na internete: <https://learning.postman.com/docs/introduction/overview/>.

[17] DURAND, W. a TARTARIN, J., 2025. *Contact Management.* [online]. [cit. 2025-02-10]. Dostupné na internete: <https://dev.mailjet.com/email/guides/contact-management/>.

[18] KURIAN, T., 2025. *Google Cloud Documentation.* [online]. [cit. 2025-02-10]. Dostupné na internete: <https://cloud.google.com/docs>.

[19] WIDENIUS, M., 2025. *MySQL 8.4 Reference Manual.* [online]. [cit. 2025-02-10]. Dostupné na internete: <https://dev.mysql.com/doc/refman/8.4/en/>.

[20] SETWEB.SK, 2024. „*Data Cache v Backend Systémoch“.* [online]. [cit. 2025-02-10]. Dostupné na internete: <https://setweb.sk/stranky/data-cache-v-backend-systemoch/>.

[21] GEEKSFORGEEKS, 2025. *„DevOps Tutorial“.* [online]. [cit. 2025-02-11]. Dostupné na internete: <https://www.geeksforgeeks.org/devops-tutorial/>.

[22] WEBSUPPORT.SK, 2024. *„Čo je to doména“.* [online]. [cit. 2025-02-11]. Dostupné na internete: <https://www.websupport.sk/podpora/kb/co-je-to-domena/>.

[22] WEBSUPPORT.SK, 2024. *„Čo je to DNS“.* [online]. [cit. 2025-02-11]. Dostupné na internete: < https://www.websupport.sk/podpora/kb/co-je-to-dns/>.

[23] AMAZON WEB SERVICES, 2025. *Amazon Lightsail Documentation.* [online]. [cit. 2025-02-11]. Dostupné na internete: <https://docs.aws.amazon.com/lightsail/>.

[24] BITNAMI, 2025. *Get started with LAMP.* [online]. [cit. 2025-02-11]. Dostupné na internete: <https://docs.bitnami.com/aws/infrastructure/lamp/get-started/get-started/>.

[25] LET’S ENCRYPT, 2025. *Documentation – Let’s Encrypt.* [online]. [cit. 2025-02-11]. Dostupné na internete: <https://letsencrypt.org/docs/>.

# Prílohy <volitelne>

## Príloha A – Zdrojový kód

**príloha B - Fotodokumentácia**