TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

# Inzertní server pro automobily

## Bakalářská práce

*Studijní program:* B2646 – Informační technologie
*Studijní obor:* 1802R007 – Informační technologie

*Autor práce:* **Aleš Foldyna**
*Vedoucí práce:* Mgr. Jiří Vraný, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

# Online car marketplace

## Bachelor thesis

*Study programme:* B2646 – Electrical engineering and informatics
*Study branch:* 1802R007 – Information Technology

*Author:* **Aleš Foldyna**
*Supervisor:* Mgr. Jiří Vraný, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

# Inzertní server pro automobily

*Jméno a příjmení:* **Aleš Foldyna**
*Osobní číslo:* M16000024
*Studijní program:* B2646 Informační technologie
*Studijní obor:* Informační technologie
*Zadávající katedra:* Ústav nových technologií a aplikované informatiky
*Akademický rok:* **2018/2019**

**Zásady pro vypracování:**

1. Seznamte se s problematikou programování webových služeb s podporou REST API a s problematikou ukládání většího množství provázaných dat.
2. Navrhněte aplikaci pro inzertní server, který umožní uživatelům zadávat, editovat a vyhledávat inzeráty z prostředí automobilů. Při návrhu se zaměřte zejména na bezpečnost systému a vyřešení konkurenčních operací obvyklých v tomto typu aplikací.
3. Navrženou aplikaci implementujte a ověřte její praktickou použitelnost. Při implementaci využijte architektury REST, vytvořte serverovou i klientskou část.

| | |
|---|---|
| *Rozsah grafických prací:* | dle potřeby |
| *Rozsah pracovní zprávy:* | 30 – 40 stran |
| *Forma zpracování práce:* | tištěná/elektronická |

**Seznam odborné literatury:**

[1] GRINBERG, Miguel. Flask web development. Sebastopol, CA: O'Reilly, 2014. ISBN 1449372627.
[2] RICHARDSON, Leonard a Michael AMUNDSEN. RESTful Web APIs. Beijing: O'Reilly, 2013. ISBN 978-1449358068.

| | |
|---|---|
| *Vedoucí práce:* | Mgr. Jiří Vraný, Ph.D.<br>Ústav nových technologií a aplikované informatiky |
| *Datum zadání práce:* | 18. října 2018 |
| *Předpokládaný termín odevzdání:* | 30. dubna 2019 |

L. S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci 18. října 2018

# Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

25. 4. 2019 Aleš Foldyna

## Abstrakt

Tato bakalářská práce se věnuje tématu vytvoření internetové aplikace pro inzerci automobilů. Předmětná internetová aplikace využívá moderní dostupné technologie. Bakalářská práce popisuje jednotlivé kroky při tvorbě aplikace, a to od samotného návrhu databáze, přes přemostění komunikace klienta s databází za pomoci REST API, až po naprogramování klientské aplikace.



**Klíčová slova:** Webová aplikace, REST API, VUEJS, JavaScript, PHP, Slim framework



## Abstract

This bachelor thesis deals with a topic of making up a web application on car advertising. The modern available technologies are used in the application. The thesis describes single steps when making up the application, since a draft of a database, via bridging a communication of clients with the database using the REST API, until the final programming of a client application.



**Keywords:** Web application, REST API, VUEJS, JavaScript, PHP, Slim framework

# Poděkování

Rád bych poděkoval vedoucímu mé práce panu Mgr. Jiřímu Vranému, Ph. D., za vstřícnost a objektivní připomínky během psaní této bakalářské práce.

Dále bych velice rád poděkoval paní doktorce Marcele Drahošové, která prováděla jazykovou korekturu této práce.

# Obsah

# Seznam zkratek

| | |
|---|---|
| **API** | Application Programming Interface |
| **CROS** | Cross-origin resource sharing |
| **CRUD** | CREATE, READ, UPDATE and DELETE |
| **DB** | Databáze |
| **DIČ** | daňové identifikační číslo právnické osoby |
| **DOM** | Document Object Model |
| **IČ** | identifikační číslo právnické osoby |
| **JS** | JavaScript |
| **JSON** | JavaScriptový objektový zápis |
| **JWT** | JSON Web Token |
| **MVC** | model-view-controler |
| **MVVM** | model-view-viewmodel |
| **PSČ** | poštovní směrovací číslo |
| **SŘBD** | Systém řízení báze dat |

# 1 Úvod

Dnešní doba se vyznačuje stoupajícím trendem internetových služeb, které v současnosti pokrývají velké množství uživatelských potřeb. Četné a užitečné využití má internet při obchodování a jeho význam v této oblasti stále významně roste. Jedněmi z průkopníků obchodu na internetu byly aplikace zaměřené na prodej automobilů, s nimiž se lze setkat i v rané době internetu. Proto pro účely této práce byla zvolena aplikace z této oblasti, neboť poměrně dlouhá doba existence inzertních serverů na prodej automobilů lépe umožňuje dlouhodobé analýzy takovýchto aplikací, a to jak z hlediska jejich vývoje a použitých technologií, tak i z hlediska jejich rozsáhlých funkcionalit.

Oblíbenost webových aplikací s sebou nese požadavek na neustálý vývoj technologií, jejichž prostřednictvím se aplikace vyvíjejí a tvoří. Tyto nové technologie umožňují v dnešní době programovat webové aplikace za pomoci jiných prostředků než před několika lety, přičemž tyto nové technologické prostředky jsou volně dostupné a jejich použití zejména zrychluje celkovou aplikaci, zlepšuje udržitelnost kódu a také výrazně zvyšuje programátorský komfort. Proto byl při této práci kladen velký důraz právě na použití soudobých moderních technologií využitelných pro tvorbu předmětné aplikace.

Cílem této práce je zmapovat a analyzovat nynější technologie pro tvorbu webových aplikací a vybrat z nich nejvhodnější k vytvoření inzertního serveru pro prodej automobilů, dále navrhnout a za využití optimálních technologií vytvořit a implementovat plně funkční aplikaci pro internetový prodej automobilů, a také při tvorbě aplikace prakticky ověřit u zvolených technologií jak jejich funkčnost, tak i to, jakou měrou ovlivňují proces vývoje a tvorby takové aplikace.

# 2 Rešerše

Tato kapitola se zabývá zmapováním prostředí týkajícího se tvorby webových aplikací. Zaměřilo se na tři klíčové oblasti vztahující k zadanému tématu: databázi pro uchování dat, rozhraní pro komunikaci databáze s klientem (API) a klientskou aplikaci. Ve všech zmíněných oblastech se zkoumalo, jaké prostředky jsou k dispozici, jaké jsou mezi nimi rozdíly a které z nich jsou vhodné pro vytvářenou aplikaci.

## 2.1 Již existující inzertní servery

Před samostatným zkoumáním jednotlivých částí bylo vyhledáno za účelem analýzy několik již existujících inzertních serverů. Při zkoumání nebylo možné zjistit architekturu aplikací z důvodu nepřístupnosti zdrojových kódů, proto bylo provedeno zkoumání pouze z uživatelského hlediska.

Porovnány byly tři inzertní servery (TipCars, Sauto, Auto.de). Zkoumány byly možnosti vyhledávacích kritérií a vedlejší funkcionality a bylo provedeno měření načítání výsledků vyhledávání pomocí webového prohlížeče Google Chrome, kde byly měřeny údaje: čas načítání, velikost stažená do prohlížeče, počet požadavků na server a rychlost překreslení DOM (výsledky jsou uvedeny v tabulce 2.1).

Každý z těchto serverů nabízí vyhledávání podle obdobných kritérií. Uživatel nejdříve zadá značku vozidla, poté upřesní model značky a případně zvolí rozmezí ceny, roky výroby, a najetých kilometrů vozidla. Níže je u každého serveru uvedeno shrnutí individuálních funkcionalit jednotlivých serverů.

**TipCars** je považován za největší tuzemský inzertní server. Kromě osobních automobilů nabízí i inzerci na užitkové, nákladní a obytné vozy, autobusy, motocykly, pracovní stroje, přívěsy a návěsy. Rozčleňuje inzeráty podle typu vozidla na nové a ojeté. Rozšířené vyhledávání umožňuje vyhledat i podle rozsáhlého seznamu výbavy. Jako oddělenou činnost nabízí články o vozidlech (nemají návaznost a nezobrazují se k jednotlivým inzerátům). Velmi dobrou vlastností tohoto serveru je i možnost jazykové volby mezi 5 jazyky, včetně přepočtu cen na měny daných zemí. Jeho největší slabostí je vzhled. Není vycentrovaný, působí nevyváženě a křiklavé reklamy nezapadají do celkového prostředí. Rovněž výsledky měření měl nejhorší ze všech zkoumaných serverů.

**Sauto** je druhým velkým tuzemským inzertním serverem. Patří pod český vyhledávací server Seznam. Podobně jako TipCars se nezaměřuje pouze na inzerci osobních automobilů, ale rozčleňuje typy inzerátů obdobně jako TipCars. Oproti němu má vyvážený vzhled a v provedeném měření měl výrazně lepší výsledky. Neumožňuje však vyhledávat pomocí výbavy.

**Auto.de** je německý inzertní server, který se jako jediný ze zkoumaných serverů specializuje pouze na osobní automobily. Má kvalitní intuitivní vzhled a dobře působí i grafické vybírání rozpětí vyhledávacích kritérií, kde ale chybí číselné údaje o aktuálních mezních hodnotách. Jeho vyhledávání umožňuje vyhledat i více značek a modelů v jednom dotazu. Inzeráty umožňuje rozdělit podle typu karoserie, přičemž toto rozdělení má podrobně rozčleněno a obdobně jako TipCars umožňuje vyhledávat podle různé výbavy automobilů. V měření dopadl srovnatelně jako se serverem Sauto.

Tabulka 2.1: Výsledky měření pomocí prohlížeč Chrome

| Server | Načítání (s) | Velikost (MB) | Počet požadavků | DOM (s) |
|---|---|---|---|---|
| **Tipcars** | 10.09 | 6.1 | 213 | 4.07 |
| **Sauto.cz** | 2.08 | 2.5 | 112 | 0.943 |
| **Auto.de** | 2.14 | 4.3 | 35 | 1.14 |

Předmětná vyvíjená aplikace by měla umožňovat srovnatelné vyhledávací kritéria, jako zkoumané servery. Stejně jako Auto.de by měla být zaměřena pouze na osobní automobily a jako vylepšení oproti zkoumaným serverům by měla výbava být vázána přímo k danému modelu, aby se při výběru staršího modelu nenabízela možnost výbavy, která v té době ještě nebyla dostupná. Stejně jako výbavu by měla aplikace hlídat roky výroby, kdy se model skutečně vyráběl, aby nebylo možné zvolit rok, kdy se vybraný model nevyráběl.

## 2.2 Databáze

Pro inzertní server, jako téměř pro každou webovou aplikaci, je neodmyslitelnou částí databáze. Z literatury [2, Provozní a analytické databáze: Teoretické základy] vyplývá, že databáze se skládá ze surových dat a skupiny programů, které surová data řídí (tzv. SŘBD). Vzhledem k tomuto faktu se práce zaměří na analýzu dostupných SŘBD.

Grinberg v [1] popisuje dva hlavní druhy databází. Relační (SQL) a nerelační (NoSQL) databáze. SQL využívají vztahy mezi jednotlivými tabulkami. Modifikace jedné tabulky ovlivní chování druhé, a proto je nezbytně nutné dobře

navrhnout databázi předem. Ovlivnění tabulek na základě vztahů zapříčiní špatnou manipulaci při případné modifikaci za běhu. NoSQL mají nestrukturovaný předpis pro ukládání jednotlivých dat a tím získávají dynamičnost. Klíčovým faktorem výběru relační databáze pro práci se stal systém ukládání dat. SQL ukládají data do tabulek a tím umožňují lepší práci s víceřádkovými operacemi.

Na základě výše zmíněných argumentů se v kapitole 2.2.1 hledal vhodný SQL systém řízení báze dat. Pozornost se zaměřila na otázky:

1. Jaké jsou nejpoužívanější SQL SŘBD?
2. Jaké jsou mezi nimi rozdíly?
3. Jaká kritéria jsou pro práci podstatná?

### 2.2.1 Možnosti SQL SŘBD

Conrad v [3] porovnává PostgreSQL, MySQL a komerční databáze. V článku se zmiňuje, že v posledních letech zlepšila komunita open-surce kvalitu těchto databázi na takovou úroveň, že mnoho firem, které se spoléhaly na komerční databáze z důvodu podpory a správy prostřednictvím velkého množství funkcí, přešlo na open-surce, a to především PostgreSQL a MySQL. Z toho vyplývá, že nejpoužívanější SŘBD jsou právě tyto dvě open-surce databáze.

V článku [6] se Bhatia snaží porovnat výše uvedené databáze. Z jeho porovnání vyplývá, že ačkoliv jsou si podobné, každá je vhodná na něco jiného a záleží na využití. Obě podporují shodné platformy, programovací jazyky, přístupové metody. PostgreSQL se více drží standardu SQL a převyšuje MySQL ve výkonosti. V otázce bezpečnosti PostgreSQL nabízí nativní podporu SSL pro šifrované připojení, naproti tomu MySQL obsahuje velké množství bezpečnostních funkcí.

Od inzertního serveru se očekává práce s větším množstvím dat. Klíčovým kritériem je výkonost. Práce s daty je důležitou funkcionalito vyvíjené aplikace. Z tohoto důvodu byl pro aplikaci vybrán SŘBD PostgreSQL.

## 2.3 REST API

Pro splnění zadání bylo nutné využít architekturu REST API jako přemostění mezi klientskou aplikací a databází.

V článku [4, Understanding REST] je vysvětleno REST jako architektonický styl pro tvorbu systémů. Nejedná se o standard, ale o soubor omezení například pro ujasnění vztahů mezi klientem a serverem. Hlavní myšlenka REST spočívá v následujících čtyrech principech.

**Zdroje** – snadno srozumitelné URI adresované struktury

**Reprezentace** – přenášení JSON, nebo XML pro reprezentování datových objektů a atributů

**Komunikace** – využití metod HTTP (např. GET, POST, PUT a DELETE)

**Bezstavovost** – pro zachování konzistence dat v databázi se neukládají žádné kontexty klienta mezi požadavky

Pro srozumitelnost a správné navržení REST je důležité využití HTTP metod. Jak je rovněž vysvětleno v [4, Understanding REST], metody se využívají následovně.

**GET** – požadavky musí být idempotentní (při opakování stejného požadavku musí být odpovědi vždy stejné)

**POST** – požadavek, aby zdroj udělal něco s poskytnutou entitou, nejčastěji se využívá pro vytvoření nové entity

**PUT** – požadavek na aktualizaci entity (na rozdíl od POST je PUT idempotentní)

**PATCH** – požadavek na aktualizaci pouze pole zadané entity (PATCH nemůže zajistit aktualizaci celého zdroje, proto není idempotentní)

**DELETE** – požadavek na odstranění zdroje

### 2.3.1 Možnosti aplikačních rámců

Možností aplikačních rámců pro naprogramování REST API je mnoho a nelze vyloženě říct, který je nejlepší nebo nejvhodnější. Kriterií pro porovnání je téměř nespočetně a hodně aplikačních rámců si je podobných. Porovnávat se dají na základě jazyka, ve kterém bude API naprogramovaná, možností vylepšujících funkcionalit jednotlivých rámců, oblíbenosti (silná komunita tlačí vývojáře k vylepšovaní), atd.

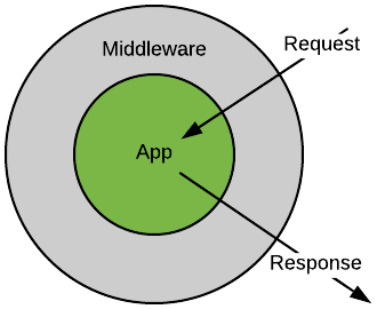
Na základě porovnání programovacích jazyků v [10, PHP vs Python vs Ruby] byl vybrán jazyk PHP, zejména z důvodů vysoké podpory pro práci s databázemi, která je klíčová pro API. Vybírání aplikačních rámců se tedy zúžilo na jazyk PHP.

V článku [11, 10 Popular PHP frameworks] lze najít v dnešní době nejpopulárnější PHP rámce a jejich popis. Na základě popsaných vlastností jednotlivých rámců byl zvolen Slim micro-framework. Je jednoduchý, s kvalitní dokumentací a často se využívá právě pro tvorbu RESTful APIs.

### 2.3.2 Aplikační rámec Slim

Slim je aplikační rámec pro PHP, který umožňuje napsat rychlou a efektivní API. Jak se lze dozvědět z [5, Slim a microframework for PHP], pomocí

něho je možné snadno napojit funkce prostřednictvím směrovače. Tato funkcionalita umožňuje získat konkrétní data na základě zadané URL adresy. Další funkcionalitou je Middleware. Jak funguje, je zobrazeno na obrázku 2.1. Požadavek přechází přes middleware, poté dojde ke zpracování odpovědi a následně odpověď znovu přechází přes middleware. Tím lze pracovat s požadavkem před tím, než dojde k požadované funkci, a následně přidat odpovědi potřebné funkcionality.




Obrázek 2.1: Schéma funkčnosti Middlewaru

Použití Slim je znázorněno v ukázce kódu 2.1. Řádkem 5 se vytvoří nový objekt Slim, přes nějž celá aplikace funguje. Následně je možné do instance Slim přidávat funkcionality. Řádkem 6 se definuje skupina s určitými stejnými vlastnostmi. V ukázce kódu přiřazuje všem směrovačům uvnitř skupiny před konkrétní adresu „v1“. To znamená, že všem adresám se před jejich konkrétní adresu předřadí adresa skupiny. Na sedmém řádku se definuje funkce konkrétního směrovače. V příkladu je použitá HTTP metoda GET. Slim umožňuje použít jakoukoliv z metod REST. Stačí pouze definovat příslušnou funkci danou metodou (např. $app->delete). Funkce má dva parametry. Prvním je již konkrétní adresa směrovače, druhý obsahuje funkci po zavolání prostřednictvím adresy. Příklad pouze vrací odpověď s úspěšným statusem 200 a informací ve formátu JSON o úspěšné operaci.

Na šestém řádku je znázorněno, jak se přiřazuje middleware (klíčovým slovem add). Middleware je možné přiřadit k celé aplikaci ($app->add), k jednotlivé funkci ($app->get()->add), nebo, jak je znázorněno na příkladu, ke skupině. V tom případě vlastnost middlewaru dědí všechny jednotlivé funkce uvnitř skupiny. Middleware má pouze jeden parametr s funkcí. Důležitou vlastností parametrické funkce jsou její parametry. Pracuje s požadavkem, odpovědí a cílovou funkcí. Na příkladu je vidět, jak se dotaz a odpověď zpracovávají. Nejdříve je možno definovat vlastnosti před zavoláním klíčové funkce (komentář „before“), poté se řádkem 12 vyvolá požadovaná akce na základě směrovače a následně se může pracovat s odchozí odpovědí (komentář „after“).

```php
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

$app = new \Slim\App;
$app->group('/v1',function() use ($app){
  $app->get('/open/znacka',function(Request $request, Response
  $response){
      return $response->withStatus(200)->withJson(array('
 success'=> array('text'=> "success response")));
  });
})->add(function($req,$res,$next){
  //before
  $response = $next($req,$res);
  //after
});
```

Ukázka kódu 2.1: Použití aplikačního rámce Slim

### 2.3.3 JSON Web Token

JWT (JSON Web Token), jak se lze dozvědět z [7], je ověřený standard, který definuje samostatný způsob bezpečného přenosu informací pomocí objektu JSON. Je digitálně podepsáný, takže může být ověřen a označen za důvěryhodný. Token je string, který se skládá ze tří částí (hlavička, požadavky a podpis), oddělených tečkami. Každá z těchto tří částí je samostatně zakódována pomocí BASE64-URL.

**Hlavička** – JSON zakódován pomocí Base64Url, který obsahuje dva atributy. Typ tokenu, kterým je JWT, a použitý algoritmus podepisování (HMAC SHA256, nebo RSA).

**Požadavky** – JSON zakódován pomocí Base64Url, který obsahuje požadavky pro ověření tokenu. JWT umožňuje tři typy požadavků (registrované, veřejné, soukromé). Registrované požadavky jsou stanoveny přímo standardem JWT, nejsou povinné, ale doporučují se (např. expirační doba), veřejné definují tvůrci knihoven a soukromé jsou určeny pro individuální využití v dané aplikaci.

**Podpis** – Používá se k ověření, že zpráva nebyla během cesty změněna. Podpis je tvořen zakódovanou hlavičkou a požadavky, které jsou oddělené tečkou. Tento string se zakóduje pomocí HMAC SHA256 tajným stringem uživatele. Zakódování může vypadat následovně: HMACSHA256(base64UrlEncode(hlavička) + "." + base64UrlEncode(požadavky), secret).

### 2.3.4 Knihovna pro dekódování JWT ve Slim

Pro zabezpečení Slim API je možné využít externí knihovnu „slim-jwt-auth“. Jak autor píše v [14], knihovna implementuje JWT autentizaci prostřednictvím Middlewaru.

Knihovna se definuje jako nový objekt v Middlewaru, který obsahuje seznam parametrů definovaných autorem knihovny. Jediný povinný parametr je „secret“ (popsáno v 2.3.3). Ostatní parametry jsou nepovinné a jejich zkrácený výčet s vysvětlením je uveden níže.

**Path** – Zabezpečené cesty směrovače, které se definují jako string, nebo pole stringů. Cesty nejsou definované na konkrétní směrovače, ale na množinu všech směrovačů, jež začínají zabezpečenou cestou.

**Ignore** – Výjimky ze zabezpečených cest. Definují se jako string, případně pole stringů.

**Header** – Umístění tokenu v hlavičce HTTP dotazu. Ve výchozím nastavení knihovna hledá token v autorizaci.

**Regexp** – Formát záhlaví tokenu, ve výchozím nastavení předpokládá „Bearer <token>“.

**Security** – Booleovská proměnná, která ovládá hlídání nezašifrovaného HTTP. Pokud požadavek přijde z nezabezpečeného protokolu, vyhodí výjimku „RuntimeException“. Nastavením security na hodnotu False se toto chování vypíná.

**Error** – Chybové hlášení v případě neúspěšného ověření tokenu. Definuje se jako funkce vracející odpověď o chybovém stavu.

## 2.4 Klientská část

Další zkoumanou částí byla klientská aplikace. To, co uživatel uvidí a v čem bude pracovat. Práce se zaměřila na naprogramování klientské platformy pro webové prohlížeče z důvodů lehké dostupnosti pro uživatele bez nutnosti instalace softwaru na počítač.

Webovou aplikaci lze naprogramovat od základů za použití elementárních funkcionalit programovacích jazyků pro tvorbu webů. V dnešní době však existuje celá řada aplikačních rámců, které práci usnadňují. Proto se v práci zkoumaly možnosti dostupných aplikačních rámců, kterými by se dala aplikace řešit.

### 2.4.1 Možnosti aplikačních rámců

Podobně jako u aplikačních rámců pro API existuje i pro tvorbu klientských aplikací obrovské množství možností. Bhagat v [8] se zaměřuje na tři nejpoužívanější front-end rámce (Angular JS, ReactJS, VueJS), které popisuje a zmiňuje jejich výhody a nevýhody.

**Angular JS** je open-source aplikační rámec, který je speciálně navržen pro vytváření webů s jednou stránkou dle vzoru MVC. Kombinuje HTML a TypeScript, který vykresluje vzhled stránky. AngularJS je nový aplikační rámec, nenavazuje na předchozí verze Angular a není s nimi kompatibilní.

**ReactJS** je zobrazovací knihovna, která má velkou výhodu v rychlosti. DOM je při aktualizaci nebo zobrazování velkého množství dat pomalý. ReactJS se snaží předcházet aktualizaci DOM a tím zrychluje vykreslování.

**VueJS** je aplikační rámec, který poskytuje ViewModel v architektuře MVVM. Jeho užitečnou vlastností je oddělená struktura HTML, CSS a JS, kterou získává přehlednost v kódu. Má obsáhlou a dobře popsanou dokumentaci.

Schae a jeho spolupracovníci v [9] vytvořili téměř totožné aplikace na různých rámcích a zkoumali jejich reálné vlastnosti. Zaměřili se na výkon (překreslování komponent), velikost (KB stažené na klienta) a počet řádků kódu. Pro porovnání byla v práci vybrána data jen výše zmíněných rámců. V tabulce 2.2 jsou znázorněna naměřená data z jejich testů.

AngularJS zaostává jak v rychlosti, tak i v komprimovanosti dat, počet řádků kódu měl nejméně z rámců zkoumaných v práci. ReactJS a VueJS mají ve všech třech oblastech podobné výsledky. ReactJS má o něco málo lepší hodnoty než Vue, ale rozdíly jsou téměř zanedbatelné.

Tabulka 2.2: Porovnání front-end aplikačních rámců

| | AngularJS | ReactJS | VueJS |
|---|---|---|---|
| Výkon (ms) | 4190 | 2623 | 2820 |
| Velikost (KB) | 317 | 98 | 100 |
| Řádky kódu | 1532 | 1937 | 2030 |

Pro vytvoření klientské aplikace bylo zvoleno VueJS. Rozhodujícím faktorem bylo především to, že Vue je plnohodnotný aplikační rámec, který má výkonnostní parametry srovnatelné s ReactJS, přičemž React je pouze zobrazovací knihovna.

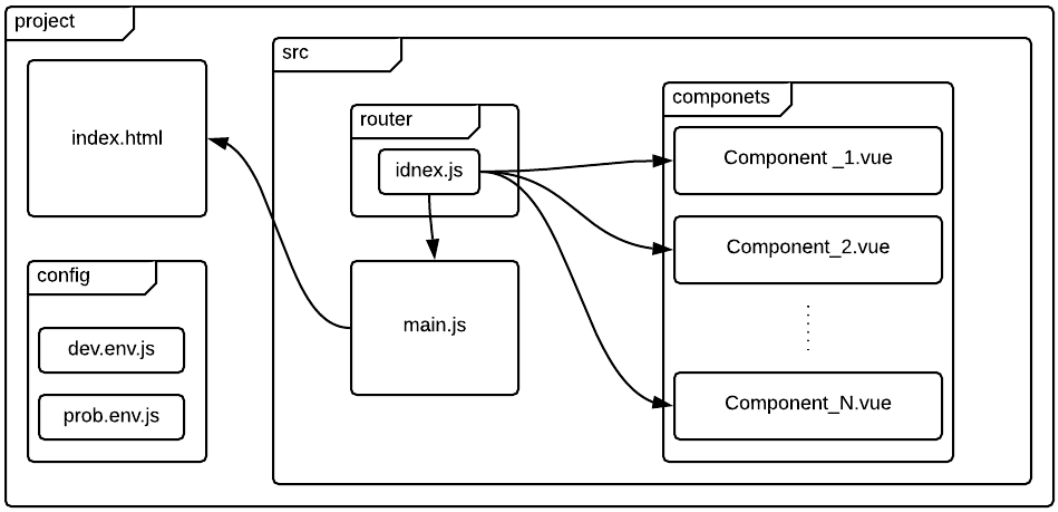
### 2.4.2 Aplikační rámec VueJS

V této části se práce zaměří na vysvětlení struktury VueJS a její základní vlastnosti. Jak lze nalézt na oficiálních stránkách organizace [12], Vue umožňuje dva módy běhu aplikace.

Prvním je vývojový mód, který se využívá při vytváření aplikace. Spouští se příkazem (npm run dev) z příkazového řádku v adresáři vytvořeného projektu Vue (příkaz pro inicializaci balíčku: vue init webpack název_projektu). Tento mód spustí virtuální server localhostu na volném portu. Pomocí webového prohlížeče je možné sledovat momentální stav aplikace, který se aktualizuje po každém uložení některého ze zdrojových souborů.

Druhý mód je určen pro reálné využití aplikace. Příkazem (npm run build) z příkazového řádku v adresáři projektu se spustí kompilace kódu. Po ukončení procesu jsou vytvořeny soubory v adresáři projektu ve složce „dist“. Obsah této složky se nahraje na server.

Obrázek 2.2 znázorňuje strukturu souborů, se kterými vývojář pracuje v rámci prvního módu pro vývoj aplikace ve VueJS. Níže jsou vysvětleny jednotlivé soubory.




Obrázek 2.2: Schéma konstrukce Vue ve vývojovém módu

**Komponenty** jsou soubory s koncovkou vue, kterými programátor vytváří aplikaci. Mohou vytvářet celou stránku, ale i část využitelnou v jiné komponentě. Tvoří jí tři oddělené části (template, script, style), jak je znázorněno v ukázce kódu 2.2. Template je HTML část (vzhled). Ve script se definují proměnné (data), metody (funkce), created (možnost inicializovat hodnoty při zavolání komponenty) atd. Možností funkcionalit Vue je hodně a všechny jsou popsány v dokumentaci aplikačního rámce. Poslední částí jsou styly, kde je prostor k definování vlastních kaskádových stylů.

```
<template>
  <div id="nazev_komponenty">
    ...
  </div>
</template>

<script>
  export default {
  name: 'nazev_komponenty',
  data () {
    return {
    }
  },
  methods: {
  },
  created: function(){
  }
</script>

<style scoped>
...
</style>

```

Ukázka kódu 2.2: VueJS struktůra komponenty

**Router** je část, ve které se ke komponentám přiřazují adresy. Vue má samostatný objekt pro směrování. V ukázce kódu 2.3 je znázorněno, jak se objekt vytváří. Mód history zajišťuje přesnou adresu, ve výchozím nastavení se před adresu vloží náhodný string, aby nebylo možné přes historii prohlížeče komponentu vyhledat. Následuje list objektů směrovačů. Nastavuje se URI adresa, jméno směrovače pro přesměrování a importovaná komponenta.

```
1  import Vue from 'vue'
2  import Router from 'vue-router'
3  import komponenta from '@/components/nazev_komponenty'
4  Vue.use(Router)
5  export default new Router({
6    mode: 'history',
7    routes: [{path: '/',name: 'komponenta', component:
     komponenta}]
8  })
9
10
```

Ukázka kódu 2.3: VueJS struktůra směrovače

**Main.js** je srdce aplikace. Obsahuje vytvoření instance Vue. V ukázce kódu 2.4 je vidět, jak může vypadat vytvoření instance Vue. Importují se routery, které ukazují na jednotlivé komponenty. Atributem template je možné definovat něco společného pro celou aplikaci (v template tagem router-view se přenáší vzhled jednotlivých komponent). Instanci Vue je možné přiřadit stejné vlastnosti jako komponentám. Ty jsou poté dosažitelné v celé aplikaci jako globální.

```
1  import Vue from 'vue'
2  import router from './router'
3  new Vue({
4  router,
5  template: `
6  <div id="app">
7  <router-view></router-view>
8  </div>`
9  }).$mount('#app')
10
```

Ukázka kódu 2.4: VueJS struktůra main.js

**Index.html** je jediný soubor, který zobrazuje uživateli aplikaci. Jedná se o klasické HTML obsahující hlavičku a tělo. Pro VueJS je klíčové umístit do těla div tag, přes který napojí main.js zobrazování komponent tím, že použije id tagu (ukázka kódu 2.4, 7. řádek).

```
<!DOCTYPE html>
<html>
  <head>
    ...
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>

```

Ukázka kódu 2.5: VueJS struktůra index.html

**Config** obsahuje dva souvory (dev.env.js a prod.env.js), v nichž se definují proměnné, které jsou závislé na módu VueJS. Oba soubory obsahují stejně nazvané proměnné s rozdílnými vlastnostmi, dle potřeby závislé na módu. Do kódů je možné pomocí struktury process.env.nazev_promene vložit data, která jsou rozdílná pro vývojový a distribuční mód. Pro vývojový mód se čerpají berou z dev.env.js, distribuce využije data z prod.env.js.

Proměnné, stejně jako metody nadefinované v datové oblasti komponent, jsou součástí objektu Vue a je nutné k nim přistupovat jako k instanci objektu (this). Jak bylo zmíněno výše, vlastnosti definované v main.js jsou dostupné pro všechny komponenty. K těmto vlastnostem mají přístup přes objekt Vue nazvaný root (např. this.$root.vlastnost_main).

### 2.4.3 Modul pro uplouvání fotografií

Pro aplikační rámec VueJS existuje velké množství externích knihoven, který usnadňují práci. Jednou z nich je i modul „vue-image-upload-resize“, jež slouží k nahrávání obrazu s možností volby velikosti ([13, Vue Image Upload and Resize]).

Modul je jedna komponenta VueJS, která zobrazí rámeček s možností nahrát obrázek. Po nahrání obrázku se v rámečku zobrazí jeho náhled. Důležité vstupní parametry jsou popsány níže.

**maxWidth** – Maximální šířka obrázku v pixelech.

**maxHeight** – Maximální výška obrázku v pixelech.

**maxSize** – Float hodnota pro maximální celkovou velikost v megapixelech.

**quality** – Kvalita obrázku, určuje se hodnotou 0 až 1,00.

**scaleRatio** – Změna měřítka obrázku, 1 – původní rozměr, 0.5 – poloviční.

**outputFormat** – Výstupní formát obrázku (base64, verbose, blob).

**accept** – Typy souborů pro upřesnění výběru HTML tagu input (např. image /*).

### 2.4.4 Knihovna pro získání dat z API

Pro komunikaci s externím serverem nemá VueJS v základní instalaci žádné prostředky. Je ale možné využít externí knihovnu „vue-resource“. Jak uvádí autor knihovny v dokumentaci [15], tato knihovna umožňuje vytvářet požadavky a zpracovávat odpovědi XMLHttpRequest a JSON.

V ukázce kódu 2.6 je vidět, jak se knihovna používá. Zavolá se funkce příslušná k metodě, jako vstupní parametr je dotazovaná adresa (mimo metodu GET se využívá i druhý parametr, pro tělo dotazu), poté se zavolá funkce „then“, která má dva vstupní parametry. U obou se jedná o funkci se vstupem odpovědi. První z nich je použita při úspěšné odpovědi, druhá naopak při chybě. Kritériem pro rozhodnutí, jestli je odpověď úspěšná, nebo neúspěšná, je číselný status odpovědi.

```
1 this.$http.get(https://api.foldynatulbp.cloud/v1/open/znacka).
   then(function(response){
2   // success response
3 }, function(response){
4   // error response
5 });
```

Ukázka kódu 2.6: Použití knihovny vue-resource

# 3 Specifikace požadavků

V této části práce jsou přesně specifikovány vlastnosti vyvíjené aplikace tak, aby mohla být pro rozsah této práce označena za dokončenou. Jedná se především o rozdělení typů uživatelů do kategorií a vymezení dat, se kterými bude aplikace pracovat.

## 3.1 Uživatelé

Pojem inzerce je nedílně spojen s dvěma typy uživatelů. Kupujícím a prodávajícím. Prvním z nich může být kdokoliv bez bližší specifikace. Naproti tomu prodávající může být uživatel, který prodává výjimečně svůj automobil, ale také profesionální prodejce, který se prodejem živí. Z uživatelského hlediska je proto vhodné tyto typy prodejců rozlišit. Specifikace typů uživatelů byla zvolena následovně.

**Kupující** – uživatel, který vyhledal aplikaci za účelem koupě vozu. Může volně vyhledávat v inzerátech. Nevyžaduje se autentizace.

**Soukromý prodejce** – uživatel, který vyhledal aplikaci za účelem prodeje vozu, ale prodej vozů není jeho běžnou aktivitou. Tento typ uživatele se sám registruje pomocí aplikace, přičemž při registraci musí přijmout právní podmínky, aby vznikl právní vztah mezi soukromým prodejcem a provozovatelem aplikace. Po registraci uživatel dostane možnost přidávat inzeráty bez omezení, které se zveřejní až po odsouhlasení provozovatele aplikace. Uživatel má plnou kontrolu nad svým účtem.

**Profesionální prodejce** – uživatel, který vyhledal aplikaci za účelem prodeje vozu a prodej vozů je jeho dlouhodobě provozovanou aktivitou (např. autobazar). Tento typ uživatele vytváří a edituje provozovatel aplikace na základě smlouvy. Po uzavření smlouvy přidá provozovatel aplikace uživatele do systému a následně mu zašle přihlašovací údaje. Uživatel bude moci vytvářet inzeráty do limitního počtu uvedeného ve smlouvě. Inzeráty tohoto uživatele budou okamžitě po vytvoření zveřejněny.

**Administrátor** – uživatel, který provozuje aplikaci. Disponuje nad aplikací kompletní kontrolou. CRUD operace nad všemi tabulkami.

Pro všechny registrované uživatele si aplikace bude ukládat následující informace: email, jméno, příjmení, adresu, město, PSČ, telefon, typ uživatele (dle výše uvedených kategorií), heslo pro ověření autentizace a informaci, zda je uživatel právnická osoba.

V případě, kdy je uživatel registrován jako právnická osoba, se k výše uvedeným informacím o uživateli přidávají následující informace o společnosti: název, sídlo (adresa, město, PSČ), IČ a DIČ.

Další informace o uživatelích se rozděluje podle typu uživatele následovně:

- Soukromý prodejce – datum posledního přihlášení, právní podmínka
- Profesionální prodejce – limitní počet inzerátů, faktury uživatele, uzavřené smluvní vztahy

## 3.2 Obsah inzerátu

Jako dostačující informace v inzerátu se předpokládá následující výčet údajů: tovární značka, modelová řada, seznam výbavy, kupní cena v korunách českých, možnost odpočtu DPH (v případě možnosti také uvedení kupní ceny bez DPH v korunách českých se základní sazbou DPH 21 %), VIN kód automobilu, počet najetých kilometrů (práce předpokládá pouze jednotky délky v metrické soustavě), rok výroby, měsíc a rok první registrace vozu, fotogalerie, individuální text a kontaktní údaje prodejce.

Aby se zamezilo chybám vzniklým zadáváním inzerátu uživatelem, omezí se tovární značky, modelové řady a seznam výbav na množinu atributů spravovaných administrátorem.

## 3.3 Kritéria vyhledávání inzerátů

Inzeráty se budou moci vyhledávat pomocí níže uvedených kritérií:

- značka
- model
- množina let výroby od – do
- množina najetých kilometrů od – do
- množina cen od – do
- množina vybavení vozidla

# 4 Návrh řešení

V této kapitole ja navržena aplikace na základě zadání a specifikace požadavků. Pro návrh se využilo prostředků uvedených v kapitole 2. Pro tvorbu databáze byly zvoleny SŘBD PostgreSQL (2.2), API s využitím Slim (2.3.2) a klientská aplikace pomocí VueJS (2.4.2).

## 4.1 Databáze

Z kapitoly 3.2 této práce vyplývá, že chtěná vlastnost aplikace je možnost spravovat značky automobilů, modelové řady a výbavy. Dále je nutné do návrhu databáze zahrnout požadavky na uživatele uvedené v kapitole 3.1 a klíčové informace o inzerátech. Z těchto požadavků byl vytvořen entitně relační diagram zobrazený na obrázku 4.1. Vysvětlení jednotlivých entit je uvedeno níže.

**značka** – Silná entita obsahující tovární značky. Pomocí ní si uživatel vybere značku inzerovaného vozidla (při vytváření inzerátu a při možnostech vyhledávání).

**výbava** – Silná entita zahrnující výbavu a kategorii vozu. Kategorie rozčleňuje výbavu (např. hlídání mrtvého úhlu – Asistent, pohon 4 x 4 – Technické, atd.). Zde se nabízí možnost vytvoření nezávislé entity kategorií. V práci však bylo vybrání zachování atributů v jedné entitě.

**model** – Slabá entita, složený primární klíč z identifikace značky a vlastního identifikátoru (každá značka vyrábí více modelů). K této entitě jsou přidány atributy ohraničující dobu, kdy se model vyráběl. Pomocí těchto atributů se uživateli budou moci nabídnout pouze ty roky, kdy se model skutečně vyráběl. Důležitým atributem je pole N identifikátorů výbavy. Díky tomuto atributu je možné přiřadit ke každému modelu pouze takovou výbavu, která se k němu váže. Při zadávání nebo vyhledávání inzerátu si uživatel vybere značku a poté se mu teprve zobrazí modely vázané k této značce (tím se upraví i možnosti výbavy a roky výroby).

**uživatel** – Silná entita, která pokryje všechny uživatelské údaje nezávislé na typu uživatele.

**všeobecné podmínky** – Silná entita podmínky vymezující pravidla pro neprofesionální prodejce.

**právnické osoby** – Slabá entita, závisí na uživateli. Přidává k němu doplňující informace o firmě, pokud je uživatel právnická osoba. Tato závislost se detekuje atributem v entitě uživatel pomocí booleovské proměnné.

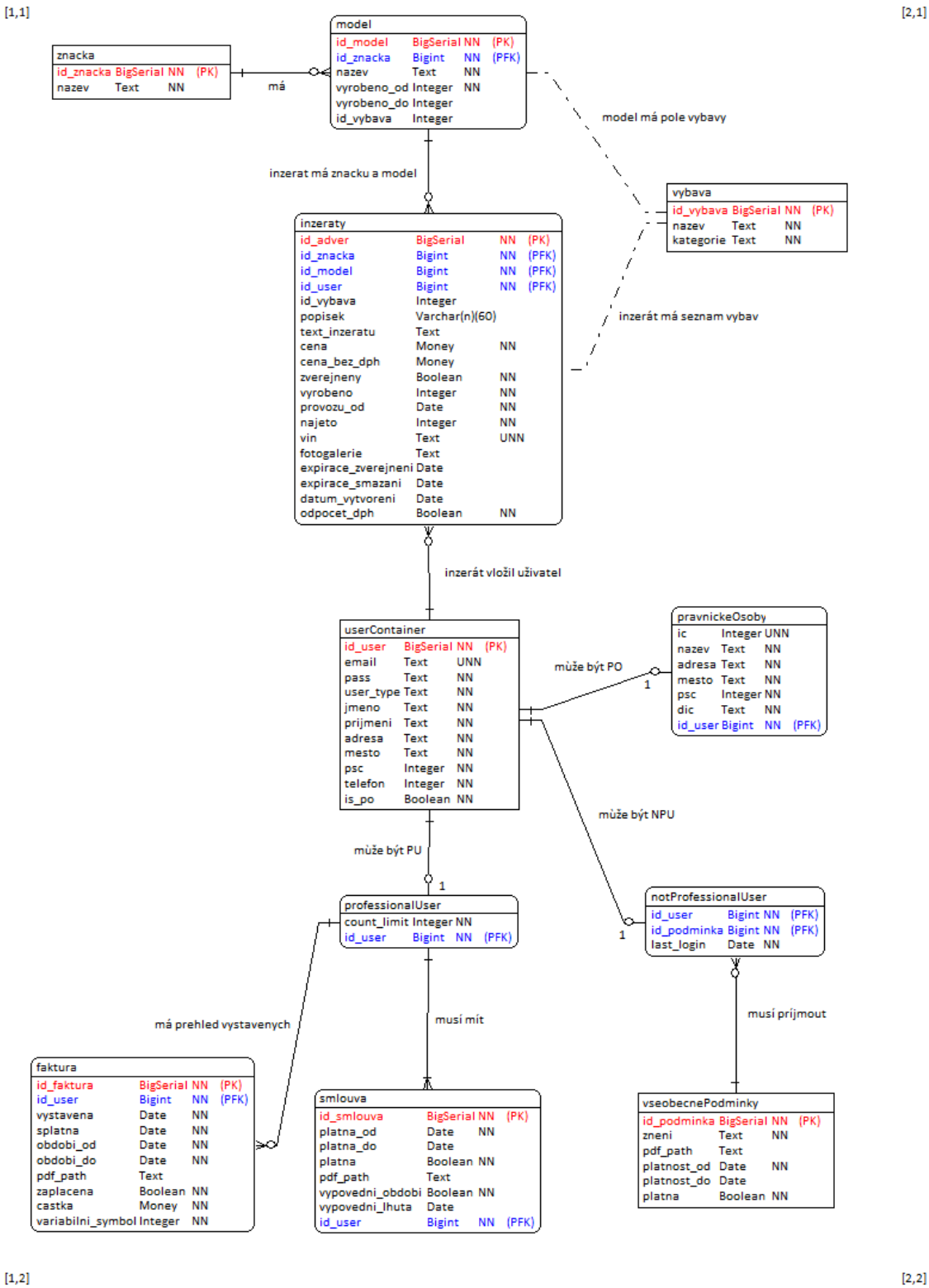
**neprofesionální uživatel** – Slabá entita, sloužící k rozšiřujícím informacím pro neprofesionální prodejce. Uchovává hodnotu posledního přihlášení a uživatelem přijaté podmínky.

**profesionální prodejce** – Slabá entita rozšiřující informace vztahující se k profesionálnímu prodejci. Uchovává limitní počet inzerátů.

**smlouva** – Slabá entita navázaná na profesionálního prodejce. Slouží k přehledu smluvních vztahů mezi prodejcem a provozovatelem serveru.

**faktura** – Slabá entita spojená s profesionálním prodejcem. Slouží k přehledu faktur vystavených prodejci.

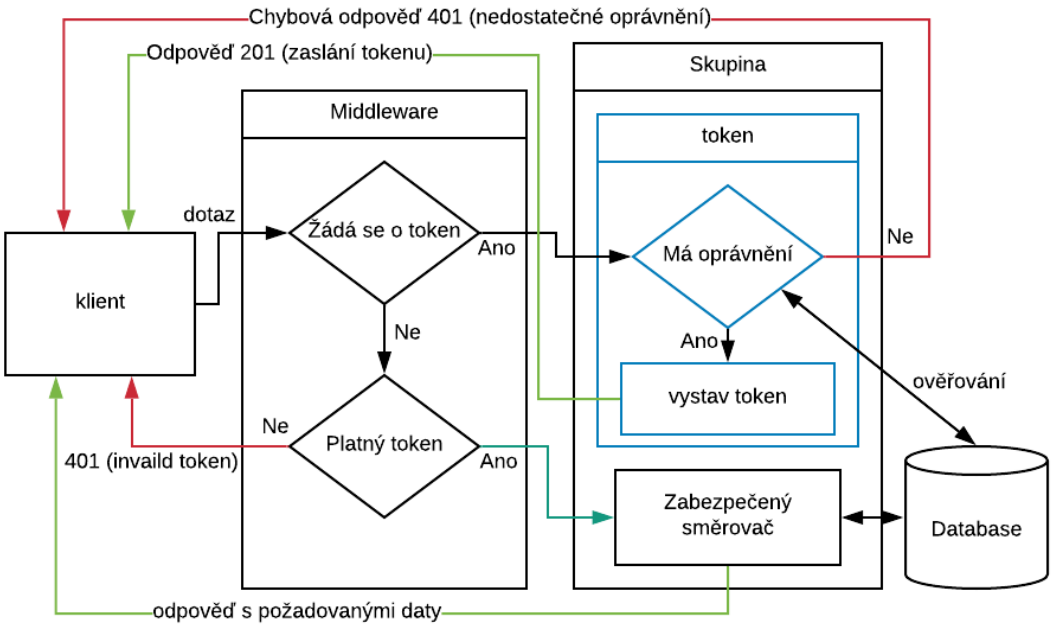
**inzeráty** – Slabá entita vytvořených inzerátů. Je dána značkou vozidla, modelem a uživatelem (prodejcem).




Obrázek 4.1: Entitně relační model

## 4.2 API

Při návrhu API se řešilo kromě potřebných směrovačů pro získání dat z databáze i jejich zabezpečení. Aplikační rámec Slim, jak je popsáno v kapitole 2.3.2, umožňuje routery seskupit do různých skupin a na ty poté navázat jednotlivé Middlewary. Chtěné skupiny se dají pomocí JWT (kapitola 2.3.3) a Middlewaru zabezpečit tím, že middleware nepropustí dotazy, které nebudou obsahovat platný token, a v každé skupině bude jeden směrovač, který bude ověřovat uživatele a generovat token. Tento směrovač bude opatřený výjimkou tak, aby ho middleware vždy propouštěl.




Obrázek 4.2: Schéma zabezpečení pomocí Middlewaru a JWT

Na obrázku 4.2 je zobrazeno schéma přístupu do zabezpečené části API. Klient pošle dotaz na získání tokenu, Middleware rozpozná, že jde o žádost směřující na výjimku pro získání tokenu, a propustí ho k směrovači, který ověří práva uživatele. Je-li uživatel oprávněný, vygeneruje se JSON Web Token a zašle se klientovi zpět. Klient po obdržení tokenu pošle dotaz obsahující token na zabezpečený směrovač, Middleware ověří platnost tokenu a po úspěšném ověření propustí dotaz k zabezpečenému směrovači.

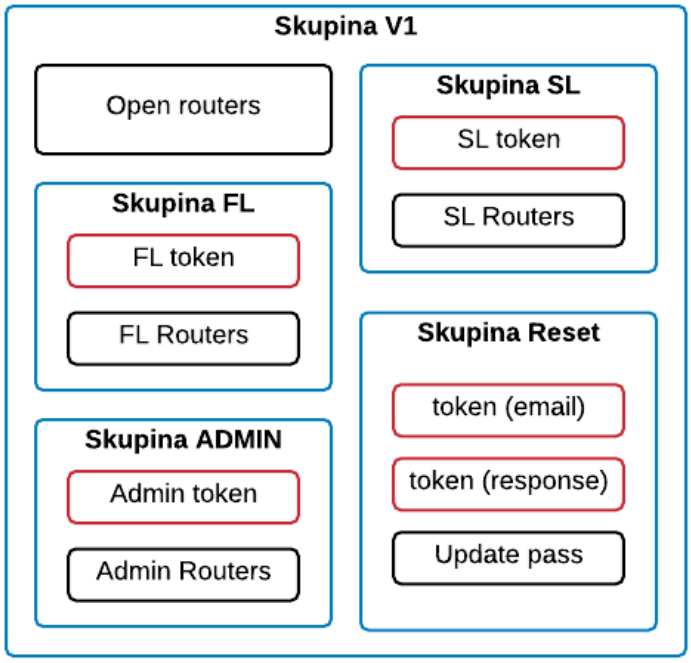
Na základě specifikace požadavků bylo provedeno rozdělení do skupin zabezpečení podle typu uživatele (zobrazeno na obrázku 4.3). Pro možnost budoucího verzování je celá API seskupena do skupiny „v1“. Tato skupina obsahuje volně přístupné směrovače a čtyři zabezpečené skupiny, které jsou vysvětleny níže.

**Skupina FL** – sada směrovačů pro všechny přihlášené uživatele. Časová platnost tokenu: 2 hodiny. Token slouží i jako směrovač pro přihlášení. Vrací včetně tokenu a jeho expirační doby i všechny údaje o uživateli.

**Skupina SL** – sada směrovačů pro profesionální prodejce a administrátora. Časová platnost tokenu: 2 minuty.

**Skupina ADMIN** – sada směrovačů pro administrátora. Časová platnost tokenu: 2 minuty.

**Skupina Reset** – sada směrovačů pro změnu hesla, přístup pro všechny registrované uživatele, dva druhy tokenů (jeden posílá token pomocí emailu – zapomenuté heslo, druhý slouží k změně hesla z aplikačního prostředí). Časová platnost tokenu: 10 minut (email), 2 minuty (aplikace).

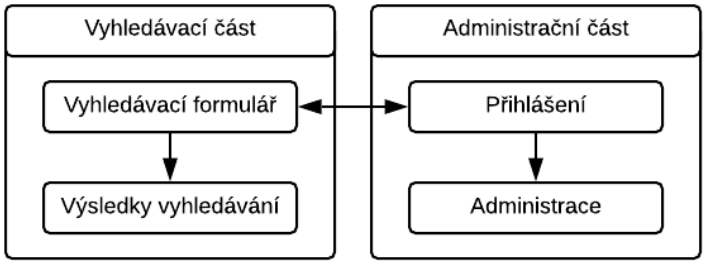



Obrázek 4.3: Rozdělení API do skupin podle zabezpečení

Dokumentace směrovačů využitých v práci nelze kvůli jejich rozměrům vložit do její písemné části. Proto byla vytvořena dokumentace pomocí programu Postman, která je zveřejněna na adrese: https://documenter.getpostman.com/view/4294049/S1EJX1fz

## 4.3 Klient

Klientská aplikace představuje nejrozsáhlejší problém zadání. Proto bylo rozhodnuto o jejím rozdělení na dvě oddělené části. První slouží k vyhledávání inzerátů a druhá k administrativě. Obě části jsou samostatné aplikace běžící na rozdílných subdoménách.




Obrázek 4.4: Schéma struktury klientské části aplikace

Struktura je znázorněna na obrázku 4.4. Vyhledávací část slouží uživatelům, kteří vyhledávají inzeráty. Je složena z vyhledávacího formuláře a výsledků hledání. Administrační část slouží ke správě aplikace. Má do ní přístup pouze přihlášený uživatel a její funkčnost se omezuje na základě typu uživatele následovně.

**Soukromý prodejce** – editace vlastního účtu, seznam vložených inzerátů, vložení a editace inzerátu

**Profesionální prodejce** – seznam vložených inzerátů, vložení a editace inzerátu, seznam vystavených faktur, seznam uzavřených smluv

**Administrátor** seznam vložených inzerátů, vložení a editace inzerátu, správa uživatelů (editace, vytvoření účtu profesionálního prodejce a administrátora, správa profesionálního prodejce – vystavení faktur a smluv), správa databáze (značek, modelů, výbavy)

### 4.3.1 Vyhledávací část

Titulní stránka obsahuje vyhledávací formulář dle parametrů stanovených v kapitole 3.3. Po uživatelské akci na vyhledání inzerátů se zobrazí jednotlivé výsledky s možností úpravy kritérií a opětovného vyhledávání.

Pro výše uvedené vlastnosti byla vytvořila samostatná aplikace pomocí VueJS (2.4.2) s následujícím rozvržením komponent.

**Navbar** – Horní lišta menu s odkazem na administrační část.

**MainSearch** – Vyhledávací formulář. Slouží jako hlavní stránka, ale při inicializaci je schopna rozpoznat, že je volaná z jiné směrovací adresy, a tím upravit svůj vzhled tak, aby mohla být použita i u výsledků vyhledávání pro další upřesnění kritérií.

**AdvertisingShortCard** – Definuje vzhled výsledků vyhledávání.

**LoadingComp** – Design načítání pro informování uživatele, než aplikace dostane výsledky.

**SearchResult** – Stránka výsledků vyhledávání. Využívá komponenty MainSearch, AdvertisingShortCard a LoadingComp. Pokud byl poslán požadavek na vyhledávání, ale aplikace ještě nemá odpověď, zobrazuje se LoadingComp. Po obdržení výsledků zobrazí výsledky hledání (AdvertisingShortCard).

Proměnné pro kritéria a výsledky vyhledávání byly z důvodů používání ve více komponentách definovány v rámci main.js jako globální proměnné.

### 4.3.2 Administrativní část

Při vstupu se zobrazí přihlašovací stránka, na níž se bude moci uživatel přihlásit, případně soukromý prodejce vytvořit účet. Po přihlášení vstoupí do administračního prostředí, kde bude mít možnost provádět operace uvedené v kapitole 4.3.

Tato část byla opět vytvořena jako samostatná aplikace pomocí VueJS. Z důvodu zajištění přístupu pouze přihlášeného uživatele se do main.js zadefinovala globální proměnná s informacemi o uživateli, která se při inicializaci aplikace nastavuje na null hodnotu. Každá směrovací komponenta pak při své inicializaci zjišťuje, jestli je tato proměnná prázdná. V případě, že ano, přesměruje se na přihlašovací stránku. Po úspěšném přihlášení se do globální proměnné uloží informace o uživateli a tím se zpřístupní ostatní komponenty administrační části. Níže je uveden seznam použitých komponent.

**NavbarTop** – Horní lišta menu. V případě přihlášeného uživatele zobrazuje jeho typ a možnost odhlášení. Pro případ, kdy není ještě uživatel přihlášen, zobrazí odkaz na vyhledávací část.

**NavbarLeft** – Levá lišta menu. Zobrazuje nabídku možností pro přihlášeného uživatele podle jeho typu.

**Login** – Přihlašovací formulář. Přihlášení za pomocí zadání emailu a hesla. Součástí je odkaz na vytvoření nového soukromého prodejce a funkce zapomenutého hesla. Zahrnuje i kontrolu platnosti podmínky pro soukromého prodejce.

**AddNpUser** – Formulář pro přidání soukromého prodejce.

**AdminHome** – Informace o přihlášeném uživateli a odkaz na změnu hesla. V případě soukromého prodejce odkaz na úpravu jeho účtu.

**EditNpUser** – Formulář pro editaci účtu soukromého prodejce.

**TableOfAdvUser** – Tabulka s inzeráty uživatele. Obsahuje i odkaz na podrobnosti inzerátu.

**AdvList** – Komponenta pro zobrazování inzerátů přihlášeného uživatele. Využívá TableOfAdvUser.

**AdvDetails** – Podrobné informace konkrétního inzerátu s funkcí smazání celé inzerce a odkazem na možnost úpravy daného inzerátu.

**myUploaderImages** – Upravená komponenta knihovny (2.4.3) pro zpracování nahrávaných obrázků tak, aby bylo možné dynamicky přidávat více komponent této knihovny a tím pracovat s více obrázky.

**EditAdvertising** – Úprava konkrétní inzerce. Využívá myUploaderImages pro přidání, odebrání a změnu fotek do fotogalerie inzerátu.

**AddAdvertising** – Přidání inzerátu. Komponenta zahrnuje kontrolu překročení limitního počtu inzerátů profesionálního prodejce (při počtu vložených >= než limitní počet nedovolí vložit inzerát). Podobně jako EditAdvertising využívá komponentu myUploaderImages

**AdvForCheck** – Komponenta na kontrolu inzerátů soukromého prodejce (dosud nezveřejněné inzeráty).

**UserList** – Seznam všech uživatelů aplikace ve zkrácené podobě (jméno, příjmení, email, typ uživatele) s odkazem na podrobnosti o uživateli.

**TableForInvoice** – Tabulka pro seznam faktur profesionálního prodejce. Vstupními parametry může povolit editaci a smazání faktury, nebo tyto funkcionality zakázat.

**TableForContracts** – Tabulka pro seznam smluvních vztahů mezi profesionálním prodejcem a provozovatelem. Vstupními parametry povoluje, nebo zakazuje editaci a mazání smluvních vztahů.

**UserDetail** – Kompletní informace o uživateli s možností smazání uživatele a odkazem na jeho úpravu. Obsahuje TableOfAdvUser pro přehled uživatelových inzerátů s možností jejich správy. V podrobnostech profesionálního prodejce je též zahrnutá možnost přidávání faktur a smluvních vztahů. K přehledu a správě (editace, smazání) využívá komponenta TableForInvoice a TableForContracts.

**EditUser** – Komponenta pro úpravu informací o uživateli (nelze měnit jeho typ).

**AddUser** – Přidání uživatele typu profesionální prodejce, nebo administrátor.

**CarBrandManagement** – Správa automobilových značek. Obsahuje tabulku všech značek v databázi s možností editace a smazání. Součástí komponenty je i přidání značky do databáze.

**ModelManagementList** – Seznam všech modelů v databázi s odkazem na jejich podrobnosti. Zahrnuje funkcionalitu přidání modelu do databáze.

**ModelManagementDetails** – Podrobné informace o modelu s možností jeho správy (úprava, smazání).

**GearManagement** – Seznam všech výbav v databázi včetně správy databáze výbavy.

**ConditionsManagmentList** – Seznam všech vydaných podmínek s odkazem na jejich podrobnosti. Komponenta obsahuje též odkaz na vytvoření nové podmínky.

**ConditionsManagementAdd** – Formulář pro vytvoření nové podmínky.

**ConditionsManagementDetails** – Obsahuje podrobné informace o podmínce včetně jejího znění. Zahrnuje funkcionalitu smazání podmínky (pokud již podmínka není využívána) a úpravu informace, dokdy je podmínka platná (včetně boolovské proměnné – „platna“).

**PuMyInvoice** – Komponenta zobrazující profesionálnímu prodejci přehled jeho faktur. Pro zobrazení využívá komponentu TableForInvoice bez možnosti správy dat.

**PuMyContracts** – Slouží profesionálnímu prodejci k výpisu všech smluvních vztahů mezi ním a provozovatelem aplikace. Zobrazení provádí pomocí TableForContracts bez možnosti správy dat.

**ResetPass** – Formulář pro změnu hesla.

Jak je popsáno v kapitole 2.4.2, VueJS funguje jako celistvá aplikace v rámci jednoho HTML souboru a pro zobrazování komponent využívá směrovače, na které jsou napojeny jednotlivé komponenty. Napojení je znázorněno v tabulce 4.1.

Tabulka 4.1: Směrovací adresy administrační části klientské aplikace

| Cesta | Komponenta | Dostupnost |
|---|---|---|
| / | AdminHome | NPU, PU, ADMIN |
| /login | Login | Všichni |
| /addNpUser | AddNpUser | Všichni |
| /resetPass/:id_user/token/:token | ResetPass | NPU, PU, ADMIN |
| /addAdvertising | AddAdvertising | NPU, PU, ADMIN |
| /advList | AdvList | NPU, PU, ADMIN |
| /advDetails/:id_adver/user/:id_user | AdvDetails | NPU, PU, ADMIN |
| /EditAdvertising/:id_adver | EditAdvertising | NPU, PU, ADMIN |
| /editNpUser | EditNpUser | NPU |
| /usersList | UsersList | ADMIN |
| /userDetail/:id_user | UserDetail | ADMIN |
| /addUser | AddUser | ADMIN |
| /editUser/:id_user | EditUser | ADMIN |
| /advForCheck | AdvForCheck | ADMIN |
| /gearManagement | GearManagement | ADMIN |
| /carBrandManagement | CarBrandManagement | ADMIN |
| /modelManagementList | ModelManagementList | ADMIN |
| /modelManagementDetails | ModelManagementDetails | ADMIN |
| /puMyInvoice | PuMyInvoice | PU |
| /puMyContracts | PuMyContracts | PU |
| /conditionsManagementList | ConditionsManagement List | ADMIN |
| /conditionsManagement Details/:id_podminka | ConditionsManagement Details | ADMIN |
| /conditionsManagementAdd | ConditionsManagement Add | ADMIN |

# 5 Realizace řešení

V této kapitole je popsáno, jak probíhala tvorba a implementace podle návrhu, který je popsán v kapitole 4. Začalo se konfigurací serveru, po jeho zprovoznění bylo naprogramováno REST API a nakonec byla naprogramována i klientská aplikace pro webové prohlížeče.

## 5.1 Konfigurace serveru

Pro práci bylo využito řešení pomocí cloudových serverů. Celá aplikace funguje na jednom virtuálním serveru s jednou veřejnou IP adresou. Server používá operační systém CentOS 6. Ke cloudovému serveru byla přikoupena i doména foldynatulbp.cloud, která byla napojena na IP adresu serveru. Níže je uveden popis kroků konfigurace serveru, které byly nutné pro zprovoznění aplikace.

### 5.1.1 Databáze

Prvním krokem byla instalace SŘBD. Nejdříve byl přidán balíček obsahující instalační soubory PostgreSQL 10 (rpm -Uvh <adresa balicku>) a poté byla provedena instalace (yum install postgresql10-server postgresql10). Po instalaci bylo nutné inicializovat databázi (/usr/pgsql-10/bin/postgresql-10-setup initdb). Posledním krokem ke zprovoznění databáze bylo zapnutí databázového serveru (service postgresql-10 start).

Pro vzdálený přístup k databázi pomocí administrační aplikace PgAdmin 4 byla přidána do konfiguračního souboru pg_hba.conf IP adresa vzdáleného počítače. Po přidání adresy bylo možné se vzdáleně připojit k databázovému serveru a vytvořit databázi pro aplikaci.

Pomocí programu CASE Studio 2, ve kterém byl vytvořen entitně relační diagram, se vygeneroval CREATE script, jenž byl použit pro vytvoření tabulek v databázi. Následně byl napsán a poté spuštěn INSERT script k naplnění databáze testovacími daty.

Po naplnění databáze testovacími daty bylo zjištěno, že PostgreSQL inicializoval své konfigurační hodnoty do hodnot, které byly nastaveny na operačním systému v době inicializace první databáze. Instalace CentOS 6 na serveru měla US konfiguraci, takže i PostgreSQL inicializoval hodnoty do US standardu. To zapříčinilo, že proměnné typu Money v DB se zobrazovaly v oficiálním standardu pro výpis US dolarů, přičemž chtěný standard byl

CZ. Problém se podařilo vyřešit změnou proměnné lc_monetary na CZ standard (cs_CZ.iso88592) v konfiguračním souboru postgresql.conf (/var/lib/pgsql/10/data/postgresql.conf).

### 5.1.2 Apache Hosts s ověřeným certifikátem

Z kapitoly 4 plyne, že aplikace je složena ze tří samostatných aplikací (API, vyhledávací a administrační). Z tohoto důvodu byl i server pomocí virtuálních hostů nakonfigurován tak, aby odkazoval na každou aplikaci zvlášť na základě subdomén (www – vyhledávací, api – REST API, admin – administrační). Vzhledem k faktu, že aplikace pracuje s citlivými daty uživatelů, bylo nutné zabezpečit komunikaci klienta se serverem šifrovacím protokolem HTTPS, a to pro všechny tři subdomény.

Pro získání ověřeného certifikátu byl zvolen vystavitel certifikátů „Let's Encrypt Authority X3". Ten umožňuje vytvořit ověřený certifikát na registrované domény s platností tři měsíce. Poté se musí obnovit. Pomocí postupu uvedených na oficiální stránce organizace https://www.sslforfree.com/ získá uživatel tři soubory (certificate.crt, ca_bundle.crt, private.key), které použije k zabezpečení jednoho virtuálního hostu.

K definování virtuálních hostů byl využit konfigurační soubor /etc/httpd/conf.d/ssl.conf. V ukázce kódu 5.1 je uveden příklad definice zabezpečeného virtuálního hostu.

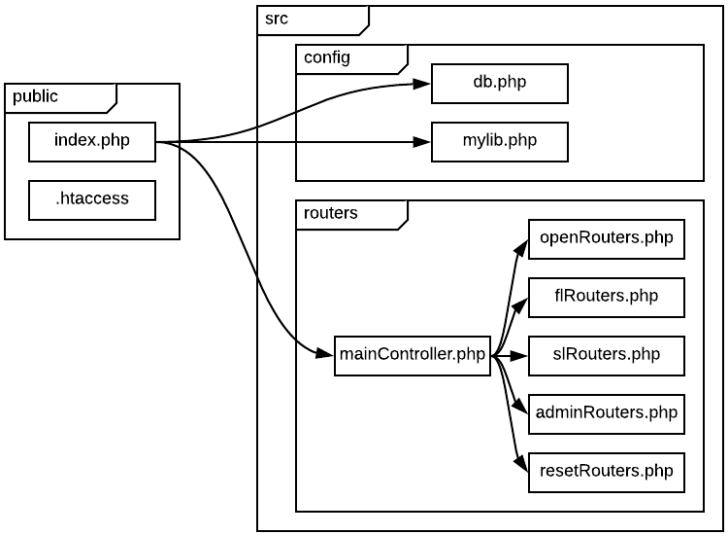
```
<VirtualHost *:443>
  ServerName www.foldynatulbp.cloud:443
  ServerAlias foldynatulbp.cloud
  DocumentRoot /var/www/html/public
  SSLEngine on
  SSLCertificateFile /etc/ssl/certs/lets_encrypt_authority/
   certificate.crt
  SSLCertificateKeyFile /etc/ssl/private/lets_encrypt_authority/
   private.key
  SSLCertificateChainFile /etc/ssl/certs/lets_encrypt_authority/
   ca_bundle.crt
</VirtualHost>
```

Ukázka kódu 5.1: Definice virtuálního hostu

## 5.2 REST API

Aplikační rámec Slim nemá jasně stanovenou strukturu uspořádání souborů. V práci bylo přistoupeno, jak je znázorněno na obrázku 5.1, k rozdělení na konfigurační soubory, směrovače a veřejnou část, na kterou ukazuje konfigurace Apache. Níže je uveden popis jednotlivých souborů.

**Index.php** obsahuje základní inicializaci Slim a příkazy na zahrnutí souborů db, mylib a mainController.




Obrázek 5.1: Schéma napojení souborů

**Soubor .htaccess** konfiguruje Apach server tak, aby vždy odkazoval na soubor index.php

**Db.php** definuje třídu pro připojení databáze pomocí PDO. Třída obsahuje pevně definované proměnné pro objekt PDO a jednu funkci, která vrátí spojení s databází.

**Mylib.php** je soubor podpůrných funkcí, které se využívají ve směrovačích. Například obsahuje funkci na složení SQL dotazu pro vyhledávání inzerátu.

**MainController.php** definuje Slim skupiny s Middlewary. V každé skupině je příkaz na zahrnutí souboru dané skupiny dle kapitoly 4.2.

**Směrovače** jsou soubory openRouters, flRouters, slRouters, adminRouters a resetRouters. V každém z nich jsou definované jednotlivé směrovače dané skupiny.

Níže jsou uvedeny problémy, které kromě samotného naprogramování jednotlivých směrovačů bylo nutné řešit, a popis jejich řešení.

**Zabezpečení neveřejných skupin** – Na každou zabezpečenou skupinu je napojený Middleware, který obsahuje knihovnu pro ověřování tokenu (2.3.4). Do objektu knihovny se definuje zabezpečená cesta, výjimka pro cestu k tokenu, parametr „secret“ (jedinečný pro každou skupinu, aby se tokenem z jedné skupiny nedalo dostat do druhé) a chybová funkce (vrací odpověď o chybě – text knihovny s HTTP statusem 401).

**Získání tokenu** – Každá skupina má svůj vlastní směrovač pro získání tokenu. Po ověření uživatele žádajícího o token vygeneruje JWT s expirační dobou a zabezpečí ho parametrem „secret“ dané skupiny. Doby expirací jsou uvedeny v kapitole 4.2.

**CROS** – Na celou aplikaci je použit Middleware, který všem odchozím odpovědím přiřadí do hlavičky Cross-origin resource sharing. Bez CROS by nemohla API fungovat jako samostatná aplikace. Žádosti z jiných domén by prohlížeče zablokovaly.

## 5.3 Klientská aplikace

Jak již bylo zmíněno v kapitole 4.3, aplikační část tvoří dvě samostatné aplikace, vytvořené pomocí VueJS. Obě běží na samostatných subdoménách a jsou na sobě nezávislé.

Pro získání dat používají jednotlivé Vue komponenty funkce, kterými se dotazují do databáze. V těchto funkcích je použita knihovna „vue-resource“, která je popsána v 2.4.4.

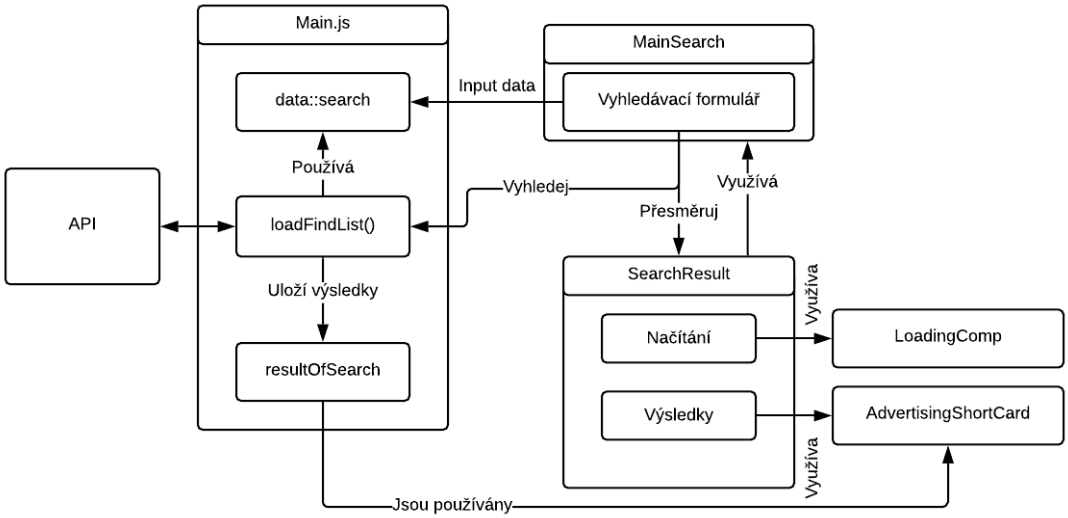
Protože vývoj aplikace byl prováděn na localhostu, bylo využito konfiguračních souborů Vue (config) popsaných v kapitole 2.4.2 pro definování proměnných, které nesou informaci o adrese API aplikace. V dotazech na API se proto objevuje proměnná „process.env.API_URL“, která definuje adresu API. Ve vývojovém módu odkazuje na localhost a v produkčním na reálnou adresu API.

### 5.3.1 Vyhledávací část

Vyhledávací aplikace byla naprogramována pomocí komponent a jejich vlastností uvedených v kapitole 4.3.1. Propojení komponent je znázorněno na obrázku 5.2.

**Main.js** – Z důvodu celistvosti řešeného problému této části (vyhledávání inzerátů) bylo přistoupeno ke sdílení kritérií, výsledků a funkce vyhledávání mezi komponentami. Proto jsou tyto dva datové objekty a jedna funkce umístěny přímo do instance Vue v main.js. Výsledky vyhledávání jsou rozděleny do dvourozměrného pole, z nichž každé vnořené pole má maximálně 10 položek, aby při větším množství mohly být výsledky zobrazeny po deseti inzerátech na stranu. V tomto souboru jsou definovány i proměnné indexOfSearchPage a isBusy. První z nich slouží k indexování první souřadnice dvourozměrného pole výsledků vyhledávání a druhá signalizuje průběh načítání inzerátů.

**MainSearch** – Komponenta, která slouží i jako výchozí cesta směrovače. Obsahuje vyhledávací formulář pro zadávání kritérií vyhledávání včetně




Obrázek 5.2: Schéma propojení komponent vyhledávací části

funkcí pro načítání možností voleb. Po uživatelském požadavku na vyhledávání se zavolá funkce loadFindList, vynuluje se proměnná indexOfSearchPage a přesměruje se na komponentu SearchResult.

**LoadingComp** – Definuje pouze HTML a CSS pro signalizování načítání.

**AdvertisingShortCard** – Komponenta pro seznam a vzhled vyhledaných inzerátů. Pomocí cyklu v-for zobrazí všechny položky z resultOfSearch na indexu indexOfSearchPage. Výsledky jsou zobrazovány jako tlačítka s funkčností otevření modálního okna s detaily o inzerátu.

**SearchResult** – Importuje komponenty MainSearch, LoadingComp a AdvertisingShortCard. Pokud nemá aplikace dosud výsledky vyhledávání, zobrazuje komponentu LoadingComp, v opačném případě AdvertisingShortCard.

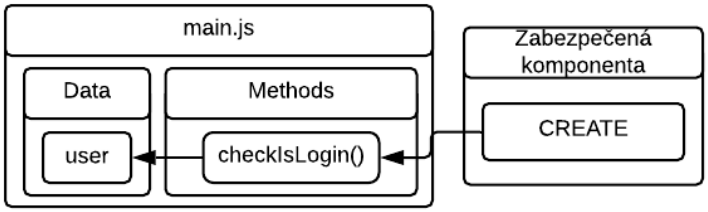
V rámci aplikace je vytvořena elektronická dokumentace k vyhledávací části dostupná na adrese: https://doc-www.foldynatulbp.cloud.

### 5.3.2 Administrativní část

Administrační část byla naprogramována dle návrhu uvedeného v kapitole 4.3.2. Včetně naprogramování výše zmíněných komponent bylo řešeno zabezpečení přístupu tak, aby se do administrační části nedostal nepřihlášený uživatel.

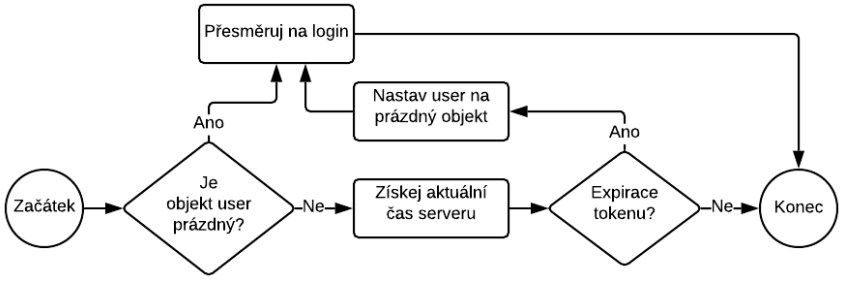
Jak je znázorněno na obrázku 5.3, každá komponenta, která má být zobrazena pouze přihlášenému uživateli, volá ve své oblasti create (oblast kom-

ponenty VueJS, která se volá před jejím zobrazením) funkci „checkIsLogin“ z main.js.




Obrázek 5.3: Schéma zabezpečení komponent

Funkce „checkIsLogin“ kontroluje přihlášeného uživatele, jak je znázorněno na obrázku 5.4, tím, že zjišťuje, zda je objekt „user“ prázdný (výchozí nastavení). Ten se totiž naplňuje daty z dotazu na získání tokenu pro FL skupinu (při úspěšné shodě uživatelského emailu a hesla v přihlašovacím formuláři). Pokud je objekt „user“ prázdný, přesměruje zobrazení místo požadované komponenty na komponentu pro přihlášení. V opačném případě funkce požádá o aktuální čas serveru a porovná ho s údajem o expiraci tokenu přihlášeného uživatele. Jestliže je token již neplatný, odhlásí uživatele vynulováním objektu „user“ a opět přesměruje na komponentu pro přihlášení s informací, že uživatel byl přihlášený příliš dlouhou dobu. Nevypršela-li expirační doba, funkce se ukončuje bez vlivu na aplikaci a tím umožní zobrazit požadovanou komponentu.




Obrázek 5.4: Algoritmus funkce checkIsLogin

Jak již bylo zmíněno v návrhu (4.3.2), MyUploderImages je lehce upravená komponenta knihovny pro práci s nahráváním obrázku (2.4.3). Úprava byla provedena, aby bylo možné pracovat s více obrázky. Byl přidán vstupní parametr indexOfArray (index přiřazený z nadřazené komponenty), který se vypisuje i ve výstupu po nahrání obrázku. Dále bylo přidáno tlačítko pro smazání komponenty a výstupní parametr pro informování nadřazené komponen-

ty o indexu smazané komponenty. Po výše uvedených úpravách bylo možné dynamicky pracovat s MyUploderImages v komponentě AddAdvertising.

Pro dotazování do skupiny FL se využívá token poslaný po přihlášení uživatele uložený v objektu „user“. Pro dotazy do skupin SL a Admin se z důvodu časové expirace tokenu nejprve zašle žádost na získání tokenu a poté se teprve dotazuje na konkrétní dotaz. V ukázce kódu 5.2 je znázorněno, jak je v práci naprogramováno dvojí dotazování pomocí knihovny „vue-resource“. Konkrétní dotaz je vnořený do úspěšné odpovědi žádosti o token a používá v hlavičce odpověď s vygenerovaným tokenem (řádek 8).

```
1  let param = {
2      id_user: this.$root.user.id_user,
3      pass: this.$root.user.pass
4    }
5  this.$http.post(process.env.API_URL+'v1/admin/token', param).
       then(function(response){
6    this.$http.get(process.env.API_URL+'v1/admin/users', {
7      headers: {
8        'Authorization': 'Bearer '+ response.body.token,
9        'Accept': 'application/json'
10     }
11   }).then(function(response){
12     this.userDetail = response.body;
13   }, function(response){
14     // error from request to v1/admin/users
15   });
16 },function(response){
17   // error from request to v1/admin/token
18 });
```

Ukázka kódu 5.2: Dvojfázové dotazování na zabezpečené skupiny

Pro šifrování hesla uživatele byl použit algoritmus SHA512. Heslo je v databázi uloženo zašifrované, přičemž k zašifrování dochází v klientské aplikaci před tím, než je poslán požadavek do API.

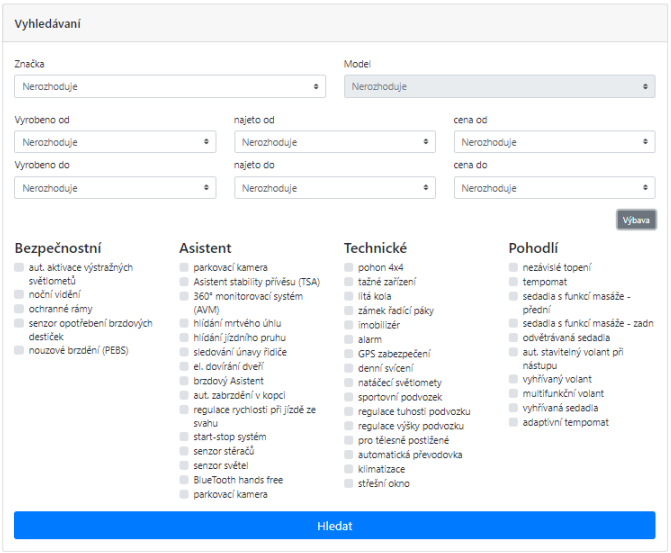
Podobně jako pro vyhledávací část je i pro administrační část vytvořena elektronická dokumentace dostupná na adrese: https://doc-admin.foldynatulbp.cloud.

# 6 Uživatelský popis aplikace

V této kapitole je zodpovězeno několik základních otázek potencionálního uživatele aplikace. Otázky jsou položeny podle typu uživatele tak, aby mu pomohly lépe se orientovat ve vytvořené aplikaci.

## 6.1 Kupující uživatel

**Jak vyhledat inzerát?** Po vstupu do aplikace se na stránce https://foldynatulbp.cloud/ zobrazí vyhledávací formulář (obrázek 6.1). Pomocí něhož omezujete množinu inzerátů. Po kliknutí na tlačítko „Hledat“ se načtou výsledky vyhledávání.




Obrázek 6.1: Vyhledávací formulář

**Jak zjistit více informací o inzerovaném vozidle?** Kliknutím na kartu inzerátu se zobrazí jeho podrobnosti. Jsou rozděleny do tří sekcí. První sekce obsahuje informace o vozidle, druhé fotografie vozidla a třetí kontakt na prodejce.

## 6.2 Všichni registrovaní uživatelé

**Jak vložit inzerát?** Po přihlášení do administrační části klikněte v levém menu na Správa inzerátů →Přidat inzerát.

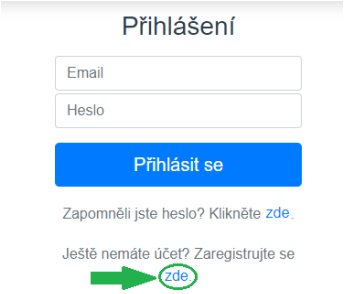
**Jak změnit heslo?** Po přihlášení do administrační zóny v sekci Uživatelský přehled je karta s tlačítkem na změnu hesla.

**Jak obnovit heslo?** Přejděte na přihlašovací okno, pod formulářem je odkaz na změnu hesla. Vyplňte email, kterým jste registrován, a klikněte na tlačítko OK. Aplikace zašle email s odkazem pro obnovení hesla.

**Jak upravit nebo smazat vložený inzerát?** V levém menu klikněte na Správa inzerátů →Přehled mých inzerátů. V seznamu najděte inzerát, se kterým chcete pracovat, a klikněte na tlačítko Podrobnosti. V pravém horním rohu je možnost jeho editace, nebo smazání.

## 6.3 Neprofesionální prodejce

**Jak si vytvořit účet?** Po vstupu do aplikace na stránce https://foldynatulbp.cloud/ klikněte v pravém horním rohu na možnost Přihlásit se. Pod formulářem je uvedený odkaz na vytvoření účtu (obrázek 6.2).




Obrázek 6.2: Vytvoření účtu

**Jak upravit svůj účet?** V sekci Uživatelský přehled v levém menu je na kartě osobních údajů tlačítko Upravit.

**Proč vložený inzerát nelze dohledat?** Vložený inzerát soukromého prodejce zveřejňuje administrátor. Pokud ještě k tomuto zveřejnění nedošlo, inzerát se ve vyhledávání nezobrazí.

## 6.4 Profesionální prodejce

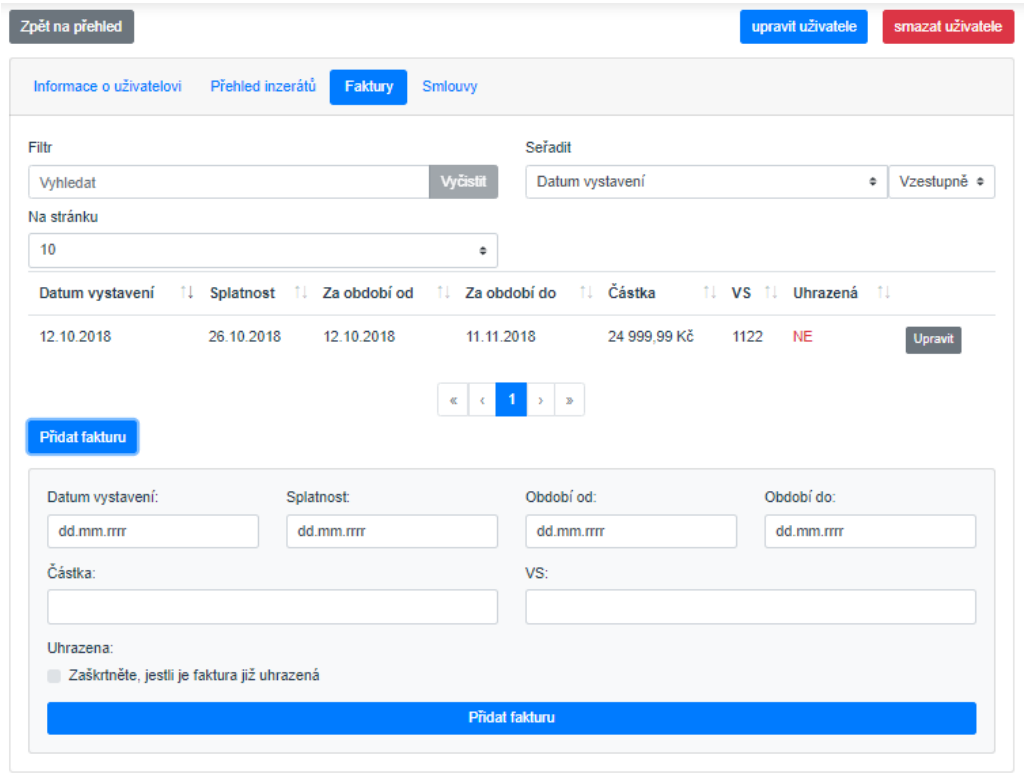
**Jak si vytvořím účet?** Účet vytváří administrátor. Profesionální prodejce nemůže sám sebe přidat, ani se upravovat.

**Kde naleznu přehled faktur a smluv?** Kliknutím na Můj přehled v levém menu a vybráním požadovaného seznamu.

## 6.5 Administrátor

**Jak přidám uživatele?** V sekci Správa uživatelů →Přidat uživatele lze vytvořit účet profesionálního prodejce a administrátora.

**Jak spravuji účet uživatele?** Kliknutím na sekci Správa uživatelů →Seznam uživatelů se zobrazí seznam všech uživatelů. Po kliknutí na Podrobnosti můžete upravit, nebo smazat kteréhokoliv uživatele (nelze smazat účet, pod kterým je administrátor přihlášený). U podrobností profesionálního uživatele se spravují i jeho faktury a smlouvy (obrázek 6.3).




Obrázek 6.3: Správa uživatelů

**Jak spravuji značky, modely, výbavy a všeobecné podmínky?** V sekci Správa databáze kliknutím na požadovanou tabulku.

# 7 Srovnání vytvořené aplikace

V této části byly porovnány již existující inzertní servery s vytvořenou aplikací této práce. Porovnání bylo provedeno se třemi vybranými servery uvedenými v kapitole 2.1, a to obdobným způsobem jako předchozí srovnání. Měření bylo uskutečněno z důvodu jiných podmínek internetového připojení u všech serverů znovu, aby byly výsledky co nejméně zkresleny.

Výsledky měření lze vidět v tabulce 7.1. Měřené veličiny času načítání a velikosti stažené do webového prohlížeče mohou být nepřesné z důvodu malého počtu inzerátů v databázi vytvořené aplikace.

Počet požadavků vzhledem k použití architektury REST API zůstane neměnný pro libovolné množství inzerátů a je vidět, že používá nejméně dotazů na server ze všech zkoumaných serverů.

Rovněž z rychlosti překreslování DOM vytvořené aplikace je možné usoudit, že využití aplikačního rámce VueJS výrazně zrychlí překreslování dokumentárního objektu. Je to způsobeno tím, že VueJS využívá virtuální prostředí pro DOM, který se porovnává s novým požadavkem na překreslení a překresluje pouze změněné prvky, nikoliv celý objekt.

Tabulka 7.1: Výsledky druhého měření pomocí prohlížeč Chrome

| Server | Načítání (s) | Velikost (MB) | Počet požadavků | DOM (s) |
|---|---|---|---|---|
| **Tipcars** | 11.30 | 5 | 224 | 5.44 |
| **Sauto.cz** | 5.07 | 2.3 | 113 | 1.68 |
| **Auto.de** | 4.12 | 4.3 | 38 | 2.02 |
| **foldynatulbp** | 0.84 | 2 | 19 | 0.62 |

# 8 Možnosti dalšího řešení

V této kapitole jsou uvedena doporučení pro další pokračování vývoje aplikace. Jedná se o zlepšení kvality kódu a možnosti rozšiřující funkcionalitu předmětné aplikace.

## 8.1 Zlepšení kvality kódu

Zdrojové kódy koncové aplikace obsahují několik již objevených nedostatků, které nebylo možné z časových důvodů již odstranit. Proto jsou tyto nedostatky níže uvedeny jako body, které by se při dalším vývoji měly řešit.

### 8.1.1 API

Pro zpřehlednění a lepší udržitelnost kódu by bylo dobré funkce jednotlivých směrovačů nedefinovat přímo u konkrétního směrovače, ale vytvořit objekt, ve kterém by byly tyto funkce definované, a v jednotlivých směrovačích je pouze volat.

Pro zlepšení bezpečnosti by se do JWT mohl přidat požadavek na IP adresu klientského počítače, ze kterého byla žádost odeslána. Nynější zabezpečení pouze pomocí expirační doby nijak nezabezpečuje situaci, kdy někdo odchytí token. V případě odchycení tokenu za současného stavu by mohl útočník po dobu platnosti tokenu využívat všechny směrovače skupiny, ke kterým by se token vázal.

S problémem se v současném stavu potýká i odesílání emailu pro zapomenuté heslo. Někteří emailoví klienti zprávy označují jako spam (např. gmail), jiní je vůbec uživateli nedoručí (např. Apple). Pro budoucí pokračování je nezbytně nutné tento problém vyřešit.

### 8.1.2 Klientská aplikace

V administrační části se často používá tabulka s možností řazení a vyhledávání (např. Seznamy uživatelů, značek, modelů atd.). Ve všech těchto komponentách se kód zbytečně opakuje a bylo by vhodné vytvořit jednu komponentu pro seznam s možností vyhledávání a prostřednictvím vstupních parametrů definovat sloupce a předávat data. Vytvořenou komponentu poté pouze využívat v ostatních komponentách.

Další úpravou v administrační části by měla projít komponenta levého menu. V aplikaci se nemá zobrazovat na stránce pro přihlášení, proto nebyla importována v main.js, ale je součástí všech ostatních komponent, což zapříčiňuje menší přehlednost v jejich sekci template. Vhodnější varianta je naprogramovat komponentu navbarLeft tak, aby rozpoznala směrovací adresu přihlašovací stránky a v takovém případě se nezobrazovala. Poté by mohla být importována v main.js.

Ve všech komponentách administrační části, které jsou napojeny na směrovače, je využíván alarm pro zobrazování hlášení uživateli. Alarmy jsou zbytečně definované v každé komponentě zvlášť a bylo by lepší vytvořit pouze jeden globální alarm v main.js, který by komponenty pouze využívaly.

## 8.2 Zautomatizování údržby DB

Databáze je navržena tak, aby některé položky mohly být smazány na základě časové prodlevy. Jedná se například o dlouho nepřihlášeného soukromého prodejce, nebo jeho nezveřejněné inzeráty (expirace smazání). V současném stavu může jak inzeráty, tak i uživatele mazat pouze administrátor v administračním prostředí. Na server mohou být však napsané skripty, které se budou automaticky spouštět v nějakých časových intervalech a mazat staré položky, případně informovat uživatele o blížících se termínech expirace.

## 8.3 Přidání funkcionalit

Velký počet funkcionalit není příliš žádoucí, protože poté se může stát, že aplikace ztratí svou přehlednost a uživatelskou přívětivost. Proto by se neměly přidávat všechny funkcionality již existujících serverů, ale je potřeba zaměřit se na něco nového a uživatelsky žádaného.

Jednou z takových funkcionalit by mohla být neuronová síť, která na základě údajů v inzerátu najde různě dostupné odkazy na testy k danému vozidlu. Další neuronová síť by mohla na základě údajů o vozidle a srovnání podobných již existujících inzercí predikovat prodejci prodejní cenu (nacenění vozidla).

# 9 Závěr

V této práci byla úspěšně naprogramována a implementována plně funkční webová aplikace na prodej automobilů, která splňuje všechny body zadání, včetně architektury REST API. Tato aplikace umožňuje soukromým a profesionálním prodejcům vkládat a upravovat inzeráty, kupujícímu uživateli v nich vyhledávat podle rozsáhlých kritérií a administrátorovi umožňuje spravovat aplikaci jak v oblasti uživatelů, tak i v oblasti správy aplikační databáze. Předmětná aplikace je zveřejněna na adrese https://foldynatulbp.cloud.

Na základě poznatků z odborných zdrojů byl pro tvorbu databáze předmětné aplikace zvolen open source SŘBD PostgreSQL, neboť dle odborných názorů patří PostgreSQL spolu s MySQL k nejlepším open source SŘBD pro správu databází. Z dostupných srovnání rovněž vyplynulo, že v současné době se oba uvedené open source SŘBD vyrovnají komerčně nabízeným produktům. Použití SŘBD PostgreSQL pro správu databáze bylo upřednostněno před SŘBD MySQL zejména z důvodu větší výkonnosti SŘBD PostgreSQL, neboť tvořená aplikace předpokládá práci s velkým množstvím dat v databázi.

Pro naprogramování Application Programning Interface (API) byl zvolen aplikační rámec Slim od Slim Framwork Team, protože tento aplikační rámec se pro tvorbu API často využívá a je tak již praxí ověřený. Slim byl zvolen také pro intuitivnost jeho použití, snadné směřování URI adres funkcí a zejména jeho funkcionalitu Middleware, která umožňuje pracovat s požadavkem před příchodem do cílové funkce a následně i pracovat s odchozí odpovědí. Pro zabezpečení API byla zvolena technologie JWT, neboť jde o produkt ověřené autorizační společnosti Autho.

Pro tvorbu klientské části předmětné aplikace byl na základě poznatků z odborných zdrojů zvolen aplikační rámec VueJS, který byl v srovnávacím testu vyhodnocen jako jeden z nejlepších a který umožňuje reaktivitu a zachovává si přehlednost kódů.

Při navrhování webové aplikace pro prodej automobilů byl brán zřetel především na to, aby byla tato aplikace dobře zabezpečená, stabilní a rychlá i při práci s rozsáhlou databázovou strukturou. Stejnou mírou bylo zohledněno i to, aby byla aplikace uživatelsky maximálně přívětivá, jak pro administrátora (správce aplikace), tak i pro její další uživatele, zejména pro ty, kteří jejím prostřednictvím nabízejí k prodeji automobil, nebo mají zájem o koupi automobilu.

Za účelem kvalitního zabezpečení tvořené aplikace byly při jejím navrhová-

ní pro programování API jednotlivé směrovací adresy rozděleny do skupin dle stupně nastaveného přístupového oprávnění. Při navrhování aplikace pro prodej automobilů bylo nutné vycházet z toho, že taková aplikace bude vždy pracovat s velkým množstvím různých dat a že tak bude vždy vyžadovat rozsáhlou a vnitřně členitou databázovou strukturu. Tato struktura proto musela být navržena tak, aby umožňovala co nejefektivnější práci s daty a současně byla optimalizována pro daný účel, tedy nebyla zbytečně komplikovaná a nezpomalovala či neohrožovala stabilitu aplikace. Zároveň bylo třeba zajistit, aby se v databázi neopakovala stejná data tam, kde to není účelné.

Pro dosažení uživatelské přívětivosti aplikace bylo pro administrátora i pro prodávajícího uživatele navrženo jedno přehledné administrativní prostředí pro vkládání a správu dat a pro zájemce o koupi automobilů bylo navrženo prostředí umožňující snadné a rychlé vyhledávání vozidel dle zadaného požadavku. Vyhledávání bylo navrženo tak, aby se při zadání značky zobrazily jen modely této značky, při zadání modelu se zobrazily jen roky výroby, kdy se takový model skutečně vyráběl, a při zadání modelu se zobrazila pouze výbava dostupná pro vybraný model.

Dle vypracovaného návrhu a za použití zvolených technologií, tedy při tvorbě databáze za využití SŘBD PostgreSQL, při programování API v aplikačním rámci Slim, při zabezpečení API technologií JWT a při tvorbě klientské aplikace v aplikačním rámci VueJS, byla vytvořena webová aplikace pro prodej automobilů. Tato aplikace obsahuje mimo jiné 61 směrovacích adres API, 29 komponent, ze kterých se skládá administrativní část, a 5 komponent, které tvoří vyhledávací část.

Pro účely vlastní implementace této webové aplikace pro prodej automobilů musel být nakonfigurován server, aby na něj mohla být aplikace nainstalována. Za tím účelem byly nakonfigurovány virtuální hosty Apache pro jednotlivé části aplikace, přičemž každý virtuální host byl zabezpečen ověřeným šifrovacím protokolem Hypertext Transfer Protocol Secure (HTTPS). Konfigurace serveru zahrnovala i instalaci a konfiguraci databázového systému PostgreSQL.

Výsledkem bylo vytvoření plně funkční webové aplikace pro prodej automobilů, která je přístupná na adrese https://foldynatulbp.cloud. Tato aplikace byla následně porovnána s jinými inzertními servery pro prodej automobilů. Rychlost načítání stránky nemohla být hodnocena, neboť vytvořená aplikace nemá databázi naplněnou daty, ale vytvořená aplikace dosáhla výrazně rychlejšího překreslování DOM, které je nezávislé na velikosti databáze, než srovnávané inzertní servery. Vytvořená aplikace využívá oproti těmto serverům rovněž méně dotazů na server pro získání potřebných dat (díky požití REST API).

Vytvořená webová aplikace pro prodej automobilů tak potvrdila vhodnost výběru a použití zvolených moderních technologií. Plná funkčnost této aplikace prokázala i správnost návrhu, vlastní tvorby a implementace vytvořené aplikace.

# Seznam tabulek

# Seznam obrázků

# Literatura


[1] GRINBERG, Miguel. Flask web development: [developing web applications with Python]. Sebastopol: O'Reilly, 2014. ISBN 978-1-449-37262-0.

[2] TYRYCHTR, Jan. Provozní a analytické databáze - Teoretické základy [e-kniha]. Praha: ČSVIZ, 2015 [vid. 2015-03-17]. ISBN 978-80-87968-02-4. Dostupné z: http://www.csviz.cz/nakladatelstvi/e-knihy/.

[3] CONRAD, Tim. CONRAD, Tim. PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need [online]. 2004 [cit. 2019-03-24]. Dostupné z: http://www.devx.com/dbzone/Article/20743

[4] Understanding REST [online]. [cit. 2019-04-02]. Dostupné z: https://spring.io/understanding/REST

[5] CONRAD, Tim. LOCKHART, Josh, Andrew SMITH a Rob ALLEN. Slim a micro framework for PHP [online]. [cit. 2019-03-30]. Dostupné z: http://www.slimframework.com/

[6] BHATIA, Sagar. PostgreSQL vs. MySQL: Everything You Need to Know [online]. 2019 [cit. 2019-04-02]. Dostupné z: https://hackr.io/blog/postgresql-vs-mysql

[7] Introduction to JSON Web Tokens [online]. [cit. 2019-04-03]. Dostupné z: https://jwt.io/introduction/

[8] BHAGAT, Varun. Best Web Development Frameworks Comparison [online]. 20.6.2017 [cit. 2019-04-03]. Dostupné z: https://www.pixelcrayons.com/blog/web/best-web-development-frameworks-comparison/

[9] SCHAE, Jacek. A Real-World Comparison of Front-End Frameworks with Benchmarks [online]. [cit. 2019-04-03]. Dostupné z: https://medium.freecodecamp.org/a-real-world-comparison-of-front-end-frameworks-with-benchmarks-2018-update-e5760fb4a962

[10] PHP vs Python vs Ruby: The Battle of Web Programming Languages [online]. [cit. 2019-04-03]. Dostupné z: https://codecondo.com/the-battle-of-web-programming-languages/

[11] NJENGA, Alice. 10 Popular PHP frameworks in 2019 [online]. 21.11.2018 [cit. 2019-04-03]. Dostupné z: https://raygun.com/blog/top-php-frameworks/

[12] Vue.js: The Progressive JavaScript Framework [online]. [cit. 2019-04-04]. Dostupné z: https://vuejs.org/

[13] Vue Image Upload and Resize [online]. [cit. 2019-04-08]. Dostupné z: https://github.com/kartoteket/vue-image-upload-resize

[14] PSR-7 and PSR-15 JWT Authentication Middleware [online]. [cit. 2019-04-08]. Dostupné z: https://github.com/tuupola/slim-jwt-auth

[15] The HTTP client for Vue.js [online]. [cit. 2019-04-15]. Dostupné z: https://github.com/pagekit/vue-resource

# Seznam příloh

- Zdrojové kódy (elektronická verze)
  - API (bez zdrojových kódů Slim)
  - Databáze
  - Klientská aplikace
    * Administrační část (bez zdrojových kódů VueJS)
    * Vyhledávací část (bez zdrojových kódů VueJS)