UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE

FAKULTA PRÍRODNÝCH VIED

Katedra informatiky

VÝVOJ WEBOVEJ APLIKÁCIE S VYUŽITÍM MODERNÝCH WEBOVÝCH TECHNOLÓGIÍ

BAKALÁRSKA PRÁCA

Študijný odbor:	9.2.9 Aplikovaná informatika
Študijný program:	Aplikovaná informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Mgr. Martin Drlík, PhD.

Nitra 2018 Miroslav Grofčík

Univerzita Konštantína Filozofa v Nitre
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Miroslav Grofčík
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9 aplikovaná informatika
Typ záverečnej práce: Bakalárska práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vývoj webovej aplikácie s využitím moderných webových technológií

Anotácia: Súčasné webové aplikácie možno vytvárať s využitím frontendových ako aj backendových frameworkov. V prípade frontendových aplikácií je biznis logika webovej aplikácie presunutá do frontendu, pričom úlohou backendu je poskytovanie jednoduchých API. Do popredia sa presadzujú JavaScriptové frameworky Angular, Vue, React alebo Node.js. Veľmi populárne však zostávajú klasické backendové frameworky pre generovanie dynamických HTML stránok ako sú napríklad Laravel, Symfony a Zend.
Cieľom práce je charakterizovať súčasné trendy vývoja webových aplikácií s využitím frontentových a backendových frameworkov. Študent na vývoji konkrétnej aplikácie prezentuje, ako možno vhodne využiť silné stránky moderných vývojových frameworkov.
Predpokladané znalosti: programovanie webových aplikácií, tvorba GUI webových aplikácií, JavaScript, PHP

Školiteľ: Mgr. Martin Drlík, PhD.
Oponent: Mgr. Tomáš Tóth
Katedra: KI - Katedra informatiky

Dátum zadania: 28.09.2016

Dátum schválenia: 10.10.2016 prof. Ing. Milan Turčáni, CSc.
schválil/a

ABSTRAKT

Miroslav Grofčík: Vývoj webovej aplikácie s využitím moderných webových technológií. [Bakalárska práca]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied; Katedra informatiky. Školiteľ: Mgr. Martin Drlík, PhD. 36 strán.

Cieľom bakalárskej práce je vývoj webovej aplikácie za pomoci moderných technológií, určených pre vývoj webových aplikácií. Teoretická časť tejto práce približuje bližšie problematiku vývoja webových aplikácií a opisuje niektoré zo súčasných technológií pre vývoj webových aplikácií. V praktickej časti práce je podrobne opísaný vývoj webovej aplikácie s použitím konkrétnych technológií, takisto je v nej popísané ako táto webová aplikácia funguje. Výsledkom tejto práce je analýza problematiky vývoja webových aplikácií a vytvorená webová aplikácia z viacerých pohľadov.

ABSTRACT

Miroslav Grofčík: Web application development using modern web technologies. [Bachelor thesis]. Constantine the Philosopher University in Nitra. Faculty of natural sciences; Department of Informatics. Supervisor: Mgr. Martin Drlík, PhD. 36 pages.

The purpose of this bachelor thesis is to develop a web application by using modern technologies designed for web development. The theoretical part of the thesis is focused on web development and describes some of the modern technologies used for web development. The practical part of the thesis describes development of the web application using specific technologies and it describes how the web application works. The result of this work is an analysis of web application development and a created web application from multiple perspectives.

OBSAH

ÚVOD

Obsahom tejto práce bude analýza súčasného stavu vývoja webových aplikácií a moderných technológií používaných pri vývoji týchto aplikácií. V praktickej časti bude opísaný vývoj konkrétnej webovej aplikácie, ktorá bude prílohou tejto práce. Takisto v praktickej časti bude popísané ako daná aplikácie funguje.

Na začiatku práce priblížime problematiku vývoja webových aplikácií a vysvetlíme si základné pojmy a technológie, ktoré sa týkajú vývoja webových aplikácií. Potom prejdeme k samotnému opisu a analýze konkrétnych technológií, ktoré sa v dnešnej dobe používajú pre vývoj webových aplikácií.

V praktickej časti práce sa budeme venovať najprv opisu samotného vývoju aplikácie, a to opísaním jednotlivých štruktúr aplikácie. Vysvetlíme aké technológie a ich časti boli použité pre vývoj tejto webovej aplikácie. Ukážeme výhody použitia jednotlivých častí použitých technológií.

Týmto by som chcel ešte poďakovať svojmu školiteľovi pánovi Mgr. Martinovi Drlíkovi za trpezlivosť a pomoc pri vypracovaní tejto bakalárskej práce.

1 ANALÝZA SÚČASNÉHO STAVU WEBOVÝCH TECHNOLÓGIÍ

Webové technológie sa rok čo rok rýchlo vyvíjajú a už dlho nám pre vývoj webovej stránky alebo aplikácie, nestačia dva až tri jazyky alebo technológie, ale potrebujeme rovno celý arzenál. Projekty sa verziujú, automatizujú a stále viac sa tiež využívajú frameworky v podstate na všetko, keďže ohromne uľahčujú ďalší vývoj – alebo by aspoň mali.

Vyznať sa v súčasných trendoch a ovládať ich, alebo sa aspoň orientovať v danej technológii, je teda veľmi náročné. V súčasnej dobe sa taktiež stále viac vyvíjajú na báze open-source, kde desiatky, stovky, a dokonca aj tisíce vývojárov pracujú na jednom projekte a spoločne vytvárajú nádherné, a súčasne náročné, aplikácie či technológie. Na projektoch sa uplatnia vo veľkej miere výhody verziovacích systémov (Bittner Honza, 2015).

Vývoj webu prechádza neustálymi zmenami. V priebehu rokov si prešli webové aplikácie vývojom od statických webstránok až po komplexné systémy, ktoré sú vyvíjané pomocou najrôznejších back-end technológií. Webové aplikácie v dnešnej dobe smartfónov a tabletov si vyžadujú hlavne responzivitu, aby mohli byť korektne zobrazené na všetkých zariadeniach. Práve preto sa pri vývoji webu pojem responzivita stal akým-si nepísaným štandardom. Na nahrávanie vecí na server dnes už nestačí obyčajný FTP klient, ale používajú sa rôzne nástroje, ktoré urýchlia nahranie na server ako napríklad Git-FTP konzolový nástroj, ktorý predstavuje spojenie verziovacieho nástroja Git a FTP klienta.

V dnešnej dobe, kedy je potrebné ovládať nespočetné množstvo nástrojov a technológií pre vývoj webu rozdeľujeme vývojárov do troch skupín: front-end vývojári, back-end vývojári a full-stack vývojári. V nasledujúcich kapitolách si priblížime rozdiely medzi front-end, back-end a full-stack vývojom.

1.1 VÝVOJ WEBOVEJ APLIKÁCIE

Medzi činnosti súvisiace s vývojom webových aplikácií patrí web design, tvorba webového obsahu, skriptovanie na klientskej a serverovej strane, webový server, konfigurácia sieťovej bezpečnosti a vývoj e-commerce aplikácií (internetové obchody).

Vývoj webových aplikácií odkazuje na hlavné nedizajnové aspekty budovania webových stránok: značkovanie a programovanie. Vývoj webových aplikácií môže byť v rozsahu od rozvojových najjednoduchších statických stránok z prostého textu až po

najzložitejšie on-line internetové aplikácie, elektronické obchody alebo služby sociálnych sietí.

Navyše, vývoj webových aplikácií sa dostal do novej fázy internetovej komunikácie. Počítačové webové stránky už nie sú len nástroje pre prácu alebo obchod, ale používajú sa hlavne pre komunikáciu. Internetové stránky ako Facebook a Twitter poskytujú používateľom platformu pre voľnú komunikáciu. Táto nová forma webovej komunikácie sa taktiež mení na e-commerce prostredie prostredníctvom vysokého počtu zobrazení stránok, on-line reklamy a internetového marketingu.

Web vývoj môže byť rozdelený do niekoľkých oblastí a typické alebo základné rozdelenie hierarchie vývoja webových aplikácií sa môže skladať z:

- programovania na strane klienta: Ajax, JavaScript,...
- programovania na strane servera: PHP, Java, C#, .NET,...

Internet sa stal hlavnou platformou pre vývoj webových aplikácií, najrôznejších komplexných a zložitých podnikových systémov v niekoľkých oblastiach. Popri ich vlastnej bohatej funkčnosti predstavujú tieto webové aplikácie komplexné riešenia a kladú špecifické požiadavky na ich využiteľnosť, výkon, bezpečnosť, schopnosť rásť a vyvíjať sa.

Avšak, drvivá väčšina vývojárov alebo živnostníkov aj naďalej vyvíja aplikácie ad-hoc spôsobom a prispievajú tak k problémom s použiteľnosťou, udržateľnosťou, výkonom, bezpečnosťou, kvalitou a spoľahlivosťou (Quantasoft.sk, 2017).

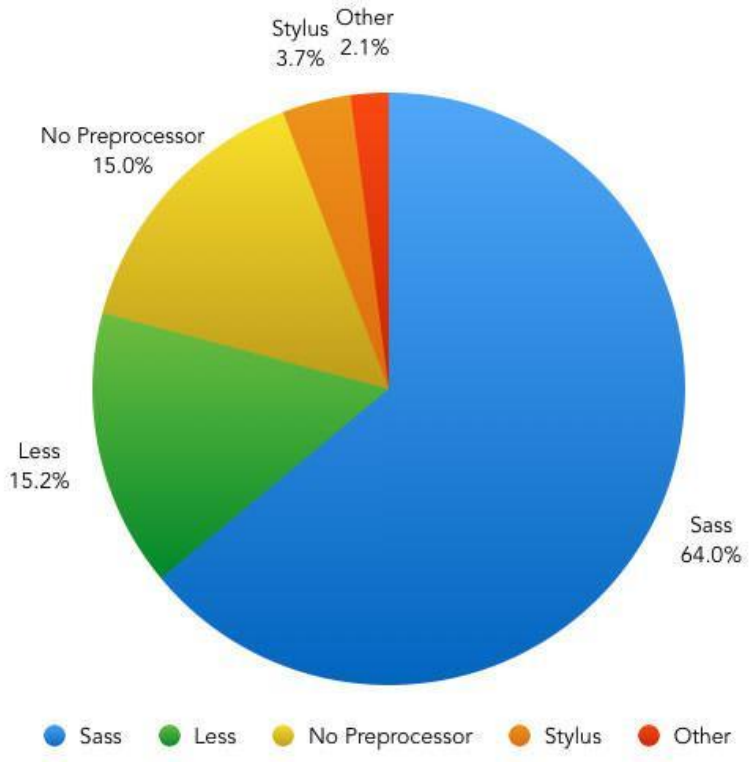
1.2 FRONT-END VÝVOJ

Front-end vývoj alebo webový vývoj na strane klienta predstavuje dve činnosti návrhu webovej aplikácie: návrh usporiadania elementov vo webovej aplikácií a programovanie interaktívnych prvkov webovej aplikácie. Z historického hľadiska sa na vývoji front-endovej časti webových aplikácií obsadzovali pozície ako HTML (hyper text markup language) vývojár, web dizajnér, frontend vývojár atď., ale vo svojej podstate vykonávali rovnaké úkony pri vývoji webových aplikácií a to návrh a interaktivita user-interface (UI), zatiaľ čo sa museli prispôsobovať stále novým technológiám. Pre zastávanie pozície front-endového vývojára webových aplikácií je potrebné ako estetické cítenie tak aj programátorské myslenie a zručnosti.

Webový vývoj na strane klienta môžeme nazvať aj ako 'robenie vecí peknými', pretože vývin UI znamená navrhovanie peknej a prehľadnej webovej aplikácie, tak je tu aj mnoho ďalších technológií, ktoré sa môžu zdať bežnému používateľovi prehliadnuteľné.

Prehľad niektorých z nich:

1. **Značkovanie (HTML)**: Štruktúra stránky a označenie príslušných elementov je základom každej webovej stránky alebo aplikácie. Značkovanie elementov stránky slúži k ich usporiadaniu, priradeniu tried alebo identifikátorov pomocou ktorých vieme definovať štýl, výzor elementov a takisto programovateľnosť jednotlivých elementov.
2. **Štýly (CSS)**: Technológia CSS (cascade style sheets) alebo inak povedané technológia kaskádových štýlov je využívaná k priradeniu určitých vlastností elementom webovej aplikácie. Slúži k vytvoreniu príjemného vizuálneho prostredia, ktoré je používateľsky-prívetivé (user-friendly). V zásade to znamená, že obyčajný používateľ dokáže jednoducho a intuitívne rozoznať a používať webové rozhranie.



Obrázok 1 - Koláčový graf najpoužívanejších CSS preprocesorov [1]

V obrázku 1 je vidieť graf, ktorý nám zobrazuje momentálne najpopulárnejšie CSS preprocesory, používané vývojármi na celom svete.

3. **CMS a šablóny, webové frameworky**: Predstavujú technický základ dynamicky generovaných webových stránok ako súčasť MVC alebo MVT alebo inej

[1] zdroj: https://ashleynolan.co.uk/blog/frontend-tooling-survey-2015-results

softvérovej architektúry.

4. **Programovanie**: Pod programovaním na front-ende chápeme hlavne programovanie v programovacom jazyku JavaScript. JavaScript sa úplne rozrástol z inline príkazov zabudovaných do HTML na plnohodnotné asynchrónne aplikácie. V dnešnej dobe je rozšírené používanie rôznych JavaScriptových knižníc ako je napríklad jQuery. Takisto, medzi front-endovými vývojármi sú veľmi populárne JavaScriptové frameworky ako napríklad AngularJS alebo VueJS, ktoré veľmi uľahčujú vývin front-endu webových aplikácií. Používanie týchto knižníc a frameworkov prinieslo množstvo vizuálnych efektov, ktoré premieňajú naše webové stránky na trojrozmerný zážitok. Programovací jazyk JavaScript bol vyvinutý špeciálne pre web a programovanie webových stránok a aplikácií (Codesido Ivan, 2009).

1.3 BACK-END VÝVOJ

Back-end vývoj alebo webový vývoj na strane servera je pojem, ktorý predstavuje to ako webová aplikácia funguje, teda sa jedná o termín vyjadrenia celej logiky webovej aplikácie. Pod back-end vývojom chápeme celú funkcionalitu a logiku webu, ktorú bežný používateľ nevidí voľným okom, ale ktorá prebieha na pozadí webovej aplikácie a je jej neoddeliteľnou súčasťou.

Osoby vyvíjajúce back-end - funkcionalitu webovej aplikácie sú označované ako programátori alebo back-endoví vývojári. Zaujímajú sa a riešia problémy týkajúce sa funkcionality, bezpečnosti, štruktúry a obsahového manažmentu webovej aplikácie. Vytvárajú dynamickosť webových stránok.

Dynamická webová stránka predstavuje stránku, ktorá sa mení v reálnom čase podľa vstupov zadaných od používateľa. Dynamická webová stránka alebo aplikácia pracuje vždy s databázou. Do databázy sú ukladané všetky dôležité údaje ako mená a heslá používateľov, obsahové časti webu,...

Back-endoví weboví programátori pracujú s programovacími jazykmi PHP alebo .NET, ale aj inými. Kód, ktorý je napísaný, komunikuje so serverom a následne určí webovému prehliadaču, s ktorými dátami z databázy má pracovať, resp. ktoré dáta majú byť zobrané z databázy a následne zobrazené používateľovi (Pluralsight.com, 2015).

1.4 TECHNOLÓGIE PRE VÝVOJ WEBU

Pre štruktúrovanie a programovanie webových aplikácií sa v dnešnej dobe využívajú mnohé webové technológie, ktoré vznikli za účelom vývoja týchto aplikácií. Medzi tieto technológie patria HTML a CSS pre štruktúrovanie webu a úpravu štýlov a grafického vizuálu používateľského prostredia na webe.

HTML a CSS sú technológie určené k značkovaniu jednotlivých elementov webovej aplikácie a k vytvoreniu grafického vizuálu týchto prvkov. CSS v kombinácii s programovacím jazykom javascript alebo rôznymi javascriptovými frameworkami dokážu vytvárať animácie a pohyblivé časti používateľského prostredia (UI).

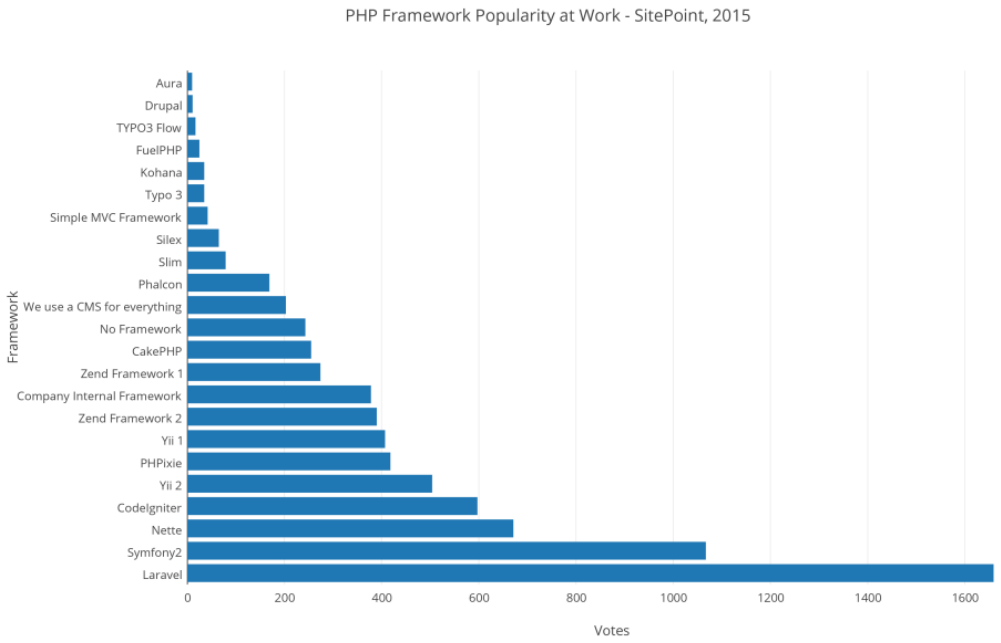
Pri vytváraní UI (user interface) sa z programovacích jazykov používa programovací jazyk javascript. V dnešnej dobe však prácu uľahčujú technológie, ktoré programátori poznajú pod pojmom frameworky. Frameworky, konkrétne javascriptové frameworky, sú knižnice s dopredu naprogramovanými metódami a funkciami, ktoré front-endovým vývojárom uľahčujú prácu a tvorbu používateľského rozhrania webových aplikácií.

Medzi najčastejšie jazyky používané back-endovými vývojármi momentálne patria PHP a JavaScript. Najviac používaným jazykom je jazyk PHP, pretože ho podporuje väčšina webových serverov a v dnešnej dobe predstavuje akýsi pomyselný štandard pre spoločnosti, ktoré obchodujú s priestorom na webe (W3schools.com, 2017).

1.5 LARAVEL, PHP FRAMEWORK

Laravel, PHP framework, je momentálne veľmi populárna technológia medzi back-endovými webovými vývojármi. Tento framework je postavený na programovacom jazyku PHP.

Laravel sa vyznačuje expresívnou, prehľadnou a peknou syntaxou. Nezáleží na tom, či na vývoji webovej aplikácie pracuje programátor sám, alebo tím programátorov tvorí 20 osôb, pretože tento framework je vyvinutý veľmi precízne. Vďaka zabudovaným technológiám, migrácii databázových tabuliek a celkovej schéme frameworku je zabezpečená synchronizácia projektu pri vývoji vo väčšom tíme programátorov.



Obrázok 2 - Graf najpopulárnejších PHP frameworkov [2]

Ako je vidieť v obrázku 2, Laravel je momentálne najpoužívanejším PHP frameworkom tejto doby.

Laravel usiluje o to, aby celá skúsenosť s vývojom v PHP bola príjemná, vrátane prostredia pre lokálny vývoj aplikácie. Toto prostredie sa nazýva Homestead. Laravel Homestead je oficiálny Vagrant box, ktorý nám poskytuje skvelé vývojové prostredie bez nutnosti inštalovať PHP, webový server a akýkoľvek iný serverový softvér na našom lokálnom počítači. Odbremeňuje nás od starostí s narušovaním nášho operačného systému.

Laravel Homestead beží na ľubovoľnom operačnom systéme Windows, Mac alebo Linux a obsahuje webový server Nginx, PHP 7.1, MySQL, Postgres, Redis, Memcached, Node, a všetky ostatné potrebné veci na vytváranie profesionálnych webových aplikácií (Laravel.com, 2017).

Súčasťou frameworku Laravel je aj Eloquent Model. Eloquent Model umožňuje neuveriteľne jednoduché interakcie s databázou. Veľmi jednoducho spracúva funkcie CRUD (create, read, update, delete). Poskytuje nám implementáciu ActiveRecord pre prácu s našou databázou. To znamená, že každý model, ktorý vytvoríme v našej MVC štruktúre, zodpovedá tabuľke v našej databáze. V praxi, to znamená, že ak vytvoríme model User, tak automaticky bude zodpovedať tabuľke users v našej databáze. Tabuľku, na ktorú sa model viaže, je možné aj upraviť pomocou prepísania premennej v modeli.

[2] zdroj: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

Keďže máme konvenciu pri vytváraní našich modelov a databázových tabuliek, môžeme jednoducho zavolať dáta z našej databázy. Napríklad, ak chceme získať všetky dáta z tabuľky users, stačí v modeli User zavolať metódu all nasledovným zápisom User::all();. Použitím takejto funkcie vygenerujeme správny SQL príkaz a budeme sa ním dotazovať na databázu, ktorá nám vráti požadované dáta (Sevilleja Chris, 2014).

1.6 ANGULAR, JAVASCRIPT FRAMEWORK

Angular je javascriptový framework. Aktuálne je to veľmi populárny front-endový framework používaný front-endovými vývojármi pri vývoji webových aplikácií.

Umožňuje nám používať jazyk HTML ako jazyk webovej šablóny, a zároveň nám umožňuje rozšíriť syntax HTML tak, aby jasne a stručne vyjadril komponenty našej webovej aplikácie.

Väzba údajov vo frameworku Angular eliminuje väčšinu kódu, ktorý by sme inak museli napísať. Toto všetko sa deje v prehliadači, takže je ideálny pre vývoj s akoukoľvek serverovou technológiou.

Impedančný nesúlad medzi dynamickými aplikáciami a statickými dokumentmi sa často rieši pomocou knižníc a frameworku (Angular.io, 2017).

Knižnica predstavuje zbierku funkcií, ktoré sú užitočné pri písaní webových aplikácií. Kód Angularu volá do určitej knižnice, iba keď to považuje za vhodné (napr. knižnica - jQuery).

Framework funguje takisto veľmi systematicky a odkazuje na časti kódu, keď potrebuje niečo konkrétne pre aplikáciu.

Angular má ďalší prístup. Pokúša sa minimalizovať impedančný nesúlad medzi dokumentom zameraným na HTML a tým, čo potrebuje naša webová aplikácia vytvorením nových konštruktorov HTML. Angular učí prehliadač novú syntax, ktorú nazývame direktívy (Angular.io, 2017).

1.7 JQUERY, JAVASCRIPT FRAMEWORK

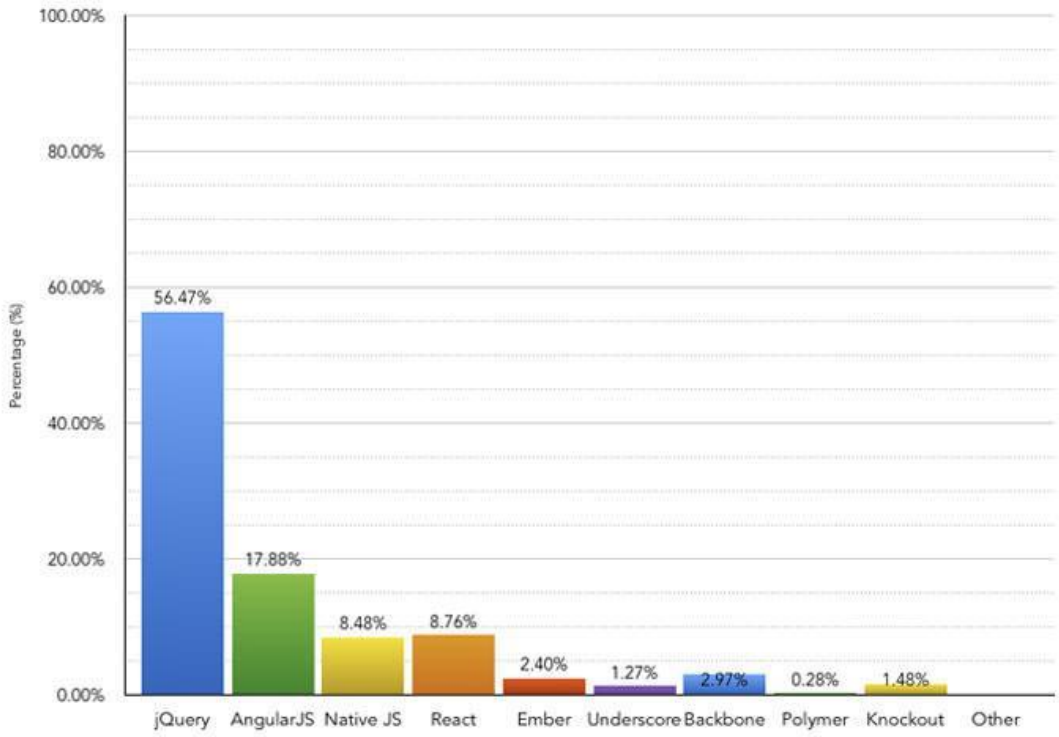
JQuery je rýchla, malá javascriptová knižnica, ktorá je bohatá na funkcie. Uľahčuje nám prácu s programovaním front-endu webovej aplikácie. Konkrétne uľahčuje programovanie v JavaScript-e.

JQuery nám dokáže ušetriť množstvo kódu. Vďaka kombinácii všestrannosti a rozšíriteľnosti JQuery spôsobilo zmenu, akým milióny vývojárov píšu skripty v programovacom jazyku JavaScript.

JQuery kladie dôraz na interakciu medzi JavaScriptom a HTML. Bola vydaná v roku 2006. Táto knižnica je veľmi populárna, čo dokazuje aj fakt, že je používaná vo viac ako 34% z milióna najnavštevovanejších webových stránok (Jquery.com, 2017).

JQuery je slobodný a otvorený softvér pod licenciou MIT (Massachusetts Institute of Technology). JQuery syntax je navrhnutá pre jednoduchšiu navigáciu dokumentu, výber DOM elementov, vytváranie animácií, spracovanie udalostí a vývoj Ajax aplikácií.

JQuery tiež poskytuje možnosti pre vývojárov na vytváranie pluginov postavených na tejto JavaScriptovej knižnici (Jquery.com, 2017).



Obrázok 3 - Graf najpopulárnejších JavaScript frameworkov [3]

Na obrázku 3 je jasne vidieť, že aj keď väčšina JavaScriptových frameworkov ponúka veľmi dobré možnosti pre vývoj frontendu webových aplikácií, najpoužívanejšia je práve jQuery knižnica, ktorá uľahčuje interakciu medzi HTML a JavaScriptom.

[3] zdroj: https://ashleynolan.co.uk/blog/frontend-tooling-survey-2015-results

2 CIELE BAKALÁRSKEJ PRÁCE

Cieľom bakalárskej práce je charakterizovať súčasné trendy vývoja webových aplikácií s využitím frontendových a backendových frameworkov. Ďalším cieľom je prezentovať na vývoji konkrétnej aplikácie, ako možno vhodne využiť silné stránky moderných vývojových frameworkov.

2.1 PODCIELE

- Charakterizovať súčasné trendy vývoja webových aplikácií,
- analyzovať použiteľnosť front-endových a back-endových frameworkov,
- vybrať vhodné front-endové a back-endové frameworky,
- na príklade konkrétnej webovej aplikácie prezentovať možnosti použitia front-endových a back-endových frameworkov,
- vyvinúť administrátorské prostredie konkrétnej webovej aplikácie,
- kriticky zhodnotiť prínos a slabé stránky riešenia.

3 TVORBA WEBOVEJ APLIKÁCIE

Rozhodli sme sa vytvoriť aplikáciu, ktorá bude fungovať ako rezervačný systém. Aplikácia pozostáva z dvoch rezervačných systémov, ktoré sú od seba navzájom nezávislé. Prvú časť aplikácie tvorí rezervačný systém áut. Druhú časť aplikácie tvorí rezervačný systém transferov v určitom meste.

Pred začatím tvorby webovej aplikácie sme stanovili štruktúru a dátový model. Stanovili sme, že webová aplikácia sa bude deliť na dve hlavné časti resp. dva rezervačné systémy pre klienta, ktoré sú od seba navzájom nezávislé. Takisto k webovej aplikácii patrí aj tretia časť určená pre administrátora alebo vlastníka webovej aplikácie v podobe administračného systému.

Webová aplikácia je vyvinutá pomocou týchto technológií:

- HTML – značkovací jazyk pre tvorbu webových aplikácií
- CSS – rozšírenie HTML na vizuálne formátovanie HTML dokumentov
- Sass – tzv. CSS preprocesor, pre jednoduchšiu prácu s CSS
- JavaScript – programovací jazyk na strane klienta
- jQuery – JavaScript knižnica na interakciu medzi JavaScriptom a HTML
- PHP – programovací jazyk na strane servera
- Laravel – framework jazyka PHP určený pre vývoj webových aplikácií

Jednu časť webovej aplikácie tvorí rezervačný systém pre požičanie vozidla resp. auta. V tejto časti si klient vie vybrať dané vozidlo podľa požadovaných vlastností a vytvoriť si rezerváciu resp. objednávku na dané vozidlo. Štrukturálne táto časť pozostáva z kategórií, produktov (áut) a objednávok.

Druhá časť pre klienta webovej aplikácie je zostavená ako rezervačný systém pre transfer v určitom meste z bodu A do bodu B, kde má transfer určenú jednotnú cenu pre dané mesto. Štrukturálne táto časť webovej aplikácie pozostáva z produktov (transferov) a objednávok (rezervácií).

Tretia časť webovej aplikácie je určená pre administrátora alebo vlastníka spoločnosti ako administračný systém. Administračný systém je vypracovaný jednoducho, prehľadne tak, aby sa v ňom vedel klient intuitívne orientovať bez potreby vysvetlenia alebo ďalšieho školenia.

3.1 ŠTRUKTÚRA WEBOVEJ APLIKÁCIE

Pri vývoji webovej aplikácie sme využili všetky možnosti, ktoré nám ponúkol Laravel – PHP framework. Vytvorili sme štruktúru databázy pomocou tzv. migrácií (Migrations), aby sa štruktúra databázy mohla vytvárať priamo z kódu aplikácie a aby nebolo potrebné neustále exportovať/importovať SQL skript. Migrácie majú veľké uplatnenie takisto aj pri vývoji webovej aplikácie vo viacčlennom tíme.

Pri tvorbe webovej aplikácie sme použili Laravel s jeho MVC (Model View Controller) architektúrou. MVC architektúra oddeľuje back-endový kód od HTML a tým uľahčuje vývoj a orientáciu v skriptoch. Model slúži na komunikáciu aplikácie s databázou a back-endovú logiku. View zobrazuje údaje pomocou HTML. Controller sa stará o prenos údajov medzi Models a Views.

Pre prácu s databázou používame Models, ktoré sú pomenované v jednotnom čísle podľa tabuliek v databáze, napr.: pre tabuľku ‚*orders*' existuje Model ‚*Order*'. Models pracujú s databázou pomocou systému, ktorý je zabudovaný v Laraveli a nesie názov Eloquent system. Eloquent systém umožňuje využívať CRUD operácie bez písania SQL skriptov.

Ďalšiu časť aplikácie tvoria Controllers a Views. Controllers slúžia hlavne na prenos dát medzi Models a Views. Takisto je v Controllers napísaná aj ďalšia logika aplikácie ako napríklad prerátavanie cien pri objednávkach.

Pre definovanie URL adries sme využili tzv. Routes, ktoré slúžia na zadefinovanie všetkých URL adries, ktoré bude webová aplikácia používať. Pokiaľ klient/používateľ zadá URL adresu, ktorá nie je zadefinovaná v Routes, teda neexistuje, bude presmerovaný na stránku s chybovou hláškou 404 – Stránka, ktorú hľadáte, neexistuje.

Takisto sme použili aj rôzne ďalšie súčasti Laravelu, ako sú napríklad Middlewares, Composers, Requests, Traits, Providers a ďalšie. Pomocou Middlewares sme spravili autentifikáciu používateľov aj administrátorov. Takisto pomocou Middleware je riešený prepínač jazykov aplikácie. Pomocou Requests, sme zabezpečili kompletnú validáciu aplikácie pri odosielaní formulárov a práci s databázou.

3.1.1 Migrácie

Pred začatím písania migrácií, sme si zadefinovali štruktúru databázy webovej aplikácie. Následne sme vytvorili Migrácie na databázu pre tabuľky *users, password_resets, categories, products, accessories, orders, order_accessories, transfers,*

reservations, *images* a *payments*. Každá tabuľka má svoje špecifické atribúty. Jednotlivé stĺpce v tabuľkách, ktoré sú zobrazované v aplikácii v klientskej časti, sú vytvorené s prefixom v tvare ‚*_skratka-jazyka*', teda napríklad pre názov záznamu v databáze sme vytvorili toľko stĺpcov s prefixom, koľko bude aplikácia používať jazykových mutácií. Príklad: *name_sk* a *name_en* pre jazykové mutácie slovenčiny a angličtiny.

Tabuľka *users* uchováva informácie o všetkých používateľoch aplikácie, teda aj o klientoch, aj o administrátoroch. V tabuľke *users* sú uložené informácie o mene, priezvisku, mieste bydliska, emaily, telefónnom čísle, heslo do aplikácie, informácia o tom či je používateľ administrátor alebo nie a ďalšie.

Tabuľky *categories*, *products*, *accessories*, *orders* a *order_accessories* spolu vytvárajú jeden celok pre jednu z častí klientskeho rozhrania, v ktorom je možné prenajať vozidlo. Tabuľka *categories* a tabuľka *products* ukladá informácie o autách a ich kategóriách. V tabuľke *accessories* sú uchovávané údaje o doplnkoch, ktoré je možné prenajať spolu s produktom, ako napríklad GPS systém alebo detské autosedačky. Tabuľka *orders* a *order_accessories* obsahuje kompletné informácie o objednávke, a to napríklad celkovú cenu objednávky, identifikátor produktu (auta), miesto vyzdvihnutia, miesto návratu, dátumy vyzdvihnutia a návratu, čas vyzdvihnutia a návratu, stav objednávky, informácie o objednaných doplnkoch k produktu, informácie o zákazníkovi a ďalšie.

```php
Schema::create('orders', function (Blueprint $table) {
    $table->increments('id');
    $table->string('order_number', 255)->nullable();
    $table->integer('user_id')->unsigned()->nullable();
    $table->foreign('user_id')->references('id')->on('users')->onDelete('set null')->onUpdate('set null');
    $table->integer('product_id')->unsigned()->nullable();
    $table->foreign('product_id')->references('id')->on('products')->onDelete('set null')->onUpdate('set null');
    $table->string('name')->nullable();
    $table->string('surname')->nullable();
    $table->text('address')->nullable();
    $table->string('street_number')->nullable();
    $table->string('postal_code')->nullable();
    $table->string('city')->nullable();
    $table->string('country')->nullable();
    $table->string('company_name')->nullable();
    $table->string('company_ico')->nullable();
    $table->string('company_dic')->nullable();
    $table->string('company_ic_dph')->nullable();
    $table->string('email', 255)->nullable();
    $table->string('phone')->nullable();
    $table->text('note')->nullable();
    $table->decimal('price')->nullable();
    $table->integer('payment_id')->unsigned()->nullable();
    $table->foreign('payment_id')->references('id')->on('payments')->onDelete('set null')->onUpdate('set null');
    $table->decimal('delivery_price')->nullable();
    $table->string('pick_address')->nullable();
    $table->string('pick_street_number')->nullable();
    $table->string('pick_postal_code')->nullable();
    $table->string('pick_city')->nullable();
    $table->string('pick_country')->nullable();
    $table->string('target_address')->nullable();
    $table->string('target_street_number')->nullable();
    $table->string('target_postal_code')->nullable();
    $table->string('target_city')->nullable();
    $table->string('target_country')->nullable();
    $table->date('pick_date')->nullable();
    $table->time('pick_time')->nullable();
    $table->date('target_date')->nullable();
```

Obrázok 4 - Snippet kódu Migrácie pre tabuľku orders

Na obrázku 4 je vidieť migráciu tabuľky *orders*. Tabuľky *transfers* a *reservations* takisto vytvárajú spolu jeden celok pre druhú časť klientskeho rozhrania aplikácie, v ktorom je možné objednať si transfer v danom meste z bodu A do bodu B. V tabuľke *transfers* sú uchovávané informácie a o transferoch v jednotlivých mestách ako napríklad cena transferu. Tabuľka *reservations* má obdobný význam tabuľky *orders*, teda uchováva informácie o objednávke jednotlivých transferov, informácie o zákazníkovi, počte pasažierov, počte batožín a ďalšie.

3.1.2 Models

Po vytvorení štruktúry databázy pomocou migrácií, sme vytvorili Model pre každú tabuľku v databáze. Pomocou Models vykonávame CRUD operácie vo webovej aplikácii, teda zobrazovanie, vytváranie, editovanie a mazanie jednotlivých záznamov v databáze. V nadväznosti na migrácie sme vytvorili Models: *User*, *Category*, *Product*, *Accessory*, *Order*, *OrderAccessory*, *Transfer*, *Reservation*, *Payment*, *Image*. Okrem týchto Models sme vytvorili Model s názvom *BaseModel*, ktorý dedia všetky vyššie spomenuté Models. V *BaseModel* sú definované funkcie, ktoré sú užitočné pre každý Model, ktorý pracuje s tabuľkami v našej databáze. Je tu definovaná funkcia *_translateProperty*, ktorá sa stará o vybratie stĺpca z databázy so správnou (aktuálnou) jazykovou mutáciou. Ďalej *BaseModel* obsahuje funkcie pre výpis formátovaných dátumov *created_at* a *updated_at*, ktoré obsahuje každá tabuľka v databázy našej webovej aplikácie.

```php
class BaseModel extends Model
{
    public function getFormattedCreatedAtAttribute(){
        return Carbon::parse($this->created_at)->format('d.m.Y H:i:s');
    }

    public function getFormattedUpdatedAtAttribute(){
        return Carbon::parse($this->updated_at)->format('d.m.Y H:i:s');
    }

    protected function _translateProperty($property, $returnNull = false){
        $translatedStr = $this->{$property . '_' . app()->getLocale()};

        $returnValue = $returnNull ? null : $this->{$property . '_sk'};

        return isset($translatedStr) && $translatedStr != "" ? $translatedStr : $returnValue;
    }
}
```

Obrázok 5 - Snippet kódu triedy BaseModel

V jednotlivých Models, ktoré pracujú s databázovými tabuľkami, sme určili tzv. ‚*fillable*', tieto predstavujú stĺpce v databáze, do ktorých sú zapísané informácie z Requests s rovnakým názvom kľúča ako je názov stĺpca, všetky ostatné informácie sú odignorované a do databázy zapísané nie sú. Táto časť Laravelu nám zabezpečuje jeden z dôležitých pilierov bezpečnosti celej webovej aplikácie.

3.1.2 Routes

Routes sme vytvárali priebežne počas vývoja aplikácie pre jednotlivé súčasti. Rozdelili sme ich na dve hlavné časti, a to Routes administrátorského prostredia, kde je potrebná autentifikácia administrátorov, v praxi to znamená, že sem nemá prístup obyčajný používateľ. Rozlíšenie medzi používateľom a administrátorom je vedené v databáze

v tabuľke *users* v stĺpci *is_admin* formou 0 a 1, kde 0 predstavuje hodnotu *false* a 1 predstavuje hodnotu *true*.

V Routes, ktoré sú určené pre admin sme sa snažili dodržať určitú koncepciu, aby sme boli schopní sa rýchlo orientovať v štruktúre Adminu aplikácie aj po dlhšej dobe.

```
// ADMIN Routes
Route::middleware(['auth', 'is_admin'])->namespace('Admin')->prefix('admin')->group(function(){

    // Dashboard
    Route::get('/', ['as' => 'dashboard.index', 'uses' => 'DashboardController@index']);

    // Categories
    Route::get('/categories', ['as' => 'categories.index', 'uses' => 'CategoriesController@index']);
    Route::get('/categories/create', ['as' => 'categories.create', 'uses' => 'CategoriesController@create']);
    Route::post('/categories', ['as' => 'categories.store', 'uses' => 'CategoriesController@store']);
    Route::get('/categories/edit/{id}', ['as' => 'categories.edit', 'uses' => 'CategoriesController@edit']);
    Route::post('/categories/{id}', ['as' => 'categories.update', 'uses' => 'CategoriesController@update']);
    Route::post('/categories/delete/{id}', ['as' => 'categories.delete', 'uses' => 'CategoriesController@delete']);

    // Products
    Route::get('/products', ['as' => 'products.index', 'uses' => 'ProductsController@index']);
    Route::get('/products/create', ['as' => 'products.create', 'uses' => 'ProductsController@create']);
    Route::post('/products', ['as' => 'products.store', 'uses' => 'ProductsController@store']);
    Route::get('/products/edit/{id}', ['as' => 'products.edit', 'uses' => 'ProductsController@edit']);
    Route::post('/products/{id}', ['as' => 'products.update', 'uses' => 'ProductsController@update']);
    Route::post('/products/delete/{id}', ['as' => 'products.delete', 'uses' => 'ProductsController@delete']);

    // Accessories
    Route::get('/accessories', ['as' => 'accessories.index', 'uses' => 'AccessoriesController@index']);
    Route::get('/accessories/create', ['as' => 'accessories.create', 'uses' => 'AccessoriesController@create']);
    Route::post('/accessories', ['as' => 'accessories.store', 'uses' => 'AccessoriesController@store']);
    Route::get('/accessories/edit/{id}', ['as' => 'accessories.edit', 'uses' => 'AccessoriesController@edit']);
    Route::post('/accessories/{id}', ['as' => 'accessories.update', 'uses' => 'AccessoriesController@update']);
    Route::post('/accessories/delete/{id}', ['as' => 'accessories.delete', 'uses' => 'AccessoriesController@delete']);

    // Transfers
    Route::get('/transfers', ['as' => 'transfers.index', 'uses' => 'TransfersController@index']);
    Route::get('/transfers/create', ['as' => 'transfers.create', 'uses' => 'TransfersController@create']);
    Route::post('/transfers', ['as' => 'transfers.store', 'uses' => 'TransfersController@store']);
    Route::get('/transfers/edit/{id}', ['as' => 'transfers.edit', 'uses' => 'TransfersController@edit']);
    Route::post('/transfers/{id}', ['as' => 'transfers.update', 'uses' => 'TransfersController@update']);
    Route::post('/transfers/delete/{id}', ['as' => 'transfers.delete', 'uses' => 'TransfersController@delete']);
```

Obrázok 6- Ukážka koncepcie Routov pre Admin

Routes pre klientsku časť webu sme vytvorili v cykle, pre všetky jazykové mutácie webu, ktoré sú natiahnuté z konfiguračného súboru. V praxi to znamená, že keď sa rozhodneme rozšíriť našu aplikáciu o ďalšiu jazykovú mutáciu, stačí jej skratku uviesť do konfiguračného súboru a vytvoriť nové migrácie na databázu pre všetky stĺpce s prefixom novej jazykovej mutácie. Tieto Routes pre klientsku časť sme vytvorili tak, že jazyková mutácia, ktorá je predvolená nebude mať svoju skratku v URL adrese, a naopak, ostatné jazykové mutácie budú mať v URL adrese prefix jazykovej mutácie v prvom fragmente URL adresy (napr.: www.domena.sk/en).

3.1.4 Controllers

Pri vytváraní Controllers sme dbali na to aby sme oddelili Controllers určené pre Admin a pre klientsku časť webovej aplikácie, resp. aj pre Controllers určené na autentifikáciu používateľa. Preto Controllers pre administrátorskú časť sme vytvorili

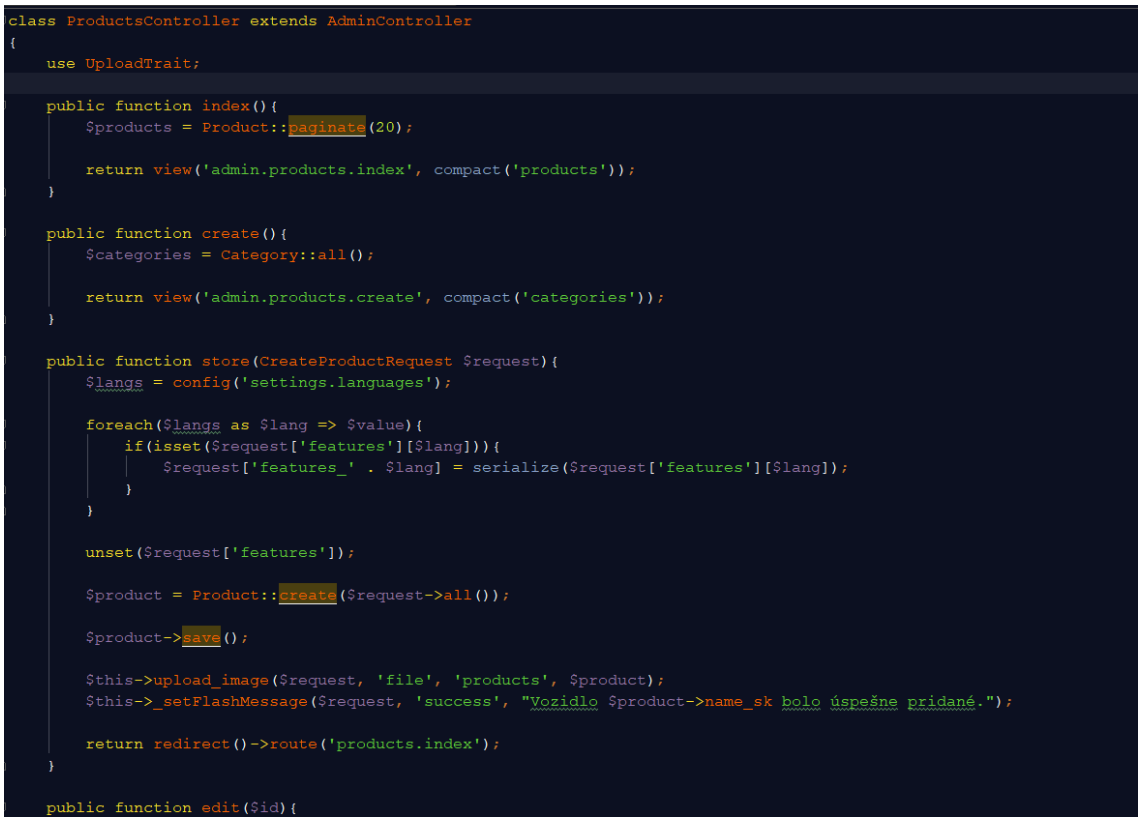
v priečinku Admin, Controllers pre autentifikáciu sme vytvorili do priečinku Auth a Controllers pre klientsku časť sú v koreňovom adresári pre Controllers.

Pri vytváraní Controllers, ale aj celej logiky aplikácie, sme dbali na to, aby bola zachovaná určitá konvencia, a aby sa v projekte vyznal viac ako jeden programátor. Preto sme sa snažili zachovať určitý vzorec medzi názvami tabuliek v databázy, Models, Controllers a Views. Napríklad: tabuľka *products* má Model s názvom *Product*. Controller v Admin rozhraní pre túto tabuľku nesie názov *ProductsController* a je teda uložený v priečinku Admin. Metódy resp. funkcie v Controller majú rovnaký názov ako má View, ktorý daná metóda zobrazuje. Teda metóda *edit* v *ProductsController* vracia View, ktorého cesta je *views/admin/products/edit.blade.php*.

Rovnakú formu konvencie sme sa snažili zachovať v celej aplikácii, aby bolo možné sa ľahko orientovať v kóde celej webovej aplikácie, a aby projekt bol použiteľný aj do budúcna. Myslíme si, že je veľmi dôležité, aby bola zachovaná určitá štruktúra v programovaní webovej aplikácie. Táto štruktúra umožňuje vývoj v tíme bez toho, aby si programátori museli medzi sebou vysvetliť funkcie jednotlivých súčastí webovej aplikácie.

Všetky Controllers v Admin časti webovej aplikácie nám zabezpečujú CRUD operácie na databázu, teda vytváranie, zobrazenie, editovanie a mazanie záznamov v jednotlivých tabuľkách databázy našej webovej aplikácie. Takisto zabezpečujú aj ostatnú logiku pri vytváraní cien alebo nahrávaní obrázkov na server.

Použitie Controllers v kombinácii s Requests nám zabezpečuje validáciu formulárov celej webovej aplikácie, či už sa jedná o formulár pre editovanie produktu alebo o kontaktný formulár v klientskej časti aplikácie.



```
class ProductsController extends AdminController
{
    use UploadTrait;

    public function index(){
        $products = Product::paginate(20);

        return view('admin.products.index', compact('products'));
    }

    public function create(){
        $categories = Category::all();

        return view('admin.products.create', compact('categories'));
    }

    public function store(CreateProductRequest $request){
        $langs = config('settings.languages');

        foreach($langs as $lang => $value){
            if(isset($request['features'][$lang])){
                $request['features_' . $lang] = serialize($request['features'][$lang]);
            }
        }

        unset($request['features']);

        $product = Product::create($request->all());

        $product->save();

        $this->upload_image($request, 'file', 'products', $product);
        $this->_setFlashMessage($request, 'success', "Vozidlo $product->name_sk bolo úspešne pridané.");

        return redirect()->route('products.index');
    }

    public function edit($id){
```

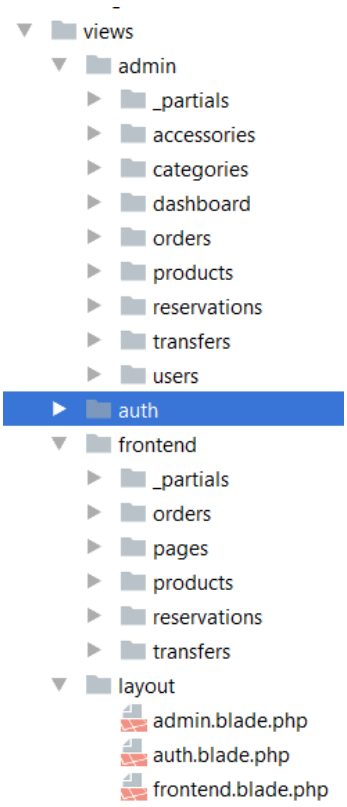
Obrázok 7 - Ukážka triedy ProductsController v Admin časti

V obrázku 7 môžeme vidieť, akou formou sú vytvárané produkty (autá) v našej aplikácii. Proces je veľmi jednoduchý, pri kliknutí v Admin prostredí na tlačidlo ‚*Nový produkt*' je zobrazený View s názvom *create.blade.php*, do ktorého si posielame premennú, v ktorej uchovávame všetky kategórie produktov, aby sme z nich vedeli v danom View vytvoriť HTML tag select. Ďalej môžeme vidieť, že po odoslaní formulára na vytvorenie produktu sa vykoná metóda *store*, v ktorej sa uložia údaje s Request do databázy, uloží sa obrázok pomocou metódy z triedy *UploadTrait*, ktorá je potomkom triedy Trait. Následne sa vytvorí správa o úspešnom vytvorení produktu pomocou metódy *_setFlashMessage*, ktorá je zdedená z triedy *AdminController*.

3.1.5 Views

Pri vytváraní každého View sme využili tzv. Blade v Laraveli. Blade je prostriedok alebo nástroj, ktorý v Laraveli uľahčuje prácu s Views. Blade má svoju určitú syntax, ktorá zjednodušuje napríklad vypisovanie premenných z PHP alebo použitie cyklov, ale aj mnoho iných funkcií. Preto každý View, ktorý sme vytvorili nesie koncovku .blade.php. Laravel takto vie, že sa jedná o View typu Blade.

Všetky Views sme rozdelili na do 4 priečinkov a to *admin, layout, frontend* a *auth*. Ako už hovoria názvy, tak v priečinku *layout* sú všetky Views, ktoré určujú layout našej aplikácie. Priečinok *admin* obsahuje všetky Views, ktoré sú zobrazované v administrátorskej časti našej webovej aplikácie. Priečinok *frontend* uchováva Views, ktoré sú zobrazované v klientskej časti webovej aplikácie. A nakoniec priečinok *auth* obsahuje Views, ktoré sú zobrazované pri autentifikácii používateľa webovej aplikácie, či už sa jedná o administrátora alebo o klienta.



Obrázok 8 - Štruktúra rozdelenia jednotlivých Views webovej aplikácie

Na obrázku 8 je vidieť ako sme zvolili členenie jednotlivých Views našej webovej aplikácie. Vo Views využívame všetky funkcie ktoré nám poskytuje Blade ako napríklad funkciu *extends* alebo funkciu *include*, aby sme neopakovali časti HTML kódu.

3.1.6 Requests

Requests v Laraveli sme použili na validáciu formulárov a takisto na definovanie chybových správ, v prípade neúspešnej validácie. Najväčšie uplatnenie Requests sme zaregistrovali pri tvorbe administračného prostredia, kde je potrebné validovať každý jeden formulár, ktorým vytvárame alebo editujeme jednotlivé záznamy v databáze webovej aplikácie.

Requests takisto používame aj v klientskej časti aplikácie, a to pri vytváraní objednávok a odosielaní kontaktných formulárov. Pri vytváraní objednávok je ich použitie kľúčové nielen kvôli bezpečnosti aplikácie, ale aj kvôli korektnosti údajov objednávky ako sú napríklad cena, informácie o zákazníkovi, objednaných produktoch alebo ďalších informácií.

Requests sme takisto vytvárali s určitou štruktúrou, a to nasledovným spôsobom. Pri vytváraní záznamu v databáze sme zvolili kľúčové slovo *create*, spojili ho s názvom objektu, ktorý vkladáme do databázy a následne doplnili kľúčové slovo *request*, aby bolo jasné, že sa jedná o triedu, ktorá je potomkom triedy Request. Napríklad Request použitý pri vytváraní produktu (auta) nesie názov *CreateProductRequest*.

Pri editovaní záznamov, keby sme potrebovali vytvoriť Request s novými, ale zároveň podobnými validačnými pravidlami, by sme zvolili rovnaký spôsob názvu Request, ale v tomto prípade by prvé kľúčové slovo bolo *update*. Teda pre editovanie produktov by Request niesol názov *UpdateProductRequest* a bol by potomkom triedy *CreateProductRequest* s tým, že by mal upravené niektoré validačné pravidlá, poprípade pridané úplne nové alebo odstránené nepotrebné validačné pravidlá.

```php
public function rules()
{
    return [
        'name_sk' => 'required|max:191',
        'name_en' => 'required|max:191',
        'price_from' => 'required|numeric',
        'price_a' => 'required|numeric',
        'price_b' => 'required|numeric',
        'price_c' => 'required|numeric',
        'price_d' => 'required|numeric',
        'price_e' => 'required|numeric',
        'deposit' => 'numeric',
        'mileage' => 'integer',
        'driver_age' => 'integer',
        'doors' => 'integer',
        'baggage' => 'integer',
        'passengers' => 'integer',
        'gearbox_sk' => 'max:191',
        'gearbox_en' => 'max:191',
        'fuel_sk' => 'max:191',
        'fuel_en' => 'max:191',
        'category_id' => 'integer'
    ];
}

public function messages(){
    return [
        'name_sk.required' => 'Pole Názov SK je povinný údaj',
        'name_sk.max:191' => 'Pole Názov SK má maximálnu dĺžku 191 znakov',
        'name_en.required' => 'Pole Názov EN je povinný údaj',
        'name_en.max:191' => 'Pole Názov EN má maximálnu dĺžku 191 znakov',
        'price_from.required' => 'Pole Cena od je povinný údaj',
        'price_from.numeric' => 'Pole Cena od musí byť číslo',
        'price_a.required' => 'Pole Cena za 1-3 dni je povinný údaj',
        'price_a.numeric' => 'Pole Cena za 1-3 dni musí byť číslo',
        'price_b.required' => 'Pole Cena za 4-11 dní je povinný údaj',
        'price_b.numeric' => 'Pole Cena za 4-11 dní musí byť číslo',
        'price_c.required' => 'Pole Cena za 12-21 dní je povinný údaj',
```

Obrázok 9 - Ukážka CreateProductRequest

Ako môžeme vidieť na obrázku 9, vo funkcii *rules* sú definované validačné pravidlá pre jednotlivé položky Request. Vo funkcii *messages* sú definované chybové hlášky v prípade, že daná validácia je neúspešná.

V obrázku 9 je vidieť, že jednotlivé validačné pravidlá sú definované v asociatívnom poli, kde kľúč nesie názov položky z Request, teda názov inputu v HTML formulári a hodnota obsahuje validačné pravidlá oddelené znakom ‚|‘. V Laraveli je možné definovať aj vlastné validačné pravidlá, avšak pri vývoji našej webovej aplikácie sme nepotrebovali využiť túto možnosť.

Na obrázku 9 je takisto vidieť ako sú definované jednotlivé chybové hlášky. Takisto sú definované pomocou asociatívneho poľa, kde kľúč je vytvorený z názvu položky

Request, potom nasleduje bodka a validačné pravidlo. Hodnota obsahuje chybovú hlášku, v prípade zlyhania validácie pre danú položku s daným validačným pravidlom.

3.1.7 Middlewares

Middlewares sme v našej webovej aplikácii využili v dvoch prípadoch. Prvý prípad je kontrola autentifikácie používateľa pri snahe o prístup do administračného prostredia a druhý prípad je nastavenie jazykovej mutácie pre aplikáciu.

Middleware pre kontrolu autentifikácie používateľa má za úlohu zistiť, či sa jedná o administrátora alebo klienta, v prípade, že sa používateľ webovej aplikácie snaží vstúpiť do administrátorského prostredia. Pokiaľ používateľ, ktorý sa snaží pristúpiť do administrátorského prostredia nie je administrátor, resp. nemá v databáze v stĺpci *is_admin* hodnotu 1, tak je presmerovaný na úvod aplikácie v klientskej časti. Pokiaľ používateľ je administrátor, tak mu Middleware umožňuje pohybovať sa v administrátorskom prostredí.

Midlleware pre nastavenie jazykovej mutácie má za úlohu zistiť prefix v URL adrese, následne zistiť, či sa daný fragment URL adresy nachádza v konfiguračnom súbore, ktorý obsahuje pole skratiek jazykových mutácií aplikácie. Pokiaľ sa nejedná o jednu z jazykových mutácií, tak je nastavená predvolená jazyková mutácia celej webovej aplikácie. Pokiaľ ide o hodnotu, ktorá sa nachádza v konfiguračnom súbore jazykových mutácií, teda sa jedná o jazykovú mutáciu, ktorú naša webová aplikácia podporuje, tak je nastavená táto jazyková mutácia celej webovej aplikácii. Môžeme teda povedať, že tento Middleware slúži ako prepínač jazykových mutácií webovej aplikácie.

3.1.8 Composers, Providers, Traits a ostatné časti

V aplikácii používame aj ďalšie nemenej dôležité súčasti Laravelu ako sú napríklad Composers, Providers alebo Traits. Ostatné použité súčasti Laravelu, ktoré sme využili sú konfiguračné súbory na uchovanie nemenných statických informácií ako sú napríklad veľkosti obrázkov, zoznam jazykových mutácií webovej aplikácie alebo zoznam jednotlivých stavov objednávok. Ďalej sme využili tzv. Seeders, ktoré majú funkciu napĺňanie databázy údajmi. Seeders sme využili napríklad na vytvorenie administrátora.

Composers používame na prenášanie premenných do určitých Views, aby sme nemuseli dané premenné prenášať do Views vo funkciách Controllers. Napríklad do hlavného menu potrebujeme stále dostať premennú, v ktorej sú uložené údaje o všetkých záznamoch z tabuľky *transfers* v databáze našej webovej aplikácie. Keďže View, ktoré obsahuje hlavné menu, je súčasťou všetkých Views v klientskej časti aplikácie tak, aby

sme sa vyhli manuálne prenášaniu danej premennej do Views, v každej funkcii každého Controller, ktorý smeruje do klientskej časti aplikácie, použijeme Composer. Ten zabezpečí prenos tejto premennej do daného View vždy keď je dané View renderované, bez toho, aby sme danú premennú museli prenášať vo funkcii Controllers, ktoré súvisia s klientskou časťou webovej aplikácie.

Využitie Traits sme uplatnili pri nahrávaní obrázkov do našej webovej aplikácie resp. na server. V triede *UploadTrait* sme zabezpečili logiku zapísania nahratého obrázku do databázy do tabuľky *images*, kde sú vedené záznamy o všetkých obrázkoch nahratých do webovej aplikácie používateľom resp. administrátorom. Takisto je tu vyriešená úprava obrázkov na jednotlivé veľkosti, ktoré sme definovali v konfiguračnom súbore určenom pre uchovanie informácií o vlastnostiach nahrávaných obrázkov ako sú šírka alebo výška v pixeloch.

3.2 ADMINISTRAČNÝ SYSTÉM

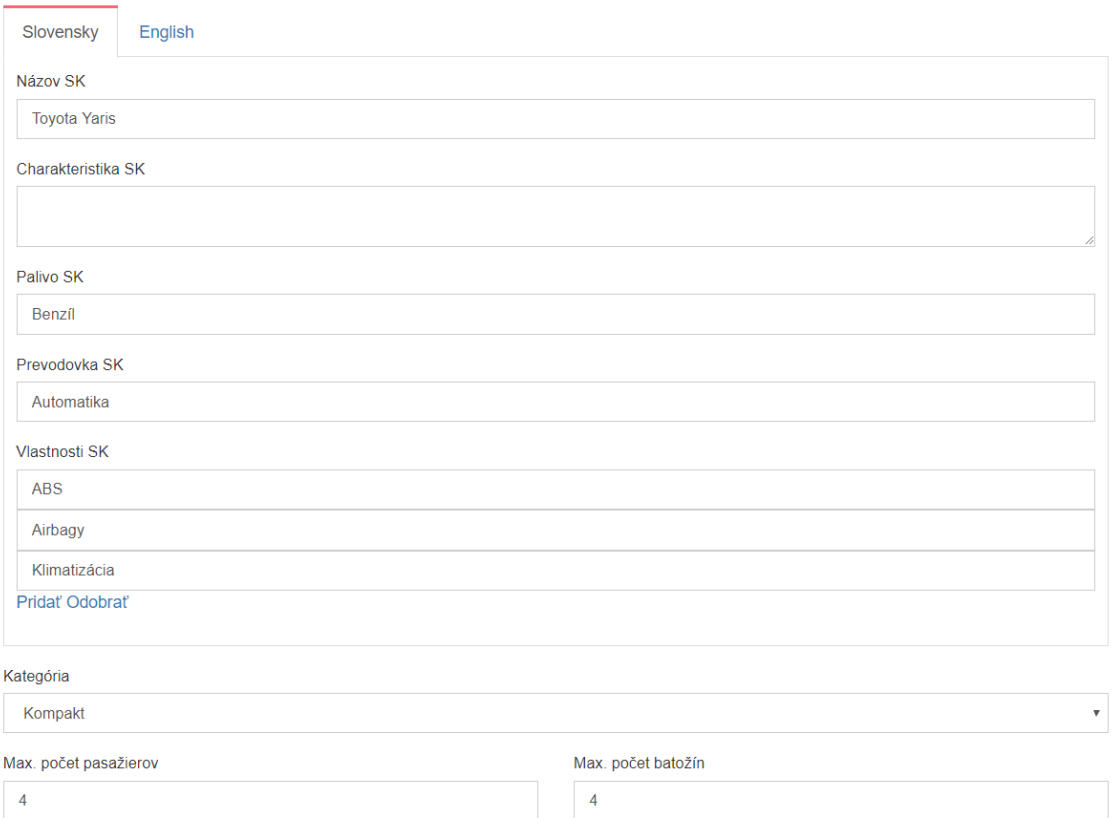
Administračný systém v našej webovej aplikácii predstavuje časť, do ktorej má prístup len administrátor, a v ktorej si vie zobrazovať, vytvárať, editovať a vymazávať záznamy v databáze. Je tu umiestnená evidencia kategórií, produktov (áut), doplnkov, objednávok áut, transferov, rezervácií transferov a evidencia používateľov. Administračný systém webovej aplikácie sme vyvíjali, tak aby bol prehľadný, jednoduchý, intuitívny, a aby v ňom neboli žiadne zbytočné a mätúce informácie. V praxi to znamená, že administrátor našej webovej aplikácie nepotrebuje žiadne bližšie vysvetlenie alebo školenie ohľadom administračného systému na to, aby ho vedel plne používať.

Takisto administračné prostredie je predpripravené na pridávanie rôznych jazykových mutácií. Pri pridaní jazykovej mutácie do poľa v konfiguračnom súbore a následnom vytvorení a spustení migrácií na databázu pre vytvorenie potrebných stĺpcov s prefixom novej jazykovej mutácie, nie je potrebné robiť nič viac a administračné prostredie automaticky vie pracovať s novou jazykovou mutáciou.

3.2.1 Autá (Produkty)

Časť Autá v administrátorskom prostredí webovej aplikácie tvorí evidencia troch celkov a to kategórií, áut (produktov) a doplnkov.

V časti pre kategórie zobrazujeme jednotlivé kategórie produktov. Máme tu možnosť vytvárať nové kategórie alebo editovať už existujúce kategórie produktov. Takisto poskytujeme možnosť aj mazať jednotlivé záznamy kategórií produktov.



Časť produktov (áut) je najrozsiahlejšou z týchto častí, keďže informácie o produktoch obsahujú najväčší počet stĺpcov v databáze z pomedzi týchto troch častí. Okrem klasických CRUD operácií, dokážeme pri vytváraní alebo editovaní produktu nahrať profilový obrázok daného produktu, ktorý je spracovaný a následne uložený na server a do databázy. Takisto v tejto časti pri vytvorení alebo editácií vieme priradiť produkt ku existujúcej kategórii.

V doplnkoch vedieme evidenciu doplnkov. Uchovávame tu hlavne názov doplnku a cenu za jeden kus doplnku. Doplnky je možné zvoliť si pri vytváraní objednávky v klientskej časti aplikácie.

3.2.2 Transfery

Evidencia transferov predstavuje zoznam transferov pre jednotlivé mestá. Pri vytváraní a editovaní transferov ukladáme hodnoty ako názov transferu, miesto v ktorom transfer poskytujeme a takisto cenu transferu pre dané mesto. Transfery v našej webovej aplikácii predstavujú prenájom zabezpečenia dopravy v danom meste z bodu A do bodu B za cenu transferu pre dané mesto.

3.2.3 Používatelia

V evidencii používateľov zobrazujeme všetkých používateľov, ktorí majú status klienta. Teda v tejto časti nie je zobrazený zoznam používateľov, ktorí sú administrátori webovej aplikácie. Táto časť administračného systému sa odlišuje tým, že tu nie je možné vykonávať všetky CRUD operácie nad tabuľkou *users* v databázi našej webovej aplikácie. Umožnili sme iba zobrazovať informácie o jednotlivých používateľoch, resp. klientoch webovej aplikácie. Informácie, ktoré si administrátor vie zobraziť, sú informácie o mene a priezvisku, takisto o mieste bydliska, kontaktné údaje klienta a takisto aj informácie o firme používateľa, v prípade, že nejaké evidujeme.

Okrem základných informácií o používateľovi zobrazujeme aj jednotlivé objednávky, ktoré používateľ webovej aplikácie vytvoril, keď bol prihlásený do svojho používateľského účtu. Priamo odtiaľto sa vieme dostať cez odkaz na danú objednávku, čo nám zjednodušuje a urýchli pohyb v administračnom prostredí webovej aplikácie.

3.2.4 Objednávky Áut

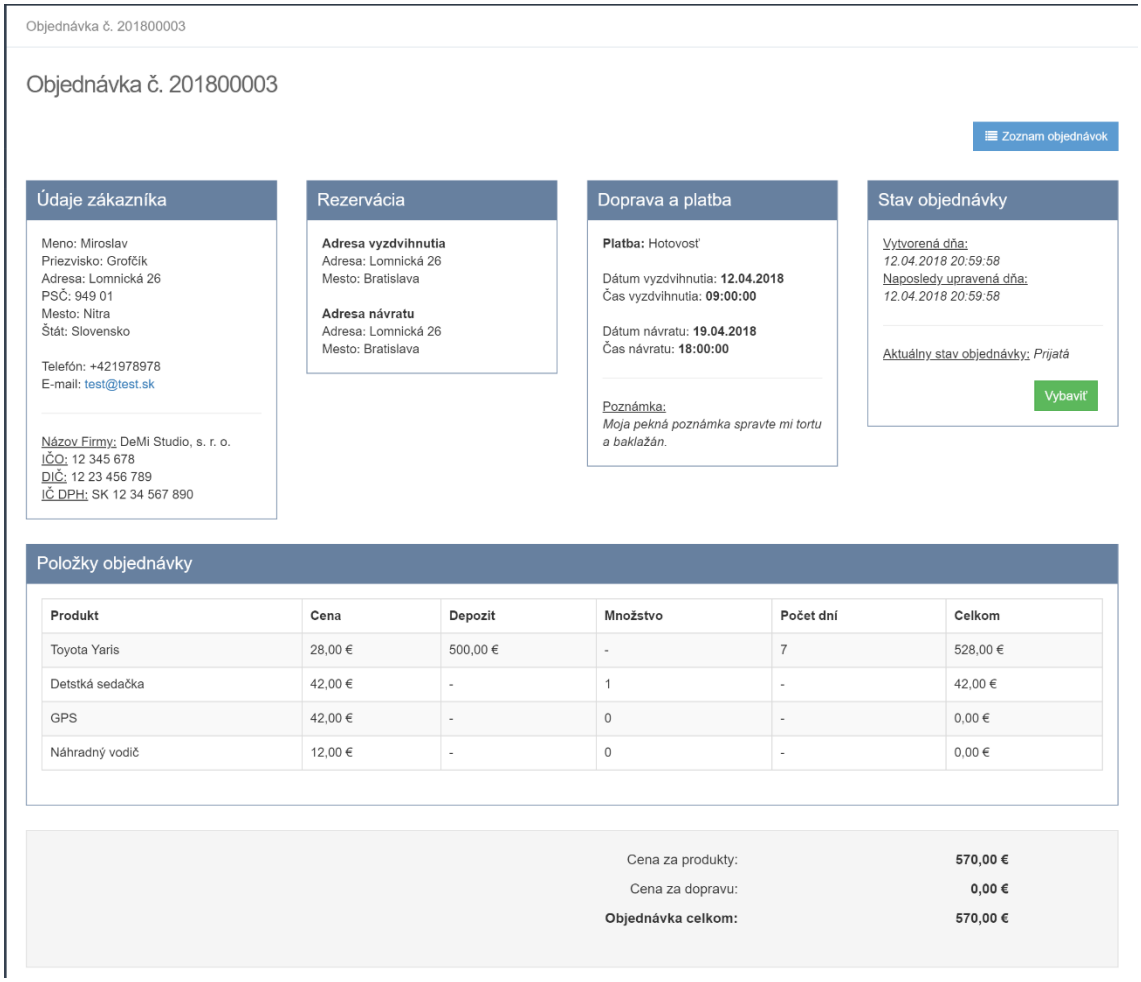
V administrátorskom prostredí webovej aplikácie sme vytvorili časť venovanú evidencii objednávok, ktorá sa delí na dve časti.

Prvú časť tvorí evidencia objednávok prenájmu áut. V tejto časti zobrazujeme všetky existujúce objednávky prenájmu áut, usporiadané v tabuľke od najnovšej objednávky po najstaršiu. V tabuľke sme sa rozhodli zobraziť tie najhlavnejšie informácie a to číslo objednávky, stav objednávky, meno a priezvisko zákazníka, e-mail, typ platby, celkovú cenu objednávky a nakoniec dátum vytvorenia objednávky. V evidencii objednávok sme vytvorili filter, aby sme umožnili administrátorovi webovej aplikácie vyhľadávať objednávky podľa čísla objednávky, stavu objednávky, mena zákazníka alebo cenového rozpätia objednávky. Vytvoreným filtrom sme uľahčili vyhľadávanie konkrétnej objednávky v prípade veľkého počtu vytvorených objednávok. V tejto časti sme takisto umožnili aj zobraziť detaily objednávky. Po otvorení detailu objednávky administrátor vidí kompletné údaje danej objednávky. Medzi tieto údaje patria informácie o zákazníkovi, informácie o mieste vyzdvihnutia a mieste návratu, informácie o dátume a čase vyzdvihnutia a návratu, typ platby, prípadná poznámka od zákazníka, informácie kedy bola objednávka vytvorená a kedy bola naposledy upravená, a potom samozrejme tabuľku s jednotlivými položkami objednávky.

Tabuľka položiek objednávky obsahuje informácie o tom aké auto si zákazník objednal, cenu, výška depozitu, počet dní na ktorý si auto objednal. Ďalej táto tabuľka

obsahuje informácie o doplnkoch, ktoré si zákazník objednal a to cenu, množstvo a celkovú cenu. Pod tabuľkou položiek sme spravili sumár objednávky kde administrátor vidí celkovú cenu objednávky, teda cenu všetkých položiek objednávky dokopy.

V detaile objednávky môže administrátor meniť stav objednávky. Stavy objednávok sú definované v konfiguračnom súbore určenom pre stavy objednávok webovej aplikácie. V tomto prípade sú to iba dva stavy *Prijatá* objednávka a *Vybavená* objednávka.



Obrázok 10 - Vzorová ukážka detailu objednávky prenájmu auta

Na obrázku 10 môžeme vidieť všetky vyššie spomenuté údaje objednávky, ktoré webová aplikácia zobrazuje administrátorovi webovej aplikácie. Údaje sme sa snažili vypísať čo najprehľadnejšie, aby sa administrátor ľahko orientoval vo všetkých informáciách o objednávke prenájmu auta.

3.2.5 Rezervácia transferov

V tejto časti administračného prostredia zobrazujeme objednávky, ktoré súvisia s rezerváciou transferov. Táto časť je veľmi podobná časti vyhradenej pre objednávky

prenájmu áut. Takisto sme v tejto časti umiestnili filter na vyhľadávanie medzi jednotlivými existujúcimi objednávkami na rezerváciu transferov.

Rozdiel v tejto časti je ten, že v detaile objednávky zobrazujeme iné informácie, resp. informácie o záznamoch v tabuľke *reservations* v databáze webovej aplikácie.

3.3 REZERVAČNÝ SYSTÉM ÁUT

Rezervačný systém áut je jedna z dvoch klientskych častí webovej aplikácie. V tejto časti webovej aplikácie je zobrazený výpis všetkých evidovaných produktov resp. áut pridaných administrátorom v administračnom prostredí. Takisto je tu možnosť zobraziť detail produktu, kde sú vypísané všetky dôležité informácie o aute a to maximálny počet pasažierov vozidla, druh pohonnej hmoty, počet dverí na vozidle, depozit za auto, najnižšia možná cena a ďalšie informácie. V detaile produktu je zobrazený aj obrázok vozidla.

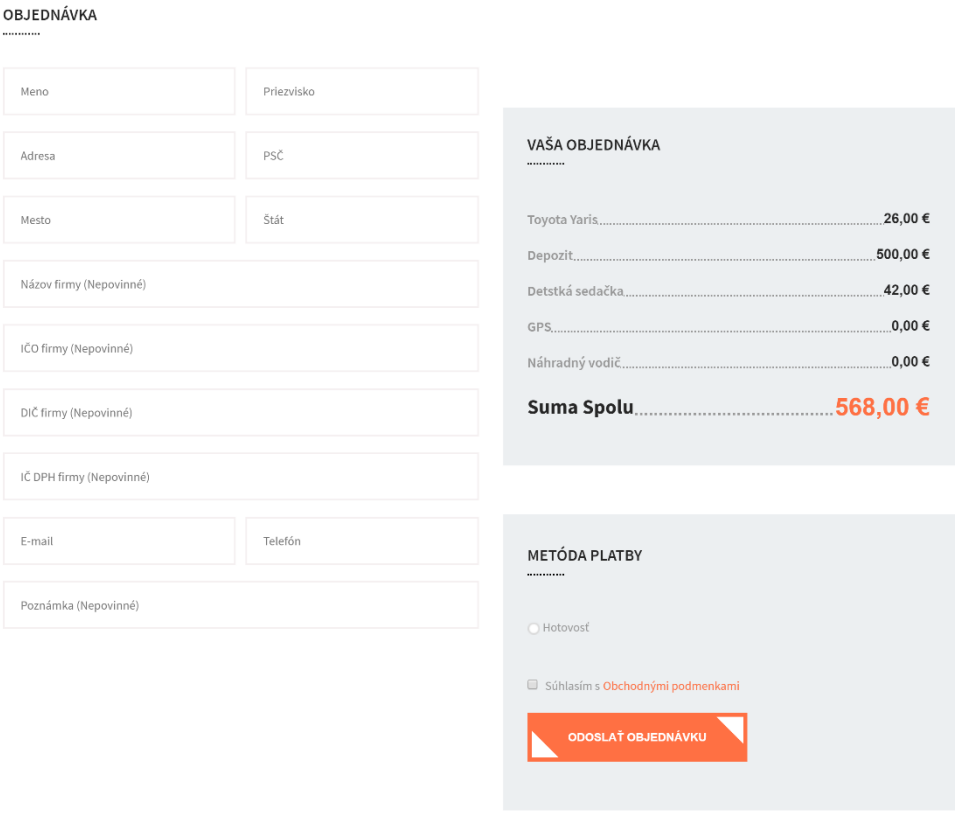
V detaile produktu sa nachádza prvý krok vytvorenia rezervácia, teda prvý formulár, kde klient vyplní informácie o mieste, dátume a čase vyzdvihnutia a návratu. Podobný formulár sa nachádza aj na úvodnej stránke webovej aplikácie s tým rozdielom, že je doplnený o výber vozidla pomocou HTML tagu *select*, aby sme vedeli identifikovať vozidlo, ktoré si klient chce prenajať.

3.3.1 Objednávka auta

Ako je spomenuté vyššie, 1. krok objednávky auta je vykonaný buď cez formulár na domovskej stránke webovej aplikácie alebo priamo v detaile produktu. V druhom kroku objednávky sa nám zobrazí formulár pre vyplnenie počtu doplnkov, ktoré si prajeme objednať spolu s produktom.

V druhom kroku sa nám takisto zobrazí cena produktu vyrátaná podľa dĺžky doby, na ktorú si chceme vozidlo objednať a takisto samozrejme aj depozit za dané vozidlo. V tejto časti sme spravili aj sumár ceny za aktuálne položky objednávky, zatiaľ bez doplnkov.

V treťom kroku objednávky klient vypĺňa osobné údaje, údaje o mieste bydliska, informácie o firme pokiaľ si praje vystaviť doklad na firmu, e-mail, telefónne číslo a môže doplniť poznámku. V tejto časti je spravený kompletný sumár objednávky a konečná celková suma objednávky, ktorá pozostáva s ceny za produkt, depozitu za produkt a cien jednotlivých doplnkov. V tejto časti si klient takisto vyberá možnosť platby a musí zaškrtnúť políčko, ktoré hovorí o súhlase s obchodnými podmienkami.



Obrázok 11 - 3. krok objednávky prenájmu auta

V obrázku 11 môžeme vidieť formulár z jednotlivými políčkami záverečného kroku objednávky prenájmu vozidla.

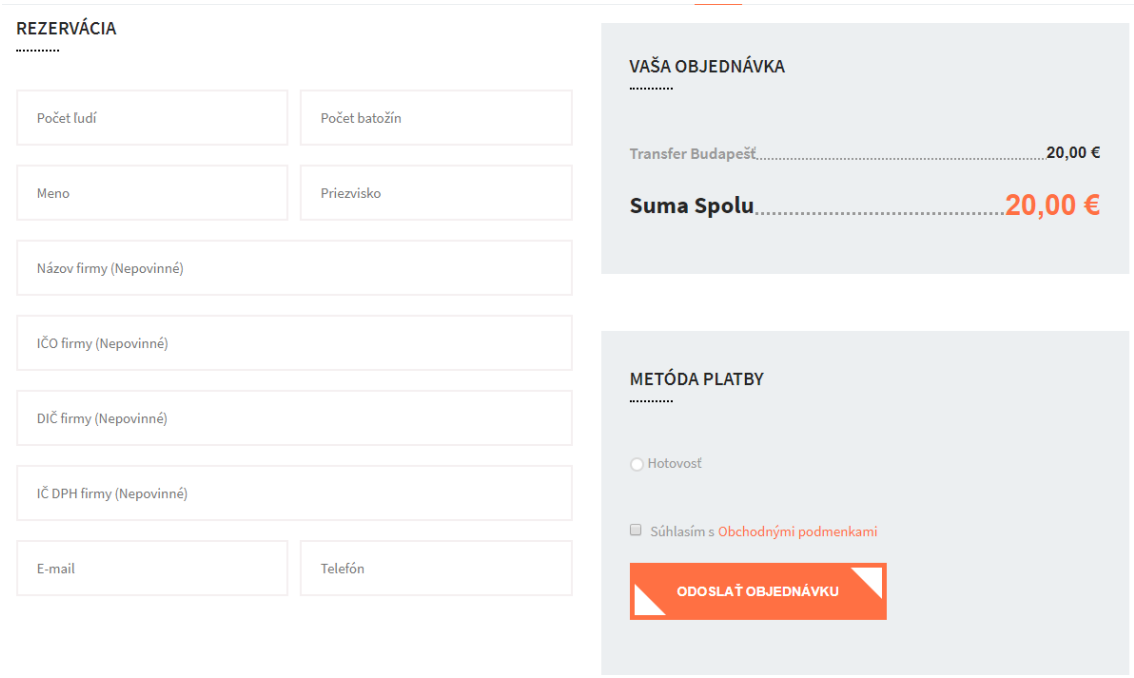
3.4 REZERVAČNÝ SYSTÉM TRANSFEROV

Rezervačný systém transferov je druhá časť klientskej časti webovej aplikácie. V tejto časti aplikácie je možné zobraziť rezervačný formulár pre jednotlivé transfery. Rezervácia transferu sa skladá iba z dvoch krokov. Jednotlivé odkazy na transfery resp. rezervačné formuláre pre dané transfery sme umiestnili priamo do hlavného menu webovej aplikácie.

3.4.1 Rezervácia transferov

Prvý krok rezervácie transferu je zobrazený hneď po kliknutí na odkaz v hlavnom menu, ktorý odkazuje na konkrétny transfer. Transfer nesie napríklad názov ‚*Transfer Budapešť*', klientovi je teda jasné pod aké mesto daný transfer a teda aj rezervačný formulár patrí. V prvom kroku klient vypĺňa údaje o adrese vyzdvihnutia, dátume a čase vyzdvihnutia a cieľovej adrese.

V druhom kroku sa klient dostane k záverečnej časti objednávky na transfer, kde musí vyplniť polia, ktoré sa týkajú osobných údajov, firemných údajov a takisto musí uviesť počet prevážaných osôb a počet batožín. V tomto kroku klient vidí aj cenu transferu resp. objednávky. V tomto kroku je potrebné zvoliť spôsob platby a súhlasiť s obchodnými podmienkami.



Obrázok 12 - Posledný krok rezervácie transferu

ZÁVER

Podarilo sa nám teda vytvoriť hlavný výstup tejto práce a to webovú aplikáciu, ktorá funguje na princípe rezervačného systému. Vytvorili sme kvalitné administračné rozhranie, ktoré je intuitívne a ľahko sa v ňom orientuje každý používateľ. Zanalyzovali sme problematiku vývoja webových aplikácií, priblížili sme si výhody a nevýhody jednotlivých technológií, ktoré sa pre vývoj webových aplikácií v dnešnej dobe používajú.

Medzi klady nášho riešenia zaraďujeme jednoduchosť celého rezervačného systému, tak ako aj jednoduchosť administračného systému. Bežný používateľ nemá problém sa vyznať vo výslednej aplikácií. Administrátor nepotrebuje technické zručnosti, aby sa vyznal v administračnom systéme, administračný systém je vybudovaný tak, aby ho pochopil aj laik.

Do budúcna je systém možné rozšíriť o akékoľvek moduly. Napríklad je možné pridať iné jazykové mutácie, pridať iné typy vozidiel, je možné vytvoriť systém zliav alebo vernostný systém, je možné napojiť aplikáciu na sociálne médiá, a tak ďalej.

V práci sme zanalyzovali front-end aj back-end frameworky a následne sme použili tie najpopulárnejšie. Celú aplikáciu sme postavili na PHP frameworku Laravel, ku ktoré sme použili knižnicu jQuery, pre lepšiu interakciu s HTML a CSS.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

ANGULAR.IO. 2017. *Angular, frontend javascript framework documentation*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <https://angular.io/docs/ts/latest/>.

BITTNER HONZA. 2015. *Technologie pro vývoj webových aplikací – Úvod a porovnání*. [online]. 2015. [cit. 2017-04-23]. Dostupné na internete: <https://www.itnetwork.cz/html-css/webove-aplikace/technologie-pro-vyvoj-webovych-aplikaci-uvod-a-porovnani/>.

CODESIDO IVAN. 2009. *What is front-end development?*. [online]. 2009. [cit. 2017-04-23]. Dostupné na internete: <https://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost/>.

JQUERY.COM. 2017. *Jquery, frontend javascript framework*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <https://api.jquery.com/>.

LARAVEL.COM. 2017. *Laravel, backend php framework documentation*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <https://laravel.com/docs/5.4/installation/>.

PLURALSIGHT.COM. 2015. *Difference between the front-end and back-end?*. [online]. 2015. [cit. 2017-04-23]. Dostupné na internete: <https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end/>.

QUANTASOFT.SK. 2017. *Vývoj webových aplikácií*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <http://www.quantasoft.sk/Web/Vyvoj-webovych-aplikacii.aspx/>.

SEVILLEJA CHRIS. 2014. *A guide to using Eloquent ORM in Laravel*. [online]. 2014. [cit. 2017-04-23]. Dostupné na internete: <https://scotch.io/tutorials/a-guide-to-using-eloquent-orm-in-laravel/>.

SITEPOINT.COM. 2015. *Best php frameworks of 2015*. [online]. 2015. [cit. 2017-04-23]. Dostupné na internete: <https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/>.

TECHNOPEDIA.COM. 2017. *Backend developer*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <https://www.techopedia.com/definition/29568/back-end-developer/>.

TUTORIALSPOINT.COM. 2017. *Tutorials of web development technologies*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <https://www.tutorialspoint.com/web_development_tutorials.htm>.

W3SCHOOLS.COM. 2017. *Web development technologies*. [online]. 2017. [cit. 2017-04-23]. Dostupné na internete: <https://www.w3schools.com/>.

ZOZNAM PRÍLOH

Príloha A – Kompaktný disk

Priložený disk obsahuje:

- bakalársku prácu vo formáte PDF
- zdrojový kód aplikácie vyvinutý v PHP frameworku Laravel