



**FAKULTA
INFORMAČNÍCH
TECHNOLGIÍ
ČVUT V PRAZE**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název: API pro backend eshopu
Student: Radomír Koudela
Vedoucí: Ing. Jiří Hunka
Studijní program: Informatika
Studijní obor: Webové a softwarové inženýrství
Katedra: Katedra softwarového inženýrství
Platnost zadání: Do konce zimního semestru 2021/22

Pokyny pro vypracování

Cílem této práce je realizace API pro vznikající novou administraci e-shopu Stylka.cz. Vzhledem k velkému rozsahu funkcí daného e-shopu je třeba zaměřit se především na všechny procesy týkající se objednávek, případně produktů. API musí respektovat souběžný běh současné staré administrace.

Postupujte v těchto krocích:

Analyzujte současné řešení e-shopového backendu a jeho databázi.

Proveďte analýzu možných řešení a navrhnete vhodné API.

Při návrhu přihlídněte k budoucímu rozšiřování API pro další části provozu e-shopu.

Konzultujte návrh s kolegyní Iuliia Evseenko, která API bude využívat.

Implementujte výsledný návrh.

V implementaci neopomeňte testování.

Zhodnoťte výsledné řešení, navrhnete úpravy do budoucna.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 26. února 2020



**FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE**

Bakalářská práce

API pro backend eshopu

Radomír Koudela

Katedra Softwarového inženýrství

Vedoucí práce: Ing. Jiří Hunka

7. ledna 2021

Poděkování

Děkuji panu Hunkovi za vedení této bakalářské práce a panu Matouškovi za zodpovídání veškerých dotazů potřebných pro zhotovení mé práce. Taktéž děkuji kolegyni Iuliia Evseenko za své připomínky k této práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2021

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2021 Radomír Koudela. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Koudela, Radomír. *API pro backend eshopu*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zaměřuje na vytvoření plně funkčního aplikačního rozhraní pro eshop www.stylka.cz. Kapitoly odpovídají etapám při vývoji běžného software. Nejdříve je analýza současného řešení (AS-IS), následovaná návrhem nového řešení (TO-BE) obsahující analýzu technologií, specifikaci požadavků a návrh vycházející z architektury moderního frameworku Laravel. V druhé polovině se práce zabývá samotnou implementací a testováním, ve kterých se představují vymoženosti frameworku Laravel. Aplikační rozhraní vychází z populární architektury MVC, je autentizováno pro administrátory a zákazníky, je autorizováno, hlídá správnost vstupů a je plně zdokumentováno. Projekt vzniká ve spolupráci s další studentkou fakulty informačních technologií ČVUT. Na závěr práce je zhodnoceno naplnění cílů této práce a možné úpravy do budoucna.

Klíčová slova eshop, API, backend, Laravel, MVC, PHP, REST, Fortify

Abstract

This thesis focuses on creation of fully working application interface for eshop www.stylka.cz. Chapters reflect stages of development of common software.

First there is an analysis of contemporary solution (AI-IS), followed by the design of new solution (TO-BE) consisting of analysis of technologies, specification of requirements and design basing on the architecture of modern Laravel framework. In the second half this thesis concerns the very implementation and testing in which there are introduced conveniences of Laravel framework. Application interface bases on popular architecture MVC, is authenticated for administrators and customers, is authorized, checks validity of inputs and it is fully documented. The project arises in collaboration of another student of faculty of information technology ČVUT. In conclusion there is an assessment of completion of goals of this thesis and possible adjustments at some time in the future.

Keywords eshop, API, backend, Laravel, MVC, PHP, REST, Fortify

Obsah

Úvod	1
Cíle	1
 I Teoretická část	 3
1 Současný stav	5
2 Analýza	7
2.1 Co je potřeba vytvořit	7
2.2 Programovací jazyk a frameworky	7
2.2.1 Analýza	8
2.3 Formát dat	9
2.4 Webové služby	9
2.4.1 SOAP	9
2.4.2 REST	10
2.4.3 Analýza	10
2.5 Další technologie	10
3 Specifikace požadavků	11
3.1 Funkční požadavky	11
3.2 Nefunkční požadavky	13
4 Architektura aplikace	15
4.1 MVC architektura	15
4.2 Router	16
4.3 Autentizace	17
4.4 Autorizace	17
4.5 Validace	17
4.6 Testování	18

II Praktická část	19
5 Realizace	21
5.1 Model	21
5.2 View	22
5.3 Controller	23
5.3.1 OrderController	24
5.3.1.1 Update	24
5.3.1.2 Store	24
5.3.1.3 Delete	25
5.4 Service	25
5.5 Router	25
5.6 Dokumentace	26
5.7 Databáze	26
5.8 Autentizace	26
5.9 Autorizace	28
5.10 Validace	29
6 Testování	33
6.1 TestDomain_setupValue	33
6.2 TestEshop	34
6.2.1 Správné požadavky	35
6.2.1.1 POST	35
6.2.1.2 PUT	35
6.2.1.3 GET	36
6.2.1.4 DELETE	36
6.2.2 Neprávné požadavky	37
6.2.2.1 Požadavek od uživatele pozbývajícího oprávnění	37
6.2.2.2 Požadavek s nesprávně zadaným vstupem	37
6.2.2.3 Požadavek se špatně zadanou URL	38
6.2.2.4 Požadavek na neexistující objekt	38
Závěr	39
Literatura	41
A Seznam použitých zkratk	43
B Obsah příloženého CD	45

Seznam obrázků

1.1	Částečné schéma databáze	6
2.1	Používanost Laravel vs. Symfony [13]	8
2.2	Struktura JSON [9]	9
6.1	Požadavek se špatně zadanou URL	38
6.2	Požadavek na neexistující objekt	38

Úvod

Charakteristikou dnešní doby je čím dál větší důraz na svět internetu, který postupně „vytlačuje“ svět takzvaně reálný. Tento proces je markantní zvláště ve světě nakupování, kdy eshopy postupně vytlačují kamenné obchody jednoduše proto, že pro zákazníka je mnohdy daleko jednodušší (a za stávající situace i společensky přijatelnější a bezpečnější) několikrát kliknout na obrazovce počítače než se vléci do města. Často je to i kvůli vysoké konkurenci, která se nachází jen na jednom místě – na internetu, i daleko levnější.

Svou prací bych tak chtěl přispět i ke vylepšování možností českého internetu, z čehož bude těžit především běžný spotřebitel. Mou motivací při výběru této práce však byla i možnost zdokonalit se a naučit se novým dovednostem ze světa PHP a realizovat application programming interface za pomoci zvolených technologií.

Struktura bakalářské práce je rozřazena do oddílů odpovídajícím modelu životního cyklu vývoje software, a sice analýza, design, implementace a testování. K tomu se zde samozřejmě nacházejí další kapitoly typické pro každou bakalářskou práci.

Na mou práci práci navazuje kolegyně Iuliia Evseenko ve svojí diplomové práci, která se zaměří na frontendovou část této práce. Jedná se o studentku magisterického studia na FIT ČVUT.

Cíle

Hlavním cílem této práce je sestavit funkční aplikační rozhraní pro administraci eshopu www.stylka.cz v jazyku PHP. Tento výstup odpovídá náležitostem správného eshopu, plní především všechny procesy týkající se objednávek, ale neomezuje se pouze na ně a odpovídá požadavkům kolegyně Iuliia Evseenko.

Každé správné API je detailně zdokumentováno a u toho projektu tomu nebude jinak.

Součástí práce je i drobné rozšíření databáze, ale nikoli úprava, jelikož databáze musí splňovat současný běh současné staré administrace.

Nezbytným výstupem jsou i funkční soubory s testy, které zajišťují, že daná aplikace má bezchybný chod. Testy se zaměřují především na business logiku aplikace, ale neopomíjí i zdánlivě jednoduché operace.

Část I

Teoretická část

Současný stav

Současný stav administrace eshopu www.stylka.cz je proveden za pomoci open-source platformy OpenCart.

„Open-cart systém pro správu eshopů. Je založený na PHP, využívá MySQL databáze a HTML komponent. Zajišťuje podporu pro různé jazyky i měny. Je zdarma dostupný pod licencí GNU General Public Licence.“ [1]

Stávající řešení je monolitní, tedy bez rozdělení na vrstvy, což významně omezuje rozšiřitelnost a modulárnost tohoto řešení, protože za této situace se často musí mazat velké množství kódu a to je velmi neefektivní.

Od této platformy se odvíjí i podoba databáze. Databáze v dnešní podobě plní svou základní funkci ale přesto má určité nedostatky:

1. Chybí jí explicitně definované cizí klíče. To má pak za nevýhodu to, že se v databázi mohou objevit řádky, které tam nemají co dělat, např. potomek bez rodiče. Tato vlastnost deleguje zodpovědnost na API, které vždy napsané bezchybně není, ať člověk chce či ne. Navíc velmi znesnadňuje mazání.
2. Obsahuje tabulky a atributy, které jsou prázdné. Tudíž neplní žádnou funkci, jsou nadbytečné a databázi zbytečně znepráhledňují.

Částečné schéma databáze je možno zhlédnout na obrázku 1.1. Jména tabulek jsou kvůli lepší čitelnosti pozměněna, většina má předponu `_oc` a žádné nezačínají na velké písmeno. V tabulkách se taktéž nachází atributy primárních klíčů.

1. SOUČASNÝ STAV



Obrázek 1.1: Částečné schéma databáze

Analýza

2.1 Co je potřeba vytvořit

Protože současné řešení je nepostačující, je třeba vytvořit API, které naše potřeby splňuje. API, z anglického jazyka jako application programming interface, je soubor podprogramů, funkcí a tříd poskytující služby, které může využít jiný software [2]. Jedná se o business vrstvu, která často funguje tak, že přijímá a zodpovídá dotazy poslané přes prohlížeč, a sahá pro data do databáze, případně je mění [3]. Typicky se zde definují typy volání, možné formáty dat atd [3].

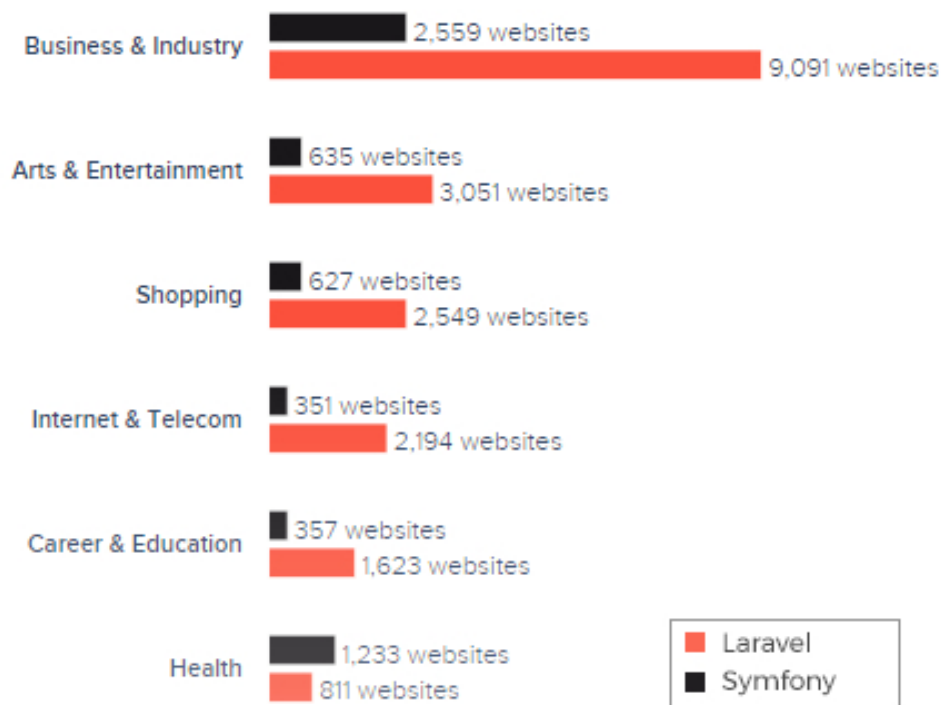
2.2 Programovací jazyk a frameworky

V prvé řadě je třeba vybrat technologii pro vytvoření samotného API. Vedoucí práce mi dal na výběr mezi frameworky Laravel a Symfony (resp. Symfony 2). V obou těchto frameworkcích se vyvíjí v jazyku PHP, jsou open-source a oba jsou postavené na návrhovém vzoru MVC.

Jak uvedeno v [12] výhody Laravel oproti Symfony:

- Jednoduchá integrace cache systémů jako Redis.
- Opravuje nejvíce kritická místa v bezpečnosti, jako SQL injection, cross-site scripting, and cross-site request forgery.
- Automatizuje testování s PHPUnit.
- Minimalizuje repetitivní úlohy. Automatizuje funkce jako autentizace, sessions, caching a routing.
- Laravel nabízí lepší výkonnost za použití méně kódu.
- V dnešní době je využívanější než Symfony.

2. ANALÝZA



Obrázek 2.1: Používanost Laravel vs. Symfony [13]

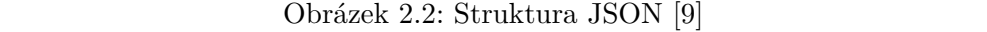
2.1.

Jak uvedeno v [12] výhody Symfony oproti Laravel:

- Komponenty symfony jsou znovupoužitelné a oddělené.
- Symfony profiler je jedním z nejlepších prvků frameworku pro sledování chování jakékoli aplikace.
- Hodí se na komplexní a složité projekty.
- Podporuje více databází.

2.2.1 Analýza

Ačkoli oba jsou to frameworky velmi vhodné, Laravel se zdá jako lepší volba. Vytvírá se v něm rychleji, má lepší výkon, má strmější křivku učení, což je pro mě jako pro začátečníka dost podstatné. A už to, že v dnešní době je Laravel používanější, o něčem vypovídá.



„JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojem. Je založen na podmnožině programovacího jazyka JavaScript (...). JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé programátorům jazyků rodiny C (C, C++, C, Java, JavaScript, Perl, Python a dalších). Díky tomu je JSON pro výměnu dat opravdu ideálním jazykem.“ [8] Jakou má formát JSON strukturu, je možno

vidět na obrázku 2.2.

3.4.1 SOAP

„SOAP, což je zkratka pro Simple object access protocol, je protokol, který definuje, jak dva objekty v různých procesech prostřednictvím výměny dat v podobě XML.“ [4] Jak uvedeno v [4] jeho nespornými výhodami jsou:

- Může být implementován pomocí libovolného jazyka a spuštěn na jakékoli platformě.

2. ANALÝZA

- Lze jej přenášet pomocí libovolného protokolu schopného přenášet text, obvykle HTTP nebo SMTP.
- V drtivé většině případů nemá problém se dostat skrz firewall.

2.4.2 REST

REST, což je zkratka pro Representational state transfer, je architektonický styl, přičemž webovým službám využívající tento styl říkáme RESTful Web services. Ty umožňují přistupovat k datům a měnit je za pomoci předem definovaných bezstavových operací. [5]

Jak uvedeno v [7] benefity technologie REST oproti protokolu SOAP jsou:

- REST nabízí více formátů dat, ne jen XML.
- Kvůli formátu JSON je s REST relativně jednoduchá práce, nemá složitý zápis.
- REST je rychlý a výkonný, zejména při použití caching.
- Jedná se o nejvyžívanější protokol a je dnes daleko více podporován.

2.4.3 Analýza

Od našeho systému požadujeme, aby poskytoval data ve formátu JSON a byl co nejvýkonnější, proto se REST jeví jako nejlepší volba. V Laravelu se sice SOAP dá využít, ale není oficiálně podporován, není k němu oficiální dokumentace. „*Konsensus mezi odborníky dnes je, že REST má typicky přednost oproti SOAP, pokud zde není opravdu dobrý důvod použít SOAP (...).*“ [7] V tomto projektu tomu nebude jinak, žádnou z výhod uvedených v předchozím textu nepotřebujeme.

2.5 Další technologie

Další technologie jsou vybrány na základě toho, zda jsou zdokumentovány v oficiální dokumentaci frameworku Laravel. Usnadní to realizaci a předejte se nechtěným překvapením například v podobě nekompatibility některých prvků. Více o tom bude v dalších kapitolách.

Specifikace požadavků

Součástí každého IT projektu je i specifikace požadavků. Tradičně se využívá rozdělení na funkční a nefunkční požadavky, přestože existují i jiná rozdělení.

funkční : specifikují, co má systém dělat v závislosti na přijatém vstupu;

nefunkční : specifikují kritéria, která by daný systém měl splňovat.

Funkční i nefunkční požadavky vychází z konzultací s vedoucím práce, konzultantem a kolegyní Iuliia Evseenko.

3.1 Funkční požadavky

1. Formát dat

- Požadovaný formát dat je JSON. Týká se jak vstupních, tak i výstupních dat.

2. CRUD operace

- API zajišťuje CRUD operace o objednávkách, produktech, zásilkách a většině tabulek na obrázku 1.1 dle specifikovaných požadavků.
- Po každé operaci by mělo být jasné, zda se operace zdařila, či ne.

3. Objednávky

- K tabulce `oc_order` se budou poskytovat i další přídatné informace získané z dalších tabulek či výpočtem rámci příkazu GET.
- Rovněž se poskytují seznamy způsobů dopravy a způsobů plateb s jejich příslušnými cenami. Poskytuje se i seznam všech možných stavů objednávky.

3. SPECIFIKACE POŽADAVKŮ

- S každou změnou ceny objednávky se v rámci JSON zasílá aktuální cena.

4. Frontendová část

- Jediným účelem, pro který bude využita frontendová část frameworku Laravel, je zaslání faktury na požadavek typu GET.

5. Business logika

- Z důvodu konvence a bezpečnosti bude většina business logiky součástí backendu nikoli frontendu i přes mírně negativní vliv na výkon.

6. Databáze

- API bere ohled na nedokonalosti databáze a zabraňuje uložení nesmyslných vstupů.
- Databáze plně podporuje všechny funkčnosti API.

7. Dokumentace

- API je zdokumentováno tak, aby se dalo zjistit, co přesně nabízí za služby a jakými způsoby lze tyto služby poskytovat.
- Dokumentace je ve formátu HTML.

8. Autentizace různých uživatelů

- Pro rozpoznání identity přihlašovaného, je nezbytná autentizace. S tím souvisí definice uživatelských rolí. Jelikož API bude využíváno i mimo administraci, nejedná se jen o administrátora.
- Uživatelskými rolemi jsou:
 - ★ Administrátor
 - ★ Nepřihlášený zákazník
 - ★ Přihlášený zákazník

9. Autorizace

- Aby si přihlášený uživatel nemohl dělat, co chce, každý úkon, který nemůže provádět libovolný člověk, bude autorizován.
- V případě pokusu o neautorizovaný přístup systém vrátí výjimku.

3.2 Nefunkční požadavky

1. Prostředí

- Požadovaným programovacím jazykem je PHP s využitím frameworku na něm postaveném.
- Využívá jen knihoven, které jsou volně dostupné.

2. Webová služba

- API je dostupné prostřednictvím REST příkazů zadaných v prohlížeči.

3. Frontendová část

- Frontendová část, kterou má na starosti kolegyně Iuliia Evseenko, bude samostatným projektem komunikujícím s backendovou.

4. Autorizace

- K autorizaci budou využívány sessions.

5. Výkon

- API zasahuje do databáze co nejméně.
- Pro všechna data se využívá technologie caching. Jako cache technologie se bude moci využít soubor i technologie Redis.

6. Rozšiřovatelnost a modulárnost

- API je rozděleno na oddělené vrstvy:
 - ★ První vrstva reprezentuje tabulky v databázi a poskytují se a mění se skrze ní data. Při změně názvu tabulky není třeba přepisovat všechny napsané přístupy do databáze.
 - ★ Druhá vrstva se stará o komunikaci s klientem a první vrstvou.
 - ★ V druhé vrstvě se nenachází hlavní část business logiky, k tomu slouží tzv. services.
 - ★ Třetí vrstva poskytuje fakturu.
- Každá třída by měla být zaměřena na jediný úkol, ale zároveň by měla minimalizovat využití ostatních tříd.
- Pokud je to možné, využít dědičnosti.
- Kód se nikde neopakuje.

Architektura aplikace

Jak jsme zjistili v analýze, nejvhodnějším přístupem pro naše API je Laravel s využitím technologie REST. Laravel staví na architektuře MVC.

4.1 MVC architektura

MVC architektura rozděluje aplikaci do 3 logických komponent:

1. Model

- „Jedná se o reprezentaci informací, se kterými systém pracuje, a tudíž řídí přístupy k těmto informacím, a také často implementuje přístupová oprávnění (...).“ [6]
- Ve frameworku Laravel se v modelu přístupové dotazy a dotazy a dotazy měnící databázi běžně nedefinují, pokud není stejný dotaz nevyužívá na mnoha místech. V opačném případě se může využít tzv. local Scopes, případně se musí vytvořit jiný soubor pro definování těchto dotazů. Někdy se využívá tzv. repository pattern, ten ale zvyšuje míru složitosti a jeho přínosy jsou pouze velmi potenciální.

2. View

- View, česky pohled, se zabývá logikou toho, co uživatel vidí - frontendovou částí.
- Frontend není náplní této práce, tedy ani view. Výjimkou je poskytnutí faktury.

3. Controller

- Controller, česky řadič, reaguje na uživatelské akce a pokud je potřeba, vyvolává požadavky na model. Taktéž zde dochází ke zpracování dat. [6]

- Získávají poslaná data přes třídu `Illuminate\Http\Request` skrze dependency injection.
- V této práci controllery nesou většinu logiky aplikace.

4.2 Router

Kromě modelů a controllerů, nezbytnou součástí aplikace je i router. Router se definuje v souboru `web.php` v záložce `resources`. Router se stará především o navigování požadavků, které uživatel zadal v prohlížeči, controllerům, které pro danou akci mají příslušné kompetence. Požadavky jsou zadávány prostřednictvím tzv. API endpointů, což jsou URL řetězce s určitou pevně danou strukturou. Takový endpoint může např. vypadat takhle: `www.stylka.cz/orders/35000`.

Požadavky jsou čtyř typů resp. pěti typů (uvedeno na příkladu objednávek):

1. GET (index)
 - Vrátí seznam všech objednávek.
 - API endpoint: `/orders`
 - Router směřuje požadavek na metodu řadiče defaultně nazvanou `index`.
2. GET (show)
 - Vrátí požadovanou objednávku.
 - API endpoint: `/orders/{id}`
 - Router směřuje požadavek na metodu řadiče defaultně nazvanou `show`.
3. POST
 - Do databáze uloží novou objednávku.
 - API endpoint: `/orders`
 - Router směřuje požadavek na metodu řadiče defaultně nazvanou `store`.
4. PUT
 - V databázi pozmění již uloženou objednávku .
 - API endpoint: `/orders/{id}`
 - Router směřuje požadavek na metodu řadiče defaultně nazvanou `update`.

5. DELETE

- V databázi smaže již uloženou objednávku .
- API endpoint: `/orders/{id}`
- Router směruje požadavek na metodu řadiče defaultně nazvanou `delete`.

V routerech se vždy k dané URL adrese musí ještě definovat controller, který se o daný požadavek postará. Mnohdy musí být definována i metoda samotného controlleru.

4.3 Autentizace

V Laravel 8 se o správnou autentizaci uživatelů stará dodaný balíček s názvem Fortify. Za předpokladu, že bychom k ukládání uživatelů využívali jen jednu tabulku, přičemž bychom jednotlivým užitelům jen přidělili roli, nebylo by potřeba nic řešit. Ale vzhledem k tomu, že se musíme přizpůsobit databázi, která pro uživatele různých rolí má různé tabulky, je potřeba doplnit defaultní kód, který jsme získali instalací balíčku Fortify. Abychom se tímto problémem mohli v praktické části zabývat, je třeba znát, co jsou to tzv. guard a tzv. provider.

„Prvky autentizace zvané guard definují, jak jsou uživatelé autentizováni pro každý požadavek. Laravel například může dodávat session guard, který zachovává stav za pomoci ukládání jednotlivých sessions a cookies.“ [10]

„Prvky autentizace zvané provider definují jak jsou uživatelé získávání z úložiště. Laravel například dodává podporu pro získávání uživatelů prostřednictvím struktury Eloquent a databázového query builderu.“ [10]

4.4 Autorizace

V Laravel se k autorizaci jednotlivých úkonů využívá tzv. gates a tzv. policies. Pro autorizaci, která se nepoutá k žádnému konkrétnímu modelu se využívá gates, zatímco policies, což jsou samostatné třídy, se starají autorizace úkonů spojených s určitým modelem.

4.5 Validace

Pro validaci, čili ošetření že zadaný vstup je správný, lze v Laravel taktéž využít více způsobů. V daném projektu se kontroluje jen vstup zadaný zákazníkem. Pro tento projekt jsem, stejně jak tomu bylo při autorizaci, vyčlenil

samostatnou třídu, která má kontrolu vstupu na starosti. V daném projektu se kontroluje jen vstup zadaný zákazníkem.

4.6 Testování

V záložce `tests` se k testování využívá záložek:

- `Unit`
- `Feature`

„Unit testy jsou testy, které se zaměřují na velmi malou, izolovou část kódu. Testy v „Unit“ testové záložce nebootují Laravel aplikace a tudíž nemůžou přistupovat k databázi aplikace ani jiné služby Laravel frameworku.“ [11]

„Feature testy můžou testovat větší část kódu včetně toho, jak mezi sebou jednotlivé objekty komunikují nebo dokonce celý HTTP požadavek na JSON endpoint. Obecně řečeno, většina testů by měla být typu feature. Tyto testy zajišťují nejspolehlivěji, že systém jako celek pracuje tak jak zamýšleno.“ [11]

Samotné testy jsou tzv. "white box", protože kód aplikace není tajný, sám ho budu psát. Nejsou to testy automatizované, postupuje se dle testovacích scénářů.

Laravel pro své testování využívá testovací framework PHPUnit, který se využívá pro testování jak unit tak feature testů.

Část II

Praktická část

Realizace

5.1 Model

V Laravel není explicitní definice nutná, ale aplikaci to zpřehlední, mírně zkrátí zápis určitých dotazů (např. metoda `find()`) a při změně jména tabulky se pak nemusí nemusí přepisovat databázové dotazy na více místech. Výhodou je rovněž možnost zavedení automatických razítek, např. `CREATED_AT`, což je případě naší databáze nazváno jako `date_added`.

```
class Product extends Model
{
    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'oc_product';

    /**
     * The primary key associated with the table.
     *
     * @var string
     */
    protected $primaryKey = 'product_id';

    /**
     * Indicates if the model should be timestamped.
     *
     * @var bool
     */
    public $timestamps = true;
```

```
const CREATED_AT = 'date_added';
const UPDATED_AT = 'date_modified';

/**
 * The attributes that aren't mass assignable.
 *
 * @var array
 */
protected $guarded = [];
}
```

Listing 5.1: Model pro tabulku oc_product

Jak vidno, obvykle se do modelu zapisuje název příslušné tabulky, identifikátor, zda chceme možnost časových razítek a atributy, kterým mohou být přiřazeny nové hodnoty, během hromadných akcí, například za použití metody `update()`. Laravel bohužel neřeší možnost složených klíčů, tudíž například pro tabulku `oc_product_gift` jsem model nevytvářel.

Modely, které se zároveň využívají pro autentizaci, dědí ze třídy `Authenticatable` a mohou využívat další funkce.

```
class User extends Authenticatable
{
    use HasApiTokens;
    use HasFactory;
    use HasProfilePhoto;
    use Notifiable;
    use TwoFactorAuthenticatable;

    ...
}
```

5.2 View

View je v tomto projektu reprezentováno pouze fakturou. Ta je psána v HTML pro tisk, respektive šablonovacím systému pro Laravel zvaný Blade. Fakturu získáme zavoláním URL požadavku, který zavolá funkci, která předá šabloně potřebná data a vrátí fakturu.

```
Route::get('/orders/{order}/invoice',
    function($order_id){
        ...
    }
);
```

```
return view('invoice', [
    'date' => $date,
    'date14' => $date14,
    'eshop' => $eshop,
    'order' => $order,
    'order_products' => $order_products,
    'same' => $same
],);
});
```

Listing 5.3: Route pro získání faktury

Blade umožňuje využívat různých direktiv, usnadňující psaní kódu pro frontend.

```
<tr>
    @if($order_product->is_transfer == 0)
        <td>{{$order_product->name}}</td>
        <td>{{$order_product->model}}</td>
        <td align="right">
            {{$order_product->warranty}}</td>
    @else
        <td colspan="2">{{$order_product->name}}</td>
        <td align="right">-</td>
    @endif
</tr>
```

Listing 5.4: Výstřížek z faktury

5.3 Controller

Controller skrze router přijme požadavek a zpracuje ho příslušnou metodou. Pokud se v controlleru zjistí nějaká nesrovnalost (např. uživatel nemá příslušné oprávnění), pošle uživateli příslušnou výjimku. Pakliže se vykoná celý kód dané metody, pošle příslušnou návratovou hodnotu, který se odvíjí od typu požadavku, tedy typu metody (v závorce):

1. GET (index)
 - Vrátí seznam všech objektů.
2. GET (show)
 - Vrátí požadovaný objekt.
3. POST (store)

- Vrátil idenitifikátor (id) nově vytvořeného objektu, nebo nově vytvořených objektů. Ne nutně všech, protože by to nemělo žádný význam.

4. PUT (update)

- Vrátil true nebo false pro jednotlivé atributy, které měly být aktualizovány, asociativní pole, kde každému atributu je přiřazen true, pokud byl úspěšně změněn, nebo false v opačném případě.

5. DELETE (destroy)

- Vrátil true pokud tabulka a všechny navazující tabulky byly z databáze úspěšně vymazány. Pokud se jednu tabulku vymazat nezdařilo, vrátí false.

5.3.1 OrderController

Pro ukázkou controlleru jsem zvolil OrderController, což je controller pro objednávku. Je to jeden z mnoha controllerů. Informace v této sekci se do jisté míry týkají i dalších controllerů.

5.3.1.1 Update

Abychom pro každý atribut nemuseli mít zvláštní endpoint, využívá se rozpoznávání, zda daný atribut se nachází v daném JSON.

```
if ($request->has('email')) {  
    $ret_array += array('email' => $order->update([  
        'email' => $request->input('email'),  
        'date_modified' => date("Y-m-d H:i:s")  
    ]));  
}
```

Listing 5.5: Metoda update() s jedním atributem

5.3.1.2 Store

Pro uložení objednávky se nejdříve uloží nový záznam do tabulky `oc_order`. Pokud se jedná o registrovaného zákazníka, uloží se záznamy, které jsou již v databázi, do příslušné objednávky. S tím souvisí uložení objednaného produktu do tabulky `oc_order_product`. Následně je potřeba upravit atributy týkající skladu, tedy kolik kusů objednaných produktů pochází z interního skladu (`quantity_int`), z externího skladu (`quantity_ext`), dále kolik produktů daného typu zbývá na interním skladě (`internal_quantity`) a jak je to

s produkty z externích skladů (`quantity`). Na závěr je třeba dopočítat celkovou cenu objednávky v rámci tabulky `oc_order_total`. Nakonec vrátí JSON s identifikátory a cenu:

```
return response()->json([
    'order_id' => $oid,
    'order_product_id' =>
        $order_total['order_product_id'],
    'noTaxTotal' => $order_total['noTaxTotal'],
    'tax' => $order_total['tax'],
    'total' => $order_total['total']
]);
```

Listing 5.6: JSON s identifikátory

5.3.1.3 Delete

Když pak chceme vymazat nějakou objednávku, nestačí smazat jen řádek z tabulky `oc_order`. Je potřeba smazat i tabulky, které jsou na daném záznamu existenčně závislé. Z důvodu absence cizích klíčů nelze využít kaskádovitěho mazání.

5.4 Service

Třídám, jejichž náplní není práce controllerů, ale pouze výpočty a business logika, se v Laravel říká Services a je jim vyhrazena samostatná složka **Services**.

5.5 Router

URL cesty se v routeru dají zapsat více způsoby. Chceme-li využít více typů požadavků vztahující se k jednomu modelu, využívá se tzv. `apiResources`, který namapuje příslušné požadavky na defaultní metody (`show`, `update`,...) controlleru.

```
Route::apiResources([
    'orders' =>
        API\OrderController::class,
    'orders.packages' =>
        API\PackageController::class,
    'orders.products' =>
        API\Order_productController::class,
```

```
'products' =>
    API\ProductController::class
]);
```

Listing 5.7: ApiResources

Pro ostatní URL cesty se musí pro cestu definovat typ požadavku, controller a metoda controlleru.

```
Route::get('/orders/{order}/price',
    [OrderController::class, 'getPrice']);
```

Listing 5.8: Route pro požadavek typu GET

5.6 Dokumentace

Pro dokumentaci tohoto projektu se využívá balíček **knuckleswtf/scribe**. Díky tomuto balíčku je možné aktuální dokumentaci vygenerovat za použití příkazu `php artisan scribe:generate`. Ten vygeneruje dokumentaci z toho, co jsme napsali do souboru `web.php`, kde se definují routes, a příslušných komentářů nad controllery, kde se definují záležitosti jako parametry přicházející requestu, url parametry, odpovědi a další záležitosti. Výsledná dokumentace se pak nachází v souboru `index.html` ve složce `public`.

5.7 Databáze

Databáze pro účely tohoto projektu byla databáze obohacena o tabulku `sessions` sloužící k autorizaci přicházejících požadavků. Do tabulky `oc_order` byl přidán atribut `viewed`, pro zaznamenání, zda už se na objednávku někdo podíval. Žádné další změny nebyly nutné.

5.8 Autentizace

Jak již bylo zmíněno v teoretické části, je potřeba připravit projekt na fakt, že pro přihlašované se budou v databázi využívat dvě tabulky. Tuto skutečnost je potřeba zmínit `auth.php` v záložce `config`. Zde se nastaví jednotlivé guards a providers. Pro odlišení toho, kdo se zrovna přihlašuje se využívá subdoména `admin`, která se vkládá do URL před hlavní doménu, např. `www.admin.stylka.cz`. Defaultní guard je `web`, který se stará o autentizaci zákazníků. O autentizaci administrátorů, kteří jsou v databázi v tabulce `oc_user`, se bude starat ten stejný guard, až na to, že je nezbytné za běhu v třídě `FortifyServiceProvider` v metodě `register()` změnit provider daného guard z `customers` na `admins`. Pro zajištění stavu se v obou případech využívá tabulky `sessions`, kterých se pak využívá při autorizaci.


```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'customers'
    ],
]
```

Listing 5.9: Defaultní guard při přihlašování zákazníků

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'customers'
    ],
]
```

Listing 5.10: Defaultní guard při přihlašování administrátorů po změně

```
public function register()
{
    if (request()->isAdmin()) {
        config(['fortify.domain' => adminUrl()]);
        config(['auth.guards.web.provider'
            => 'admins']);
    }
}
```

Listing 5.11: Funkce register() třídy FortifyServiceProvider

V definici pro providers definujeme k jakému modelu se provider vztahuje. Využíváme eloquent, který nám říká, že se jedná o modely, protože eloquent je implementace databáze v rámci frameworku Laravel.

```
'providers' => [
    'admins' => [
        'driver' => 'eloquent',
        'model' => Admin::class,
    ],
    'customers' => [
        'driver' => 'eloquent',
        'model' => Customer::class,
    ]
]
```

Listing 5.12: Definice providers

5.9 Autorizace

V tomto projektu se využívá pouze policies, protože všechny autorizační úkony jsou spjaté s nějakou třídou. Tyto třídy jsou ve složce Policies. Většinu požadavků na autorizaci zvládá funkce `authorize()` třídy `Policy.php`. Ta zjišťuje, jestli administrátor má požadované oprávnění a zákazník, pokud požadovaný úkon zákazníci vůbec mohou provádět, ip adresu zapsanou v session shodnou s ip adresou zapsanou v objednávce. A pokud je to akce jen pro přihlášeného zákazníka, tak se zjišťuje, jestli v tabulce `oc_customer` je ip adresa shodná s ip adresou session.

```
class Policy
{
    use HandlesAuthorization;

    public function authorize
        ($user, $needle, $action, Order $order = null,
        Customer $customer = null)
    {
        if($user instanceof User)
        {
            $permission =
                unserialize(DB::table('oc_user_group')
                    ->where('user_group_id'
                        , $user->user_group_id)
                    ->value('permission'));
            if ($permission === null) return false;
            $key = array_search
                ($needle, $permission[$action]);
            if($key === false) return false;
            else return true;
        }
        else if($order != null)
        {
            if(session('ip_address') === $order->ip)
                return true;
            else return false;
        }
        else if($user instanceof Customer)
        {
            if($user->customer_id ===
                $customer->customer_id)
                return true;
            else return false;
        }
    }
}
```

```
        }  
        else return false;  
    }  
}
```

Listing 5.13: Autorizace uživatelů

Z této třídy pak dědí policie vztahujících se k nějaké konkrétní třídě, např. `OrderPolicy.php`.

```
public function updateByAdminOrCustomer  
    ($user = null, Order $order)  
{  
    return $this->authorize  
        ($user, 'sale/order', 'modify', $order);  
}
```

Listing 5.14: Příklad funkce z třídy `OrderPolicy.php`

Když pak chceme zjistit, jestli daný uživatel má skutečně oprávnění, lze to udělat mnoha způsoby. V tomto projektu je využívána funkce `authorize()` (odlišná od dříve zmíněné), která se volá před kódem, který vyžaduje oprávnění. V případě, že uživatel oprávnění má, program pokračuje. V opačném případě vyvolá program výjimku značící neautorizovaný přístup.

```
public function update(Request $request, Order $order)  
{  
    $this->authorize('updateByAdminOrCustomer', $order);  
    $ret_array = [];  
  
    ...  
}
```

Listing 5.15: Příklad funkce `authorize()` pro autorizaci úkonu aktualizace konkrétní objednávky

5.10 Validace

K validaci vstupů se využívá převážně tříd zvaných `Form Request`, které se nachází ve složce `Requests`. V tomto projektu je to například třída `OrderUpdate.php`, která má tři základní metody:

1. authorize()

- Stará se autorizaci. Protože autorizace je řešena již na jiném místě, vrací vždy `true`, což zajišťuje nevměšování této třídy do práce jiných tříd.

2. rules()

- Definuje pravidla kterými musí dané vstupy projít. Pokud tato pravidla nejsou splněna, vrátí se výjimka.

3. messages()

- Definuje zprávy, které se s výjimkou vrací, když nějaké pravidlo nebylo splněno.

```
public function rules()
{
    return [
        'firstname' =>
            'regex:/^[\\s\\-\\p{L}]*$/u',
        'email' => "
            'email',
        'telephone' =>
            'regex:/^\\+?[\\d\\s]{9,15}$/',

        ...
    ]
}
```

Listing 5.16: Výstřižek funkce `rules()`

```
public function messages()
{
    return [
        'firstname.regex' =>
            'Neplatné jméno.',
        'email.email' =>
            'Neplatná emailová adresa.',
        'telephone.regex' =>
            'Neplatné telefonní číslo.',

        ...
    ]
}
```

Listing 5.17: Výstřižek funkce `messages()`

Dále se využívá „ručně vytvořených“ validátorů.

```
Validator::make($input,
[
    'password' => $this->passwordRules(),
    'firstname' => 'regex:/^[s\-\p{L}]*$/u',
    'email' => 'email'
],
[
    'firstname' => "Neplatné jméno.",
    'email' => "Neplatný email."
])->validate();
```

Listing 5.18: „Ručně vytvořený“ validátor ve třídě CreateUser

Testování

Aby se zamezilo odhalení chyb až při provozu aplikace, je nutné aplikaci důkladně otestovat. Testování projektu probíhá ve dvou souborech –

`TestDomainSetupValue.php` a `TestEshop.php`, v obou případech se jedná o feature testy. Zásadním problémem unit testů je, že se s jejich prostřednictvím nedá připojovat k databázi, a protože skoro všechny metody databázi v nějaké své fázi potřebují, nelze využít unit testů. To ale není problém, akorát soubory s feature testy jsou o to objemnější.

6.1 TestDomain_setupValue

Funkce má za úkol z daného řetězce vykódovat cenu dopravy pro příslušný řetězec a příslušnou objednávku. Příslušný řetězec se získává z tabulky `oc_domain_setup`.

```
public function testDomain_setupValue_1()
{
    $this->assertEquals(OrderController::
        domain_setupValue("f:0,600:49,:0", 1, 30), 0);
    $this->assertEquals(OrderController::
        domain_setupValue("f:0,600:49,:0", 1, 700), 0);
    $this->assertEquals(OrderController::
        domain_setupValue("f:0,600:49,:0", 0, 500), 49);
    $this->assertEquals(OrderController::
        domain_setupValue("f:0,600:49,:0", 0, 700), 0);
}
```

6.2 TestEshop

V souboru `TestEshop.php` se nachází takřka všechno testování aplikace. Z důvodu nezávislosti na databázi se zde vytváří vlastní produkty, objednávky, objednané produkty, zásilky, aj. za pomoci `store` metod, což jsou požadavky typu PUT. Následně se zkouší všechny potřebné požadavky typu GET a POST a nakonec se zavolají požadavky typu DELETE, kdy se všechny vytvořené tabulky smažou a vzdálená testovací databáze tak zůstává netknutá.

Ještě předtím než se začnou vykonávat vlastní testy, spustí se před každým testem funkce `setUp()`, která nastaví proměnné, které se ve většině testů budou využívat.

```
public $oid, $user;
protected function setUp() : void
{
    parent::setUp();
    $this->user = User::find(1);
    $this->oid = Order::max('order_id');
}
```

Listing 6.2: Funkce `setUp()`

Jelikož po smazání jednoho řádku tabulky odpovídající identifikátor už nebude využíván, takže nelze využít funkce `max()`, hodí se funkce `getNextId()`, která zjistí a předá identifikátor následující ukládané tabulky.

```
public function getNextId($table)
{
    $statement = DB::select("SHOW TABLE STATUS LIKE "
        . '\'' . $table . '\'' );
    return $statement[0]->Auto_increment;
}
```

Listing 6.3: Funkce `getNextId()`

Pro jednotlivé typy požadavků (GET, POST,...) přísluší funkce se stejným názvem, ke kterým se doplňuje název `Json`, pakliže se při požadavku posílá JSON. V obou případech se funkci dává příslušné URI. Pokud se posílá JSON, předá se funkci JSON. Výsledkem jsou tedy funkce `$this->get()`, `$this->putJson()` aj.

Pro otestování autorizace se využívá funkcí `$this->actingAs()`, které se předá příslušný přihlášený uživatel a `$this->withSession()`, které se předají atributy session potřebné pro autorizaci.

Předešle zmíněné funkce se napojí na sebe a návratová hodnota se chytá v proměnné `$response`.

6.2.1 Správné požadavky

Pomocí funkce `$response->assertStatus()` se z `$response` zjišťuje stavový kód, který je 200, pokud vše proběhlo tak jak má. Dále se zjišťuje, zda odpovídá poslaný JSON tomu očekávanému za pomoci funkce `$response->assertJson()`.

Pro požadavky, které databázi modifikují, se dále musí zjistit, zdali se daný řádek tabulky v databázi nachází, případně naopak nenachází. Pro POST a PUT požadavky se tedy se tedy využije funkce `$this->assertDatabaseHas()` a pro požadavky typu DELETE se využije funkce `$this->assertDatabaseMissing()`.

V následující sekci budou uvedeny příklady testů pro jednotlivé požadavky s různorodým využitím předešle zmíněných funkcí.

6.2.1.1 POST

```
$pid = $this->getNextId('oc_product');
$response = $this->actingAs($this->user)
    ->postJson('/products',
        [
            "category_id" => 202,
            "category_id2" => 312,
            ...
        ]
    );
$response->assertStatus(200)
    ->assertJson(['product_id'=>$pid]);
$this->assertDatabaseHas('oc_product',
    [
        "product_id" => $pid,
        "category_id" => 202,
        ...
    ]
)
...
```

Listing 6.4: Část funkce `testStoreProduct1()`

6.2.1.2 PUT

```
public function testPutLanguage()
{
    $response = $this
        ->withSession(['ip_address' => '90.179.92.144'])
        ->putJson('/orders/' . $this->oid,
            [
                'language' => 'Cestina'
            ]
        );
}
```

```
    ]);
$response->assertStatus(200)
    ->assertJson(['language' => 'true']);
$this->assertDatabaseHas('oc_order',
    [
        'order_id' => $this->oid,
        'language_id' => 5
    ]);
}
```

Listing 6.5: Funkce testPutLanguage()

6.2.1.3 GET

Místo `$response->assertJson()` použijeme pro otestování funkci `$response->assertJsonCount()`, která spočítá počet poslaných JSON. To, že jsou posílány správné formáty JSON jako takové, je otestováno na jiném místě.

```
public function testIndexOrders()
{
    $count = Order::count();
    $response = $this->actingAs($this->user)
        ->get('/orders');
    $response->assertStatus(200)
        ->assertJsonCount($count);
}
```

Listing 6.6: Funkce testIndexOrders

6.2.1.4 DELETE

```
public function testDeleteOrder()
{
    $response = $this->actingAs($this->user)
        ->delete('/orders/'. $this->oid);
    $response->assertStatus(200);
    $this->assertDatabaseMissing(
        ('oc_order',
        ['order_id' => $this->oid])
    ->assertDatabaseMissing(
        ('oc_order_history',
        ['order_id' => $this->oid])
    ->assertDatabaseMissing(
        ('oc_order_product',
        ['order_id' => $this->oid])
    );
}
```

```
->assertDatabaseMissing
    ('oc_order_product_move',
     ['order_id' => $this->oid])
->assertDatabaseMissing
    ('oc_order_total',
     ['order_id' => $this->oid]);
}
```

Listing 6.7: Funkce `testDeleteOrder`

6.2.2 Neprávné požadavky

Pomocí funkce `$response->assertStatus()` se z `$response` zjišťuje stavový kód, který je závislý na konkrétním případě.

6.2.2.1 Požadavek od uživatele pozbývajícího oprávnění

```
public function
testUnauthorizedAccessByAdminOrCustomer()
{
    $response = $this->withSession
        (['ip_address' => '165.154.210.246'])
        ->get('/orders/'. $this->oid);
    $response->assertStatus(403);
}
```

Listing 6.8: Test na zamítnutí požadavku z důvodu nedostatečného oprávnění

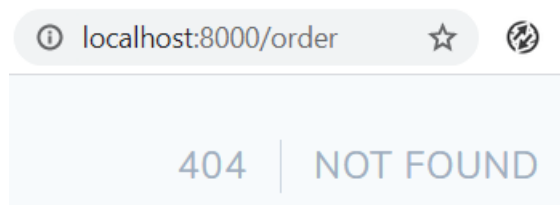
6.2.2.2 Požadavek s nesprávně zadaným vstupem

```
public function testPutWrongFirstname()
{
    $response = $this->actingAs($this->user)
        ;->putJson('/orders/' . $this->oid,
        [
            'firstname' => 'Jan. \\$'
        ]);
    $response->assertStatus(422)
        ->assertJsonValidationErrors
            (['firstname' => 'Neplatné jméno.']);
}
```

Listing 6.9: Test na zamítnutí požadavku z důvodu špatného vstupu

6.2.2.3 Požadavek se špatně zadanou URL

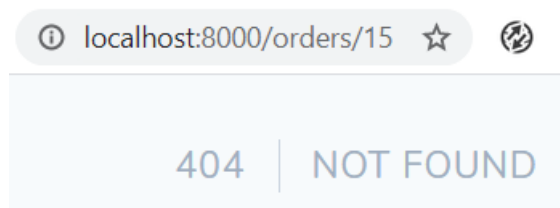
Odpověď na takový požadavek, je možno vidět na obrázku 6.1.



Obrázek 6.1: Požadavek se špatně zadanou URL

6.2.2.4 Požadavek na neexistující objekt

Odpověď na takový požadavek, je možno vidět na obrázku 6.2.



Obrázek 6.2: Požadavek na neexistující objekt

Závěr

Hlavním cílem této bakalářské práce bylo vytvoření funkčního API pro eshop Stylka.cz a tento cíl se naplnit podařilo. Nejenom to, API obsluhuje i další eshopy, které využívají stejnou databázi, tudíž tento cíl byl splněn nad očekávání. Odpovídá veškerým vzneseným požadavkům a náležitostem běžného eshopu, což bylo taktéž součástí našeho cíle. Jelikož kolegyně Iuliia Evseenko chce svou práci dokončit později, než já svou, tak z její strany byla velmi patrná absence většího množství požadavků na tuto práci. Do budoucna by tedy bylo vhodné více přizpůsobit toto API jejím zájmům, což by neměl být zásadní problém. V tomto ohledu jsem jí nabídl osobní asistenci, protože pro mě bude realizace případných úprav jednodušší než pro ni.

Jak je uvedeno v kapitole 3, požadavky jsme si rozdělili na funkční a nefunkční. Co se týče funkčních požadavků, naše API dává data ve formátu JSON. Poskytuje vše, co provoz eshopu obnáší, umožňuje měnit ta data, která jsou pro chod eshopu nezbytná. Databáze byla obohacena o prvky potřebné pro chod API, API je do detailů zdokumentováno. Pro zabránění nežádoucích operací je aplikace plně autentizována a autorizována pomocí balíčku Fortify. Nefunkční požadavky, jako nároky na technologie, jsou taktéž splněny, protože API je napsáno v PHP, je dostupné skrze REST, využívá sessions k autorizaci, využívá technologii caching a API je rozděleno na vrstvy umožňující snadnou rozšiřovatelnost a modulárnost.

Nezbytnou součástí řešení měly být i soubory s testy. Testy pokrývají velké množství funkcí aplikace, zabírají se především business logikou aplikace a CRUD operacemi na ně navázanými. Některé CRUD operace testy pokryty nebyly, jelikož se do budoucna počítá se jejich úpravou. Můžeme tedy prohlásit, že všechny vytyčené cíle této práce splňuje.

Literatura

- [1] OpenCart. In: *Wikipedia: The Free Encyclopedia* [online]. Wikimedia Foundation, 2003. Stránka naposledy edit. 19. 5. 2020 v 22:19. [vid. 2020-7-21]. Anglická verze. Dostupné z: <https://en.wikipedia.org/wiki/OpenCart>
- [2] Interfaz de programación de aplicaciones. In: *Wikipedia: La enciclopedia libre* [online]. Wikimedia Foundation, 2003. Stránka naposledy edit. 24. 6. 2020 v 05:25. [vid. 2020-7-21]. Španělská verze. Dostupné z: https://es.wikipedia.org/wiki/Interfaz_de_programacion_de_aplicaciones
- [3] Application programming interface. In: *Wikipedia: The Free Encyclopedia* [online]. Wikimedia Foundation, 2003. Stránka naposledy edit. 31. 7. 2020 v 03:24. [vid. 2020-7-31]. Anglická verze. Dostupné z: https://en.wikipedia.org/wiki/Application_programming_interface
- [4] Simple Object Access Protocol. In: *Wikipedia: La enciclopedia libre* [online]. Wikimedia Foundation, 2003. Stránka naposledy edit. 16. 4. 2020 v 23:15. [vid. 2020-7-1]. Španělská verze. Dostupné z: https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol
- [5] Representational state transfer. In: *Wikipedia: The Free Encyclopedia* [online]. Wikimedia Foundation, 2003. Stránka naposledy edit. 29. 7. 2020 v 21:44. [vid. 2020-7-30]. Anglická verze. Dostupné z: https://en.wikipedia.org/wiki/Representational_state_transfer
- [6] SOAP vs. REST: The Differences and Benefits Between the Two Widely-Used Web Service Communication Protocols. In: *Stackify* [online]. Matt Watson, 2017. [vid. 2020-7-30]. Dostupné z: <https://stackify.com/soap-vs-rest/>
- [7] Modelo-vista-controlador In: *Wikipedia: La enciclopedia libre* [online]. Wikimedia Foundation, 2003. Stránka naposledy edit. 30. 6. 2020 v 10:12.

- [vid. 2020-7-30]. Španělská verze. Dostupné z: <https://es.wikipedia.org/wiki/Modelo-vista-controlador>
- [8] Úvod do JSON In: *JSON.org* [online]. JSON.org, 2002. [vid. 2020-8-1]. Dostupné z: <https://www.json.org/json-cz.html>
- [9] In: *JSON.org* [online]. JSON.org, 2002. [vid. 2020-8-1]. Dostupné z: <https://www.json.org/json-cz.html>
- [10] Authentication In: *Laravel* [online]. Taylor Otwell, 2011. [vid. 2020-11-10]. Dostupné z: <https://laravel.com/docs/5.7/authentication>
- [11] Authentication In: *Laravel* [online]. Taylor Otwell, 2011. [vid. 2021-01-04]. Dostupné z: <https://laravel.com/docs/8.x/testing>
- [12] PHP Laravel VS Symfony: A Detailed Comparison of Web Development Frameworks. In: *Hackernoon* [online]. David Smooke, 2013. [vid. 2021-01-05]. Dostupné z: <https://hackernoon.com/php-laravel-vs-symfony-a-detailed-comparison-of-web-development-frameworks-sq493y3l>
- [13] In: *Hackernoon* [online]. David Smooke, 2013. [vid. 2021-01-05]. Dostupné z: <https://hackernoon.com/php-laravel-vs-symfony-a-detailed-comparison-of-web-development-frameworks-sq493y3l>

Seznam použitých zkratk

API Application programming interface

XML Extensible markup language

JSON JavaScript Object Notation

REST Representational state transfer

MVC Model–view–controller

Obsah přiloženého CD

readme.txt	stručný popis obsahu CD
└─ src	
└─ impl	zdrojové kódy implementace
└─ text	text práce
└─ thesis.pdf	text práce ve formátu PDF