Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

B2B řešení pro správu skladových zásob za užití REST API

Bc. Jan Jahoda

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Jahoda

Informatika

Název práce

B2B řešení pro správu skladových zásob za užití REST API

Název anglicky

B2B solution for managing stock supplies using REST API

Cíle práce

Práce je zaměřena na přípravu a zhotovení serverové aplikace využívající REST rozhraní, která bude sloužit k B2B komunikaci mezi centrálním dodavatelem a jednotlivými obchodníky. Cílem diplomové práce je popsat problematiku rozhraní REST a s využitím vybraného programovacího jazyka a dalších technologií implementovat výše zmíněný systém.

Metodika

Metodika zpracování teoretické části diplomové práce je založena na studiu odborných informačních zdrojů zaměřených na problematiku REST API. S využitím syntézy zjištěných poznatků budou formulována teoretická východiska.

Praktická část práce bude sestávat z kompletního návrhu a implementace serverové části aplikace vystavující REST rozhraní sloužící k B2B komunikaci mezi centrálním dodavatelem a obchodníky. Dále bude implementována ukázková aplikace využívající toto rozhraní a demonstrující jeho schopnosti.

Při zpracování praktické části práce bude využit programovací jazyk PHP a standardní nástroje a techniky softwarového inženýrství.

Doporučený rozsah práce

60-80 stran

Klíčová slova

rest api, webová aplikace, e-shop, user-interface

Doporučené zdroje informací

BRINZAREA, Bogdan, Cristian DARIE a Audra HENDRIX. AJAX and PHP: building modern web applications. 2nd ed. Birmingham, U.K.: Packt Pub., 2009. From technologies to solutions.

HENDERSON, Cal. Building scalable web sites. Sebastopol, CA: O'Reilly, c2006. ISBN 9780596102357.

JIM WEBBER, Savas Parastatidis and Ian Robinson. REST in practice. Farnham: O'Reilly, 2010. ISBN 9780596805821.

MASSÉ, Mark. REST API design rulebook. Sebastopol, CA: O'Reilly, 2012. ISBN 9781449310509.

RICHARDSON, Leonard a Michael AMUNDSEN. RESTful Web APIs. Beijing: O'Reilly, 2013. ISBN 9781449358068.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 27. 03. 2018

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "B2B řešení pro správu skladových zásob za užití REST API" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení této diplomové práce a ochotnou spolupráci při její tvorbě. Dále bych rád poděkoval každému, kdo poskytl svou pomoc při vypracování této práce.

B2B řešení pro správu skladových zásob za užití REST API

Abstrakt

Tato diplomová práce se zaměřuje na téma restových API. V teoretické části této práce jsou probrány veškeré informace týkající se REST API. Zároveň se zde pojednává o HTTP, které je nedílnou součástí technologie REST, a i proto jsou zde do detailu popsané jednotlivé požadavkové metody a další záležitosti týkající se tohoto internetového protokolu. V závěru teoretické části jsou probrané využité nástroje potřebné k vytvoření vlastní restové API, mezi které patří skriptovací jazyk PHP, framework nette použitý pro větší bezpečnost a pohodlný vývoj aplikace a MySQL technologie pro komunikaci s databází.

V praktické části diplomové práce je popsán celkový vývoj B2B řešení pro správu skladových zásob obuvi za užití REST API a českého PHP frameworku nette. Jedná se o navenek jednoduchou webovou aplikaci, která ale umí všechny základní funkce potřebné ke správě produktů, objednávek a klientů. Přidaní klienti jsou poté, po integraci do systému, schopni využívat poskytnutých REST API funkcí, díky kterým mohou číst informace o produktech a objednávat, či je vracet zpátky do centrálního skladu obuvi. Součástí praktické části je i ukázka vývoje menšího eshopu (klienta), který využívá právě těchto REST API funkcí, a který díky této B2B komunikaci plní své sklady obuvi. Konec práce obsahuje závěrečné shrnutí a návrh zlepšení vytvořeného prototypu REST API rozhraní.

Klíčová slova: návrh REST API, HTTP, PHP, webová aplikace, Nette

B2B solution for stock management using REST API

Abstract

This diploma thesis focuses on the topic of REST APIs. In the theoretical part of this work all informations regarding the REST API are described. At the same time, HTTP, which is an integral part of REST technology, is also discussed here, and therefore even individual HTTP request methods are described here in detail with other issues related to this Internet Protocol. At the end of the theoretical part, the tools used to create REST APIs, including PHP scripting language, the nette framework used for greater security and comfortable application development and MySQL database communication technology, are described.

In the practical part of the diploma thesis the overall development of a B2B solution for stock management of footwear, using REST API and Czech PHP framework nette, is described. This web application can do all basic functions needed to manage products, orders, and clients. Added clients are then able to use the provided REST API functions after they are integrated into the system, allowing them to read the product information and order or return products into the central footwear store. Part of the practical part also shows the development of a smaller a eshop (client) that uses these REST API functions, and thanks to this B2B communication fills its footwear store. The end of the work contains a final summary.

Keywords: REST API design, HTTP, PHP, web application, Nette

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní internetové době využívá stále víc a víc různých webových aplikací technologii REST API, jenž umožňuje snadnou a zabezpečenou komunikaci za pomoci jednoduchých HTTP volání. Díky této technologii vzniká mnoho různorodých aplikací nabízejícím svým klientům jednoduchý přístup k datům a službám za užití programovacích jazyků podporujících metody umožňujícím HTTP volání. Při správně vytvořeném HTTP požadavku umožňuje například REST API vytažení informací z Facebooku, či Google map nebo dokáže odeslat SMS zprávu při využití API sloužící k tomuto účelu. Ke všem těmto činnostem přitom stačí mít pouze, na rozdíl od konkurenční SOAP technologie, obyčejný PHP server (s ideálně nejnovější verzí PHP) a umět základy tohoto skriptovacího jazyka, neboť jednoduchý HTTP požadavek může být teoreticky i příkaz o jednom řádku.

V teoretické části diplomové práce je tato technologie detailně popsána společně s HTTP protokolem a jeho metodami, které s REST API úzce souvisí. Dále jsou zde uvedená důležitá pravidla potřebná při návrhu tohoto rozhraní. V závěru této části jsou vypsány jednotlivé programovací nástroje využité při tvorbě restového API. V praktické části je popsán vývoj naprogramování B2B řešení pro správu skladových zásob v jazyce PHP, jenž umožňuje klientům za využití REST API rozhraní číst, objednávat a vrátit produkty. Kromě toho je zde popsán i vývoj administrace menšího eshopu, který tyto všechny vytvořené REST API metody využívá.

Toto téma bylo zvoleno z důvodu velkého rozmachu této technologie, kdy většina současných API využívajících vzdálenou komunikaci fungující právě na REST rozhraní, a zároveň dokázání nenáročnosti naprogramování středně velké webové aplikace, která s tímto prostředím pracuje a dorozumívá se přes něj.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je zaměřena na přípravu a zhotovení serverové aplikace využívající REST rozhraní, která bude sloužit k B2B komunikaci mezi centrálním dodavatelem obuvi a jednotlivými obchodníky. Cílem diplomové práce je popsat problematiku rozhraní REST a s využitím vybraného programovacího jazyka a dalších technologií implementovat výše zmíněný systém.

2.2 Metodika

Metodika zpracování teoretické části diplomové práce je založena na studiu odborných informačních zdrojů zaměřených na problematiku REST API. S využitím syntézy zjištěných poznatků budou formulována teoretická východiska.

Praktická část práce bude sestávat z kompletního návrhu a implementace serverové části aplikace vystavující REST rozhraní sloužící k B2B komunikaci mezi centrálním dodavatelem a obchodníky. Součástí této aplikace bude zabezpečené přihlašování do administrace a kompletní správa produktů, klientů a objednávek. Dále bude implementována ukázková aplikace využívající toto rozhraní a demonstrující jeho schopnosti mezi které bude patřit objednávání produktů od centrálního dodavatele, procházení produktů a jejich vrácení zpátky centrálnímu dodavateli.

Při zpracování praktické části práce bude využit server-side framework Nette fungující na programovacím jazyku PHP, multiplatformní skriptovací jazyk JavaScript, jazyk pro práci s relačními databázemi SQL a další standardní nástroje a techniky softwarového inženýrství.

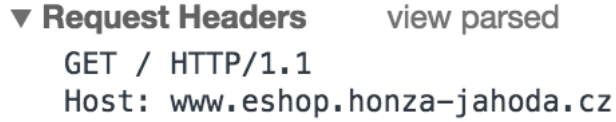
3 Teoretická východiska

3.1 WWW

Světová rozsáhla síť (neboli „WWW“, či zkráceně „web“) je informační prostor, kde jsou dokumenty a jiné webové zdroje identifikovány pomocí webové adresy URL propojenými přes hypertextové odkazy, jenž mohou být přístupné přes internet. Web, je často zaměňován s internetem. Ačkoli jsou obě tyto technologie společně propojené, jsou to různé věci. Internet je, jak již název napovídá, rozsáhlá globální síť, která zahrnuje množství menších sítí. Internet jako takový spočívá v podpoře infrastruktury a dalších technologií. Naopak, web je komunikační model, který prostřednictvím hypertextového přenosového protokolu (HTTP) umožňuje výměnu informací přes internet.

Web se skládá ze stránek, které jsou přístupné pomocí webového prohlížeče. Při vstupu na webovou stránku prohlížeč rozloží serverový název URL adresy (například www.stranka.cz) do adresy internetového protokolu (IP adresy) pomocí globálně distribuovaného doménového systému (DNS). Tento vyhledávací dotaz vrátí IP adresu (například ve tvaru 46.28.106.82). Prohlížeč poté požádá o zdroj zasláním HTTP požadavku přes internet na počítač na této adrese. Ten požaduje službu od konkrétního čísla portu TCP, takže přijímací hostitel může rozlišit HTTP požadavek od ostatních síťových protokolů, které mohou být obsluhovány. Protokol HTTP obvykle používá jako číslo portu 80. Obsah HTTP požadavku obsahuje mimo jiné následující dva řádky textu, které informují o použité požadavkové metodě GET a stránce, na kterou se přistupuje:

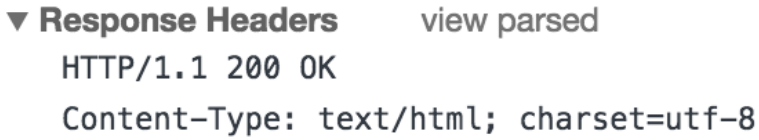
Obrázek 1 - Hlavičky HTTP požadavku



Zdroj: Vlastní zpracování

Počítač, který přijímá HTTP požadavek, tento požadavek přenáší do softwaru webového serveru, odposlouchávajícím požadavky na portu číslo 80. Pokud webový server splní požadavek, zašle HTTP odpověď zpět do prohlížeče se stavovým kódem označujícím úspěch a výsledným formátem zdroje.

Obrázek 2 - Hlavičky HTTP odpovědi



Zdroj: Vlastní zpracování

V těle odpovědi, se pak nachází obsah dané stránky ve formátu specifikovaném v hlavičce „content-type“. Webový prohlížeč parsuje HTML a interpretuje tagy (například tag <p> pro odstavce, či <title> pro titulek stránky) do správných textových formátů. Mnoho webových stránek používá HTML k zobrazování dalších zdrojů, jako jsou třeba obrázky, jiná vložená média, skripty ovlivňující chování stránky a kaskádové styly, jenž ovlivňují vzhled stránky. Prohlížeč proto provádí další HTTP požadavky na webový server pro všechny tyto ostatní typy médií. Vzhledem k tomu, že se tento obsah získává z webového serveru, je stránka v prohlížeči postupně vykreslována dle informací specifikovanými v HTML stránky. [1], [2]

3.1.1 HTTP hlavičky

HTTP hlavičky umožňují klientovi a serveru předat doplňující informace v odpovědi, či požadavku. Mezi tyto informace patří například stavový kód, název serveru, použitá požadavková metoda, prohlížeč klienta, či hlavička „cache-control“, která definuje zacházení s mezipamětí na webu. Hlavička požadavku se skládá z různě velkých znaků, dvojtečkou a poté s hodnotou konkrétní hlavičky. Hlavičky se řadí do následujících typů:

- Obecné hlavičky – do této skupiny patří hlavičky vztahující se jak na požadavky, tak na odpovědi. Data uvnitř těchto hlaviček nejsou nijak propojená s daty v tělu požadavku, či odpovědi.

- Hlavičky požadavků – obsahuje informace o zdroji, který má být načten, i o samotném klientovi.
- Hlavičky odpovědí – hlavičky s doplňujícími informacemi o odpovědi, jako je například její umístění, či o serveru samotném (název, verze a další).
- Hlavička entit – hlavičky obsahující informace o těle dané entity, jako je délka obsahu nebo MIME typ (typ média).

[1], [3]

3.1.2 Zdroje

Klíčovou abstraktní informací v rozhraní REST je tak zvaný zdroj. Jakékoli informace, které lze pojmenovat, mohou být zdrojem. Patří mezi ně tedy například dokument nebo obrázek, sbírka jiných zdrojů, ne-virtuální objekt (například osoba) a další. Technologie REST používá jednotný identifikátor zdroje (URI) k identifikaci konkrétního zdroje, který se podílí na interakci mezi komponentami. Stav zdroje v konkrétním čase je známý jako reprezentace zdroje. Reprezentace se skládá z dat, metadat popisujících data a hypermedia odkazů, které mohou pomoci klientům v přechodu do dalšího požadovaného stavu. Datovému typu reprezentace se říká „typ média“ (media type). Typ média identifikuje specifikaci, která definuje, jak má být definice zpracována. Restful API vypadá jako hypertext. Každá adresovatelná jednotka informací nese adresu a to buď explicitně (například pomocí atributů odkazů a id) nebo implicitně (odvozené z definice struktury reprezentace typu média).

V architektuře REST jsou data a funkcionality považovány za zdroje a jsou přístupné pomocí jednotných identifikátorů zdrojů. Zdroje jsou řešeny pomocí sady jednoduchých, dobře definovaných operací. Klienti a servery si vyměňují reprezentace zdrojů pomocí standardizovaného rozhraní a protokolu – typicky HTTP. Zdroje jsou odděleny od jejich reprezentace, takže je jejich obsah přístupný v různých formátech jako je HTML, XML, prostý text, PDF, JPG, JSON a další. Metadata zdroje jsou k dispozici a používají se například ke kontrole ukládání dat do mezipaměti, k odhalování přenosových chyb, k vyjednávání o vhodném formátu a ke kontrole přístupu nebo autentizaci. [4]

3.1.3 METODY

HTTP definuje sadu požadavkových metod, které označují požadovanou operaci, která má být provedena pro určitý zdroj. Každá z metod implementuje jinou sémantickou funkci, ale i přesto mezi sebou požadavkové metody sdílejí společné rysy jako například, bezpečnost, idempotentnost (vytvoření vícero identických požadavků má stejný výsledek jako jeden požadavek) nebo uložitelnost v mezipaměti. Mezi nejznámější metody patří GET, POST, PUT, PATCH a DELETE. Díky nim je možné využívat operace pro vytváření, čtení, aktualizování a mazání. Mimo tyto metody existují i jiné, ale již méně využívané, mezi ně patří například OPTIONS a HEAD. [5], [6]

3.1.3.1 GET

Metoda GET se používá pro čtení a načítání reprezentace zdroje. V případě, že byl požadavek vyřízen úspěšně, vrací GET reprezentaci v XML formátu, či JSONu a jako stavový kód vrácí hodnotu 200 (značící úspěch). V opačném případě vrací nejčastěji 404 (značící neexistující dokument na dané URL) nebo 400 (značící, že požadavek nemůže být vyřízen, poněvadž byl syntakticky nesprávně zapsán). Podle návrhu specifikace HTTP jsou požadavky GET (společně s metodou HEAD) používány pouze pro čtení dat, a ne k jejich změně. Proto jsou při tomto používání považovány za bezpečné. To znamená, že je možné je volat bez rizika změny, či poškození dat. Metody GET a HEAD jsou obě idempotentní.

Formuláře odesílající data metodou GET ukládají hodnoty a klíče do URL webu. Výhodou této věci je, že tyto stránky jsou cachovatelné a dá se na ně kdykoliv vrátit. Kvůli tomu, že jsou ale data viditelná v adresním řádku není doporučené používat metodu GET na formuláře, kde se vyplňují citlivé údaje. [5], [6], [7], [8]

3.1.3.2 POST

Metoda POST se nejčastěji používá k vytvoření nových zdrojů. Zejména se používá při vytváření podřízených zdrojů (rodičovským zdrojům). Jinými slovy, při vytváření nového zdroje se webová služba postará o přidružení nového zdroje k rodiči, přidělením unikátního ID (nový URI zdroj). Po úspěšném vytvoření vrací POST metoda stavový kód

s hodnotou číslo 201 (značící úspěšné vytvoření zdroje identifikovatelného dle URI) a hlavičku obsahující umístění s odkazem na nově vytvořený zdroj. POST není ani bezpečná (bezpečné metody lze ukládat do mezipaměti, bez jakýchkoli dopadů na zdroj), ani idempotentní metoda. Proto se doporučuje pro neidempotentní požadavkové zdroje. Vytvoření dvou identických požadavků typu POST má za následek dva zdroje obsahující stejné informace. Na rozdíl od metody GET se nedá POST uložit do mezipaměti a do historie prohlížeče. Ale oproti GET nemá žádná velikostní omezení. [5], [6], [7], [8]

3.1.3.3 PUT

Metoda PUT je nejčastěji využívána pro operace související s aktualizacemi, tím že přidá známé URI zdroje s tělem požadavku do nově aktualizované reprezentace původního zdroje. PUT může být také použit k vytvoření zdroje a to i v případě, že ID zdroje vybere klient namísto serveru. Opět platí, že tělo požadavku obsahuje zdroj reprezentace. Vzhledem k složitějšímu postupu by se přidávání za užití metody PUT mělo používat šetrně, pokud vůbec. Případně užitím metody POST, která je pro přidávání zdrojů lepším řešením.

Při úspěšné aktualizaci vrací PUT metoda v těle odpovědi stavový kód číslo 200 (úspěch) nebo 204, pokud nevrátí obsah v těle. Při použití metody PUT při vytváření zdroje se vrací stavový kód 201 po úspěšném vytvoření. Tělo odpovědi je volitelné za předpokladu, že se spotřebuje větší šířka pásma. V případě vytvoření zdroje metodou PUT se odkaz nemusí vrátit pomocí hlavičky pro umístění, protože ID zdroje bylo již nastaveno klientem. Metoda PUT nepatří mezi bezpečné, protože modifikuje (nebo vytváří) stav na serveru a přitom je idempotentní. Jinými slovy, pokud se provede vytvoření nebo aktualizace zdroje pomocí PUT, a poté se provede znova to samé, zdroj zůstane tam kde byl a bude mít stejný stav jako při prvním provedení metody PUT. Pokud má například volání PUT zá následek zvýšení hodnoty u počítadla, pak již metoda nebude idempotentní. Pro takovéto případy se důrazně doporučuje metoda POST. [5], [6], [7], [8]

3.1.3.4 PATCH

PATCH se používá pro operace související s úpravami zdrojů. PATCH požadavek musí obsahovat pouze změny zdroje, nikoliv úplný zdroj. Touto vlastností se podobá metodě PUT, ale na rozdíl od ní obsahuje tělo požadavku u metody PATCH soubor instrukcí popisujících jak by měl být zdroj, který je aktuálně umístěný na serveru – upraven pro vznik nové verze. Změny popsané v PATCH dokumentu musí být sémanticky dobře definované, a zároveň mohou být v jiném formátu, než samotný modifikovaný prostředek. Pro popis změn mohou být použity programovací jazyky JSON (JSON PATCH), či XML (XML PATCH).

Metoda PATCH není bezpečná, ani idempotentní a musí používat veškeré mechanismy, jako jsou podmíněné požadavky užitím Etagů (jeden z několika mechanismů, které poskytuje HTTP pro funkci ověření vyrovnávací paměti) a požadavkové hlavičky „If-Match“, aby se zajistilo, že data nejsou poškozena v průběhu úpravy zdroje. V případě selhání PATCH metody nebo vypršení časového limitu na serveru, může klient ke kontrole stavu zdroje použít požadavek GET. Server musí zajistit, že škodlivý klienti nadále nevyužívají metodu PATCH k vyčerpání serverových prostředků. [6], [7], [8]

3.1.3.5 DELETE

DELETE se používá pro odstranění konkrétního zdroje identifikovaného jednotným identifikátorem zdroje – URI. Tato metoda může být na serveru potlačena lidským zásahem (nebo jinými prostředky). Při úspěšném provedení tohoto požadavku, vrací DELETE stavový kód číslo 200 (úspěch), spolu s tělem odpovědi, ve kterém je reprezentace smazaného zdroje. V případě, že server úspěšně zpracoval požadavek smazání, ale nevrátil obsah – vrací DELETE stavový kód 204.

Operace DELETE metod patří mezi idempotentní. Při smazání zdroje, je zdroj smazán již navždy. Opakované volání této metody na stejný zdroj skončí vždy stejným způsobem – smazaným zdrojem. [7], [8]

3.1.3.6 HEAD

Metoda HEAD je totožná s metodou GET až na rozdíl, že server nemůže v odpovědi metody HEAD vrátit tělo zprávy. Metainformace obsažené v HTTP hlavičkách odpovědi na požadavek HEAD musí být identické s informacemi zaslanými v odpovědi na GET požadavek. Tato metoda se dá využít pro získání metainformací o subjektu bez přenesení těla. Metoda se nejčastěji používá pro testování platnosti, přístupnosti a získání informací o nedávných úpravách u hypertextových odkazů.

Odpověď na HEAD požadavek může být ukládána do mezipaměti. Díky tomu mohou být informace obsažené v odpovědi použity k aktualizaci dříve uložené mezipaměti z daného zdroje. Pokud nové hodnoty v poli naznačují, že se mezipaměť liší od aktuální entity (při změně některé z hlaviček content-length, content-MD5, ETag, či Last-Modified), pak musí mezipaměť považovat tento záznam za zastaralý. [6], [8]

3.1.3.7 OPTIONS

Metoda OPTIONS představuje požadavek, jenž získává informace o možnostech komunikace. Ty jsou dostupné ze závislosti „žádost – odpověď“ identifikovanými URI. Tato metoda umožňuje klientovi určit možnosti a požadavky spojené se zdrojem, či serverem, aniž by se jednalo o akci zdroje nebo iniciování získávání zdroje. Odpovědi na tuto metodu se nedají uložit do mezipaměti. Pokud OPTIONS požadavek obsahuje tělo zprávy (naznačeno přítomností hlaviček Content-Length nebo Transfer-Encoding), pak musí být typ média označen v hlavičce Content-Type. Přestože tato specifikace neurčuje pro tělo žádné využití, budoucí HTTP rozšíření mohou využít tělo OPTIONS pro podrobnější dotazy na serveru.

Pokud se při volání metody OPTIONS nachází v URI znaménko hvězdičky „*“, pak tato metoda odkazuje na server, nikoliv na určitý zdroj. Jelikož komunikační možnosti serveru obvykle závisí na zdroji, tak je požadavek s hvězdičkou užitečný pouze jako metoda „ping“ nebo „no-op“ (metody obnovující spojení se serverem), neboť nedělá nic, než že umožní klientovi otestovat možnosti serveru. Pokud se v URI požadavku nenachází hvězdička, pak se požadavek OPTIONS vztahuje pouze na možnosti, které jsou k dispozici při komunikaci s tímto zdrojem.

Odpověď se stavovým kódem číslo 200 (úspěch) musí obsahovat všechny hlavičky, které označují volitelné funkce implementované serverem a použity pro daný zdroj (například „Allow“ – hlavička, která uvádí seznam podporovaných HTTP metod podporovaných zdrojem), případně včetně rozšíření, které nejsou definovány touto specifikací. Tělo odpovědi (pokud se v odpovědi tělo zprávy nachází), musí obsahovat informace o možnostech komunikace. Formát takového těla není touto specifikací definován, ale může být definován budoucími rozšířeními HTTP. Pokud v odpovědi OPTIONS žádné tělo není, tak musí odpověď obsahovat hlavičku Content-Length s hodnotou 0. [5], [6], [7], [8]

3.1.4 Stavové kódy

HTTP stavový kód je odpovědí serveru na požadavek odeslaný z prohlížeče. Při návštěvě webové stránky odešle prohlížeč na server webu požadavek a ten mu poté odpoví na tento požadavek třímístným kódem, který se nazývá stavový kód. Stavové kódy jsou internetovým ekvivalentem konverzace mezi prohlížečem uživatele a serverem. Během této komunikace vrací server prohlížeči informaci o jejím výsledku.

Porozumění stavovým kódům a jak je správně používat, pomáhá rychle diagnostikovat chyby na webu. První číslice každého třímístného stavového kódu začíná jedním z čísel od jedničky do pětky. Pro označení kódů v daném rozsahu se dá použít zápis „1xx“, či „5xx“. Každý z těchto rozsahů zahrnuje jinou třídu odpovědi serveru. Mezi nejznámější stavové kódy patří:

Tabulka 1 - Nejznámější stavové kódy

Stavový kód	Význam
200	Standardní odpověď pro úspěšný HTTP požadavek.
302	Přesměrováno. Pro získání původního zdroje je potřeba jít jinam.

404	Požadovaný dokument nebyl nalezen, ale v budoucnosti může ale i nemusí být dostupný.
500	Obecná chybová zpráva. Při zpracovávání požadavku došlo ke blíže nespecifikované chybě.

Zdroj: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
[4]

3.1.5 URI

Uniform Resource Identifier, neboli „jednotný identifikátor zdroje“ je posloupnost znaků s definovanou strukturou, která slouží k přesné specifikaci zdroje informací za účelem jejich použití v počítačové síti, zejména internetu. Jelikož je URI velmi obecný koncept, jeho základní formát je velmi volný. V principu se jedná o název takzvaného schématu, následovaný dvojtečkou a dále prakticky libovolným řetězcem, jehož význam a formátování už závisí právě na použitém schématu. Standard URI specifikuje pouze základní syntaxi, která popisuje, které znaky je dovoleno v URI použít apod. URI má následující tvar:

```
Schéma:hiearchická část?dotaz#fragment
```

Schéma stanovuje konkrétní syntaxi a všechny přidružené protokoly URI. Je tvořená malými i velkými písmeny, které jsou následovány dvojtečkou. Další součástí syntaxe je takzvaná hierarchická část. Ta obsahuje identifikátor zdroje v rámci hierarchické struktury. Standard URI dovoluje, aby tato část byla formátována prakticky libovolně, ale předepisuje také několik předdefinovaných syntaxí užitečných pro obvyklé situace. Jednou z nich je formát, kde po dvojtečce oddělující název schématu následují dvě lomítka („//“), po kterých následuje označení tzv. autority, jenž je tvořeno jménem či IP adresou počítače. Informace o uživateli smí být před IP adresou počítače oddělená zavináčem („@“), za kterým smí být číslo portu oddělené dvojtečkou. Za označením autority následuje cesta, což je posloupnost

segmentů oddělených lomítky („/“). Toto značení je obdobné adresářům, ale nemusí se jednat přímo o ně, ale obecně o jakýkoli hierarchický systém.

Předposlední část syntaxe URI – dotaz obsahuje řetězec nehierarchických dat. Ačkoli jeho syntaxe není kompletně definovaná, je nejčastěji složená z posloupností dvojic, jejichž hodnoty atributů jsou oddělené oddělovačem, jako je ampersand nebo středník. Dotaz je od předchozí části oddělen otazníkem. Poslední částí základní syntaxe URI je fragment. Ten obsahuje identifikátor fragmentu, jenž poskytuje směr k sekundárnímu zdroji. Pokud je například primárním zdrojem dokument HTML, tak je fragment často atribut ID konkrétního prvku daného dokumentu. Pokud fragment identifikuje určitou část článku identifikovanou zbytkem identifikátoru URI, webový prohlížeč tento obsah posouvá. Fragment je oddělený od předchozí části znakem křížku („#“). URI se skládá ze dvou typů:

- Jednotné identifikátory zdrojů („URL“) – Tento typ začíná uvedením, který protokol by měl být použit pro vyhledání a přístup k fyzickým nebo logickým zdrojům v síti. Například pokud je zdrojem webová stránka, tak bude URI začínat protokolem HTTP. Je-li zdrojem soubor, začne URI protokolem FTP a tak dále. URL adresy nejsou trvalé, a proto pokud se změní umístění zdroje, tak je potřeba následně změnit URL adresy, aby odkazovaly na nové umístění zdroje.
- Jednotné jména zdrojů („URN“) – Tento typ neuvádí, který protokol by měl být použit pro vyhledání a přístup ke zdroji, ale jednoznačně označuje zdroj trvalým jedinečným identifikátorem nezávislým na umístění. Jednotné jména zdrojů identifikují zdroj v průběhu jeho životního cyklu a nikdy se nezmění. Každá URN se skládá z tagů URN, dvojteček a řetězce, který slouží jako jedinečný identifikátor.

[1], [9]

3.2 REST

REST definuje sadu architektonických principů, pomocí kterých se dají navrhnout webové služby, které se zaměřují na systémové zdroje jenž jsou adresovány a přenášeny přes HTTP širokému okruhu klientů - jedná se o způsob přístupu k webovým službám. Technologie REST nevyžaduje zpracování a je snazší a flexibilnější než protokol SOAP (Simple object Access protocol), což je další webová služba vyvinutá společností Microsoft

pro výměnu zpráv založených na XML jazyce přes síť, většinou pomocí HTTP. REST je oproti SOAP mnohem jednodušší a i používanější, neboť asi 70% API webových služeb dnes používá rozhraní REST API. [10], [11]

3.2.1 Historie REST

Historie technologie REST je silně spjatá s historií webu. Ten byl vytvořen ve skupině pro získávání a kontrolu dat v Evropské organizaci pro jaderný výzkum CERN v Ženevě anglickým vědcem Timem Bernsem Leem v roce 1989, který mimo jiné vymyslel a realizoval:

- Jednotný identifikátor zdroje (URI), neboli syntaxe, která přiřazuje každému dokumentu jedinečnou adresu
- Hypertextový přenosový protokol (HTTP) – jazyk založený na zprávách, pomocí kterých mohou počítače komunikovat na internetu
- Hypertextový značkovací jazyk (HTML), který představuje informativní dokumenty, které obsahují odkazy na další související dokumenty
- První webový server
- První webový prohlížeč také nazvaný World wide web, ale později přejmenovaný na „Nexus“ z důvodu přehlednosti
- První WYSIWYG editor, jenž byl zabudován přímo do prohlížeče

Od tohoto okamžiku začal web exponenciálně růst. Během pěti let počet uživatelů internetu vzrostl na 40 milionů. Ve skutečnosti byl web v této době příliš velký a rychlý a poněvadž neexistovali žádné stabilizující prostředky, tak směřoval ke zhroucení. Webová návštěvnost překonala kapacitu internetové infrastruktury. Navíc jádrové protokoly webu nebyly jednotně implementovány a neměly podporu pro cachování a jiné stabilizující zprostředkovatele. S takhle rychlým růstem nebylo jasné, zda by se web uzpůsobil, aby splnil rostoucí poptávku. Spoluzakladatel Apache HTTP Server Project si byl problému růstu webu vědom – a po důkladné analýze uznal, že tento problém byl řízen souborem klíčových omezení. On a další lidé se proto snažili zlepšit implementaci webu pomocí pragmatického přístupu a rovnoměrně uspokojit všechna tato klíčová omezení tak, aby se mohl web dál rozšiřovat.

V roce 2000, po odvrácení krize rozpínání webu, Roy Fielding (americký počítačový specialista, jeden z hlavních autorů HTTP specifikace a zakladatel stylu webové architektury REST) pojmenoval a popsal webovou architekturu v rámci jeho disertační práce „Representational State Transfer" (REST). Tímto názvem Fielding pojmenoval popis architektury webu, který se skládá z klíčových omezení world wide webu, jenž musí být splněny, aby bylo možné označit konkrétní rozhraní jako „RESTful". Mezi tato omezení, která Fielding seskupil do šesti kategorií společně označovaných jako „styl webové architektury" patří: model klient – server, jednotné rozhraní, vrstvený systém, mezipaměť, bezestavovost a kód na požádání. [4], [12]

3.2.1.1 Model klient – server

Rozložení úloh je ústředním tématem webových omezení typu klient-server. Web je systém založený na modelu klient-server, v němž mají klienti a servery rozdílné, samostatné úlohy. Mohou být implementovány a rozmístěny nezávisle užitím různých programovacích jazyků nebo technologií, dokud odpovídají jednotnému rozhraní webu. [4]

3.2.1.2 Jednotné rozhraní

Interakce mezi komponentami webu – tedy jejich klienty, servery a síťovými zprostředkovateli závisí na jednotnosti jejich rozhraní. Pokud se některá ze součástí odchyluje od zavedených standardů, pak se komunikační systém webu rozpadne. Webové komponenty soustavně spolupracují v rámci čtyř omezení, které jsou definovány jako:

- Identifikace zdrojů - Každá webová koncepce je známa jako zdroj a může být řešena jednoznačným identifikátorem – URI.
- Manipulace zdrojů prostřednictvím reprezentace - Klienti manipulují s reprezentacemi zdrojů. Stejný, přesný zdroj ale může být reprezentován různým klientům různými způsoby. Například dokument může být do webového prohlížeče reprezentován jako HTML a jako JSON do automatizovaného programu. Klíčovou myšlenkou je, že reprezentace je způsob interakce se zdrojem, ale není to samotný

zdroj. Toto koncepční rozlišení umožňuje, aby byl zdroj zastoupen různými způsoby a formáty, aniž by se nikdy změnil jeho identifikátor.

- Samopopisné zprávy - Požadovaný stav zdroje může být reprezentován pomocí zprávy uvnitř požadavku klienta. Aktuální stav zdroje může být reprezentován ve zprávě odpovědi, která se vrátí ze serveru. Příkladem může být uživatel administrace webu, jenž může odeslat v požadavku zprávu k přenosu reprezentace, jenž naznačuje aktualizaci stránky (nový obsah) pro webovou stránku (zdroj) spravovanou serverem. Nyní je na serveru, zda požadavek klienta přijme či zamítne. Samopopisné zprávy mohou obsahovat metadata pro předání dalších podrobností o stavu zdrojů, formátu a velikosti reprezentace a samotné zprávě. Zpráva HTTP umožňuje hlavičkám uspořádat různé typy metadat do jednotných polí.
- Hypermedia jako aplikační stav (HATEOAS) – Reprezentace stavu zdrojů zahrnuje odkazy na související zdroje. Odkazy jsou vlákny, které vzájemně propojují web a umožňují uživatelům přenášet informace a aplikace smysluplným způsobem. Přítomnost, či nepřítomnost odkazu na stránce je důležitou součástí současného stavu zdroje.

[4]

3.2.1.3 Vrstvený systém

Omezení vrstveného systému umožňují síťovým zprostředkovatelům jako je třeba proxy, brány (gateway), aby byly transparentně rozmístěny mezi klientem a serverem pomocí jednotného rozhraní webu. Síťový zprostředkovatel zachycuje komunikaci mezi klientem a serverem za určitým cílem. Síťový zprostředkovatelé jsou běžně používány pro zabezpečení, ukládání odpovědí do vyrovnávací paměti a vyrovnávání zatížení. [4]

3.2.1.4 Mezipaměť

Ukládání do mezipaměti (cache) je jedno z nejdůležitějších omezení webové architektury. Omezení mezipaměti instruují webový server, aby deklaroval vyrovnávací paměť dat pro všechny data z odpovědí. Ukládání dat odpovědí do mezipaměti může snížit

latenci (časovou prodlevu) klienta, zvýšit celkovou dostupnost a spolehlivost aplikace a kontrolovat zatížení webového serveru – ukládání do mezipaměti snižuje celkové náklady na web.

Mezipaměť může existovat kdekoliv na síti mezi klientem a serverem. Může být v sítích webových serverů, v sítích specializovaných distribučních sítích (mezi která patří CDN, což je síť vzájemně propojených počítačů skrze internet, která umožňuje dostupnost obsahu nebo dat uživatelům) nebo uvnitř samotného klienta. [4]

3.2.1.5 Bezestavovost

Bezestavovostní omezení nařizuje, že webový server není povinen zapamatovat si stav svých klientských aplikací. V důsledku toho musí každý klient zahrnovat všechny kontextové informace, které považuje za relevantní při každé interakci s webovým serverem. Webové servery vyžadují od klientů, aby spravovali složitost komunikace s jejich aplikačním stavem tak, aby webový server mohl obsluhovat mnohem větší počet klientů. Tento kompromis je klíčovým faktorem pro škálovatelnost architektury stylu webu. [4]

3.2.1.6 Kód na vyžádání

Web velmi často používá tzv. kód na požádání. Jendá se o omezení, které umožňuje webovým serverům dočasně přenášet klientům spustitelné programy, jako jsou skripty nebo pluginy. Kód na vyžádání má tendenci vytvářet technologickou vazbu mezi webovými servery a jejich klienty, protože klient musí být schopen porozumět a provést kód, který se stáhne na požádání ze serveru. Z tohoto důvodu je kód na vyžádání jediným omezením architektury stylu webu, který je považován za nepovinný. Technologie hostované webovým prohlížečem, jako jsou Java applety, JavaScript a Flash, ilustrují omezení kódu na vyžádání. [4]

3.2.2 Tvorba REST API

REST API je rozhraní, pomocí kterého mnoho vývojářů manipuluje s daty přes HTTP protokol. Dobře navržené API je snadné na používání a ulehčuje vývojářům práci při jeho používání. Při vytváření restového API je proto potřeba dbát na různá pravidla pro tvorbu výsledné URI, pro správné nastavení stavových kódů, pro práci s požadavkovými metodami a pro vytváření správného obsahu těla zprávy. Při dodržování těchto pravidel je výsledné rozhraní v souladu se standardy dobře navrženého restového API.

3.2.2.1 Pravidla pro URI

REST API používají jednotné identifikátory zdrojů pro adresování zdrojů. Při návrhu API je doporučeno se držet následujících pravidel:

- Přední lomítko („ / “) musí být použito k označení hierarchického vztahu.
- Na konci URI by nemělo být koncové lomítko – toto koncové lomítko nepřidává žádnou sémantickou hodnotu a může vést k nejasnostem. REST API by to proto neměla zahrnovat do odkazů, které poskytují klientům. Všechny znaky uvnitř URI počítají s unikátní identitou zdroje – dvě různé URI mapují dva různé zdroje. Pokud se URI liší, tak se liší i zdroje a naopak. Proto by REST API mělo generovat čisté URI a zároveň by mělo být tolerantní vůči drobným chybkám v adrese a proto je vhodné například pomocí souboru htaccess přesměrovávat klienty na správné URI (za užití funkce „Redirect 301“).
- Pro zlepšení čitelnosti by měly být používány pomlčky – Aby byly delší slovní spojení v URI co nejlépe čitelné, je vhodné použít pomlčku na místech, kde by byla normálně mezera.
- Nepoužívat podtržítka („_“) – Aplikace pro prohlížení textu (prohlížeče, editory a další) často podtrhují jednotné identifikátory zdroje a poskytují tak uživateli informaci, že jsou klikatelné. V závislosti na fontu, který aplikace používá se pak může stát, že podtržítko v URI bude částečně, či úplně skryto.
- Upřednostňovat malá písmena v URI.
- Koncovky souborů by neměly být součástí URI – Na webu je znak tečky („.“) obvykle používán k oddělení názvu souboru od formátu souboru. Tato informace,

ale není v URI za potřebí, poněvadž se typ výsledného formátu posílá prostřednictvím hlavičky „Content-type“.

[4]

3.2.2.2 Pravidla pro požadavkové metody

Při navrhování vlastního REST API je ideální dodržovat dané standardy a užívat pro konkrétní situace správnou požadavkovou metodu. Proto je vhodné dodržovat následující pravidla:

- Metody GET a POST nesmí být používány pro tunelování jiných metod požadavků – tunelování se týká jakéhokoliv zneužívání HTTP, které maskuje nebo zkresluje původní záměr zprávy a podkopává průhlednost protokolu. REST API nesmí ohrozit svůj návrh tím, že zneužije metody požadavků HTTP ve snaze vyhovět klientům s omezenou slovní zásobou HTTP.
- Metoda GET musí být používána k získání reprezentace zdroje – Klient REST API používá GET metodu v požadavkové zprávě k načtení stavu zdroje v konkrétní, čitelné formě. Zpráva klientova GET požadavku může obsahovat hlavičky, ale nemusí obsahovat tělo zprávy. Architektura webu se spoléhá na povahu metody GET. Klienti spoléhají na to, že budou moci opakovat požadavky GET, aniž by nastaly nějaké chyby. Ukládání do mezipaměti závisí na schopnosti zobrazovat výslednou reprezentaci v mezipaměti bez kontaktování serveru.
- Metoda HEAD by se měla používat pouze pro získávání hlaviček odpovědí – Metodu HEAD využívají klienti k získání hlaviček bez těla zprávy. Jinými slovy vrací tato metoda stejnou odpověď jako GET až na to, že API vrací prázdné tělo zprávy. Klienti mohou tuto metodu použít ke kontrole existence zdroje nebo ke čtení jeho metadat.
- Metoda PUT musí být použita pro aktualizaci a vkládání uloženého zdroje – PUT musí být použita k přidání nového zdroje za užití URI specifikovaného klientem. PUT se musí také použít při aktualizaci nebo nahrazení již uloženého zdroje.
- Klient musí použít metodu požadavku PUT k provádění změn zdrojů. Zpráva požadavku může obsahovat tělo, které obsahuje požadované změny.

- POST metoda musí být použita při vytvoření nového zdroje v kolekci – Klienti využívají POST při pokusu o vytvoření nového zdroje v „kolekci“ (adresář zdrojů spravovaný serverem). Tělo požadavku POST metody obsahuje navrhovanou reprezentaci nového zdroje, která má být přidána do kolekce vlastněné serverem.
- Metoda DELETE musí být použita k odebrání zdroje od jeho nadřazeného zdroje – Jakmile je pro konkrétní zdroj zpracován DELETE požadavek, není již možné klientem tento zdroj nalézt. Proto by měl každý budoucí pokus o načtení reprezentace zdroje pomocí metody GET, či HEAD vést ke statusu 404 („nenalezeno“).
- Metoda OPTIONS by měla být používána pouze k načtení metadat, které popisují dostupné interakce zdroje – Klienti mohou využít požadavkovou metodu OPTIONS k načtení metadat zdrojů, která obsahují hlavičku „Allow“ (hlavička, jenž uvádí seznam podporovaných metod zdroje). V odpovědi na OPTIONS požadavek může REST API obsahovat tělo, které obsahuje další podrobnosti o veškerých interakcích.

[4]

3.2.2.3 Pravidla pro obsah těla zprávy

REST API obvykle používá ve svých odpovědích takzvané „tělo“ zprávy, které pomáhá sdělit stav odpovědi identifikovaného zdroje. Tato zpráva je často v textovém formátu, jenž reprezentuje stav zdroje jako soubor smysluplných polí. Mezi nejčastější textové formáty používané v těle zprávy patří JSON a XML. XML podobně jako HTML organizuje data dokumentu do vnořujících tagů ohraničenými hranatými závorkami. Aby si XML dokument zasloužil označení „well-formed“ (správný zápis), musejí být veškeré tagy v dokumentu správně ukončeny. Díky tomuto párovému systému si drží XML svoji strukturu.

JSON (javascriptový objektový zápis) na rozdíl od XML používá složené závorky pro hierarchické strukturování informací v dokumentu. Na tento styl zápisu je většina programátorů zběhlých v objektovém programování zvyklá, a proto je tento jazyk velice populární při tvorbě REST API. JSON využívá předností javascriptu a těží z bezproblémové integrace s nativním prostředím prohlížeče. Pokud zdroj nemá standartní typ (například typ „image/jpeg“ pro obrázky komprimované ve formátu JPEG) – měl by být použit jazyk JSON pro strukturování jeho informací.

Objekt v JSON formátu je neseřazená sada dvojic ve formátu – název-hodnota. Syntaxe JSON formátu definuje názvy jako řetězce, které jsou obklopeny dvojitými uvozovkami, kdy správný zápis může vypadat třeba takto:

```
{
    "jmeno": "Karel",
    "prijmeni": "Novák",
    "vek": "26",
    "vzdelani": {
            "zakladni" : "ZŠ Brána Jazyků",
            "stredni" : "SPŠE Ječná"
    }
}
```

[4]

3.3 Využité nástroje

3.3.1 PHP

PHP je skriptovací jazyk, jenž se vykonává na straně serveru (server-side), navržený speciálně pro tvorbu dynamických webových stránek. Umožňuje vložit uvnitř HTML jazyka PHP kód (uvnitř <?php a ?> tagů), který se provede při každém načtení stránky. Kód PHP je interpretován na webovém serveru a generován do HTML formy nebo klidně do jiného finálního formátu, který uživatel uvidí v prohlížeči. Jedná se o velmi populární jazyk v oblasti tvorby webu. V květnu 2013 běželo PHP na tří čtvrtině světových webových stránek a tento počet vyrostl v červenci 2016 na více než 82%. Tento jazyk běží pod open-source licencí, což znamená že uživatel má přístup k jeho zdrojovému kódu a má možnost jej používat, měnit a redistribuovat.

Někteří z hlavních konkurentů PHP jazyka jsou Python, Ruby, Node.js, Perl, Microsoft .NET a Java. Ve srovnání s těmito produkty má PHP mnoho silných stránek včetně následujících:

- Výkon – PHP je velmi rychlé. Pomocí jediného levného serveru se dá obsloužit až jeden milion návštěv za den, ať už se jedná o stránku s jednoduchým formulářem, či sociální síť.
- Rozhraní s mnoha různými databázovými systémy – PHP má nativní připojení k mnoha databázovým systémům. Kromě služby MySQL se přes tento jazyk dá připojit mimo jiné k databázovým serverům PostgreSQL, Oracle, MongoDB a MSSQL (Microsoft SQL server). PHP 5 a PHP 7 mají také vestavěné rozhraní pro ploché databázové soubory (jednoduchá databáze ukládající data do textového souboru ve formě prostého textu), které se nazývá SQL-Lite. Pomocí standardu Open Database Connectivity (ODBC) se může přes PHP připojit k jakékoliv databázi, která poskytuje ovladač ODBC. Toto zahrnuje produkty od Microsoftu a mnoho dalších. Kromě nativních knihoven přichází PHP s databázovou abstrakční vrstvou nazvanou PHP database objects (PDO), která umožňuje konzistentní přístup a podporuje bezpečné metody kódování pomocí SQL „předpřipravených“ dotazů. Takto předpřipravené dotazy zadané pomocí knihovny PDO jsou chráněné vůči SQL injection (útok na internetové stránky, z pravidla přes neošetřený formulář). Předpřipravený PDO dotaz může vypadat například takto:

```php
<?php
$query = $pdo->prepare("
      SELECT name, username FROM users WHERE id = ?"
);
$result = $query->execute( array($user_id) );
```

- Vestavěné knihovny pro spoustu běžných úkonů – Vzhledem k tomu, že byl jazyk PHP navržen pro používání pro web, tak obsahuje mnoho vestavěných, užitečných funkcí souvisejících s webem. Pomocí PHP se dají za pomoci pár řádků kódu za běhu například generovat obrázky, připojovat se k webovým službám a dalším síťovým službám, parsovat XML, odesílat emaily, využívat COOKIE proměnné, či generovat PDF dokumenty.
- Nízká cena – PHP je zadarmo. Jeho nejnovější verze se dá kdykoliv stáhnout z jeho oficiálních stránek.

- Lehký na naučení a používání – Syntaxe PHP je založená na jiných programovacích jazycích, především na C a Perl, čili je pro programátory znající jakýkoliv jazyk založený na C snadný k pochopení.
- Přizpůsobený pro objektové programování – Od páté verze má PHP lépe navržené objektově orientované funkce, které byly ještě více vylepšeny v sedmé verzi. Funkce jsou velmi podobné funkcím v jazycích založených na jazyce C. Patří mezi ně funkce jako dědičnost, privátní a chráněné atributy a metody, abstraktní třídy a metody, rozhraní, konstruktory a destruktory.
- Přenositelnost – PHP je k dispozici pro mnoho různých operačních systémů. Programovat se dá v tomto jazyce na bezplatných unixových systémech jako je Linux nebo FreeBSD, komerčních verzích unixu (nejznámější je v této kategorii OS X od Apple) , či na různých verzích Microsoft Windows. Dobře napsaný kód v PHP se dá bez problému přenášet mezi systémy.
- Flexibilita rozvojového přístupu – PHP umožňuje provádět jednoduše snadné úkoly. Stejně jednoduše zvládá ale i komplexnější úkoly jako je implementace velkých aplikací za užití frameworku založeným na architektuře MVC (architektura jenž dělí aplikaci na tři logické části – model, view, controler).
- Dostupnost zdrojového kódu – Na rozdíl od komerčních produktů s uzavřenými zdrojovými kódy, je možné v PHP cokoliv měnit, či přidávat. Programátoři mají tedy možnost nutně nečekat až výrobce uvolní opravy nebo funkčnosti a sami si zdroj upravit podle svého.
- Dostupnost podpory a dokumentace – Společnost stojící za hlavním vývojem PHP se nazývá Zend engines. Tato firma řídí a financuje vývoj PHP tím, že nabízí podporu a související software na komerční bázi. PHP má obrovskou, aktivní komunitu na internetu. Téměř jakýkoliv problém spojený s tímto jazykem již někdo jiný na internetu vyřešil, takže není problém najít lehce řešení na jakékoliv nesnáze při programování.

[13], [14]

3.3.1.1 Historie

V roce 1994 vytvořil Rasmus Lerdorf webové rozhraní, které se označuje jako první verze PHP jazyka. Jednalo se o malou sadu skriptů naprogramovanou v jazyce C nazvanou CGI (Common Gateaway Interfaces), kterou Lerdorf vytvořil pro obsluhu ke svým webovým stránkám. Později byly tyto skripty rozšířeny o schopnost práce s webovými formuláři a komunikací s databázemi. Tato nová implementace se nazývala „Personal HomePage / Forms Interpreter“ (zkráceně PHP /FI) a sloužila jako Framework, na kterém mohli vývojáři vytvářet dynamické webové aplikace. V červnu 1995 byl zdrojový kód uvolněn pro veřejnost, a díky tomu tak bylo možné tento jazyk volně využívat, opravovat v něm chyby a vylepšovat jej. Původní záměrem PHP/FI nebylo vytvoření nového programovacího jazyka, jelikož ale jeho vývoj probíhal samovolně a bez větší kontroly, docházelo k některým problémům, jako je nesoulad názvů funkcí nebo jejich parametrů. Konkrétně byly názvy některých funkcí stejné jako názvy menších knihoven, které PHP využívalo. V říjnu 1995 vydal Lerdorf novou, kompletně přepsanou verzi kódu. Toto vydání PHP bylo první vydání, jenž bylo považováno za pokročilé rozhraní pro vytváření skriptů a tím se začalo PHP podobat jazyku, kterým je dnes. Jazyk PHP byl navržen, aby byl co nejvíc podobný struktuře jazyku C, aby se o něm dozvěděli vývojáři, kteří byli s tímto populárním jazykem seznámeni. Společně s růstem funkcí PHP, rostl i počet uživatelů. Průzkum firmy Netcraft v květnu v roce 1998 ukázal, že téměř 60 tisíc domén mělo hlavičky obsahující PHP, což bylo přibližně 1% domén na internetu v té době.

Oficiální vydání PHP 3 bylo v červnu 1998. Tato verze opět obsahovala velký přírůstek nových funkcí, díky kterým byl vhodný pro všechny druhy projektů. Mezi těmito novými funkcemi bylo například rozhraní podporující připojení na více databází, podpora více protokolů, API a hlavně zahrnutí objektového programování a silnější jazyková syntaxe. V roce 1999 vznikla firma Zend Technologies, která začala přepisovat jádro PHP a tím vznikl Zend Engine (open-source skriptovací engine, který interpretuje PHP jazyk). Zend technologies se stali nejdůležitější PHP společností a hlavním přispěvovatelem jeho zdrojového kódu.

V následujících verzích docházelo k pravidelnému vylepšování Zend Enginu, většímu zabezpečení, či přidání PDO pro spolehlivé připojení k datovým úložištím a bezpečné používání SQL dotazů. Šestá verze PHP je brána jako jedno z největších selhání

pro tento jazyk. Vývoj byl zahájen v roce 2005, ale v roce 2010 byl ukončen, kvůli potížím s implementací unicode kódování. Šestá verze PHP tak patří společně s MySQL 6 a Perl 6 k neúspěšným verzím v technologickém světě.

Dne 3. Prosince 2015 byla vydána velice očekávána sedmá verze PHP s nejnovější verzí Zend Enginu. Zvýšení výkonu získané pouze změnou běžící verze PHP dosáhlo až 70%. Jazyk nyní obsahuje mimo jiné lepší, konzistentní 64-bitovou podporu a bezpečný generátor náhodných čísel. Díky této verzi se PHP stalo vážnou konkurencí takzvaným enterprise jazykům. [15]

3.3.1.2 Zpracování dynamických PHP stránek

Dynamická webová stránka je stránka vytvořená programem nebo skriptem, který běží na serveru. To znamená, že stránku lze měnit při každém zobrazení. Změny, které se na stránce můžou projevit, mohou pocházet ze zpracování dat formuláře, které uživatel odesílá, nebo zobrazováním dat, získanými z databázového serveru. Databázový server ukládá data, které jsou uspořádány v tabulkách. Tato data mohou být rychle získány databázovým dotazem. Dynamické webové stránky umožňují vývojářům webových stránek vytvářet interaktivní webové aplikace. V důsledku toho mohou uživatelé nakupovat zboží a služby, vyhledávat informace na webu a komunikovat s ostatními uživateli prostřednictvím fór, blogů a sociálních sítí. Tyto stránky by bylo obtížné nebo nemožné vytvořit nebýt dynamických webových stránek využívajících databází.

Proces zpracovávání dynamické PHP stránky začíná tím, když uživatel požaduje danou stránku ve webovém prohlížeči. To může uživatel vyvolat buď zadáním adresy URL do adresního řádku prohlížeče, kliknutím na odkaz stránky, která se má načíst, či kliknutím na tlačítko odesílajícím formulář, jenž obsahuje data, která má dynamická stránka zpracovávat. V každém z těchto případů vytvoří webový prohlížeč HTTP požadavek a odešle jej webovému serveru. Pokud uživatel odesílá formulář, tak vyplněná data budou zahrnuty do HTTP požadavku. V momentě kdy webový server obdrží HTTP požadavek, vyhledá příponu souboru požadované webové stránky a zjistí, který server programu má zpracovat požadavek. Pro PHP stránku webový server předá požadavek PHP interpretovi, který běží na webovém serveru. PHP interpret poté dostane z pevného disku příslušné PHP skripty a také načte veškeré data z odeslaného formuláře. Poté webový server dané skripty

spustí. Během vykonávání skriptů se generuje na výstup webová stránka. Skript může také požadovat data z databázového serveru a tyto data použít jako součást webové stránky, kterou vytváří.

Po dokončení skriptu předá interpret PHP dynamicky generovanou stránku zpět na webový server, který ji pošle zpět do prohlížeče v HTTP odpovědi v HTML formátu. Ve chvíli kdy prohlížeč obdrží odpověď, tak ji následně zformátuje a zobrazí jako webovou stránku. Této fázi se říká vykreslování stránky. Webový prohlížeč nemá žádný způsob jak rozeznat, jestli HTML v HTTP odpovědi pochází se statické nebo dynamické stránky, neboť jediný co přijímá je HTML. Když se stránka v prohlížeči zobrazí může uživatel konečně vidět její obsah. Poté kdy si uživatel vyžádá další stránku, tak celý tento proces začne znovu. [14]

3.3.1.3 PHP a REST API

Pro vzdálenou komunikaci s ostatními servery a volání API má PHP několik svých vlastních řešení. Jedním z nich je „cURL“. PHP podporuje knihovnu „libcurl“, která za pomoci technologie cURL umožňuje připojení a komunikaci s mnoha různými typy serverů a síťových protokolů mezi které patří například: HTTP/S, FILE, FTP, IMAP či TELNET. Na výběr jsou ale i další protokoly. CURL je velmi přizpůsobitelný a při používání umožňuje nastavit spoustu možností, které umožňují určit jakým způsobem bude probíhat interakce s URL. Užívání této technologie se skládá ze čtyř hlavních PHP příkazů:

- curl_init() – Tato metoda vrací instanci cURL pro pozdější použití. Obsahuje jeden volitelný parametr, do kterého se dá zadat string, který bude cURL automaticky brát jako URL adresu, s kterou bude nadále pracovat. URL se dá, ale nastavit i pomocí metody curl_setopt(), takže to není nutné.
- curl_setopt() – Funkce cURL je z velké části ovlivněna opakovaným voláním této metody. Ta se skládá ze tří parametrů. Do prvního parametru se vkládá instance cURL, která se má použít. Dalším parametrem je konstantní hodnota pro nastavení, které se má změnit a posledním parametrem je hodnota pro toto nastavení. Mezi věci, které se dají díky této metodě nastavit patří například nastavení URL adresy,

nastavení typu požadavkové metody, či vložení dat, která se mají přes HTTP požadavek odeslat.

- curl_exec() – Voláním této metody se dává funkci cURL najevo, že inicializace a nastavení veškerých možností je dokončeno a volání vzdálené URL se může uskutečnit. V jediném parametru, který tato metoda má je pouze instance cURL.
- curl_curl_close() – Poslední metoda má jako parametr instanci cURL. Po jejím zavolání se ukončí cURL spojení a uvolní se paměť.

Jednoduchý POST dotaz se pomocí funkce cURL zaslat takto:

```
// Inicializace cURL
$ch = curl_init();

// Nastavení URL
curl_setopt($ch, CURLOPT_URL, "http://www.supply.honza-jahoda.cz/api");

// Zvolení POST metody
curl_setopt($ch, CURLOPT_POST, true);

// Přiložená data
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);

// Vytažení dat
$responseData = json_decode(curl_exec($ch));

// Ukončení cURL spojení
curl_close($ch);
```

Další funkcí PHP pomocí který se dají řešit REST API je funkce „file_get_contents()“. Oproti cURL funkcím je její použití mnohem jednodušší a rychlejší, avšak nenabízí tolik možností nastavení. Její defaultní funkcí je, že dokáže přečíst obsah souboru, jehož cesta je uvedená v prvním parametru. Do tohoto parametru, se dá ale vložit i URL, díky čemuž i tato funkce umožňuje podobně jako cURL vzdálenou komunikaci. Defaultně tato funkci umí vykonat pouze HTTP požadavek typu GET. Pro užití ostatních požadavkových metod je třeba užití třetího parametru této funkce, do kterého se vkládá pole hlaviček. Dotazovací metoda typu POST vypadá při použití funkce file_get_contents() takto:

```php
// Uložení parametrů do řetězce, kde pole bude uloženo ve
// tvaru &klic=hodnota a interpunkce a další speciální
// znaky budou nahrazeny znaky vhodnými pro URL
$postdata = http_build_query(
array(
        'var1' => 'value 1',
        'var2' => 'value 2'
      )
);

// Uložení ostatních hlaviček společně s parametry do pole
$opts = array(
    'http' => array(
        'method' => 'POST',
        'header' => 'Content-type: application/x-www-form-urlencoded',
        'content' => $postdata
    )
);

// Zavolání funkce "file_get_contents" a vrácení výsledku do proměnné
results
$result = file_get_contents(
      'http://www.supply.honzajahoda.cz/api',
      false,
      stream_context_create($opts)
);
```

Další z možností jak v PHP pracovat s Rest API jsou již hotové pluginy. Výše uvedené metody nejsou sice co se týče náročnosti zápisu nikterak složité, ale co se týče zabezpečení, ošetřování veškerých vstupů a výstupů, tak je potřeba ještě spousta řádků kódu, aby komunikace v pořádku proběhla. Alternativou jsou pluginy, jenž mají veškeré věci ošetřeny a jsou připraveny na nejrůznější požadavky spojené s REST API. Nejznámějším z nich je PHP plugin Guzzle. Jedná se o PHP, HTTP klienta, který usnadňuje odesílání HTTP požadavků a nabízí triviální integraci s webovými službami. Guzzle obsahuje rozhraní pro vytváření dotazovacích řetězců, POST požadavků, streamování velkých souborů ke stažení, používání HTTP cookies, odesílání synchronních a asynchronních požadavků, či pro uploadování dat ve formátu JSON. Po nainstalování knihovny je použití velmi jednoduché v podstatě stačí zavolat pouze funkci, jejíž název odpovídá názvu požadované požadavkové metody. [16], [17], [18]

```php
<?php

$client = new GuzzleHttpClient();
$url = "http://www.supply.honza-jahoda.cz/api";

// Uložení parametrů do pole
$params = array(
    'form_params' => array(
        'var1' => 'value1',
        'var2' => 'value2',

    )
);

// Získání odpovědi
$response = $client->post($url, $params);
```

3.3.1.4 Frameworky

Server-side framework je software, který usnadňuje vytváření, údržbu a správu webových aplikací. Jedná se o dobře organizovaný, opětovně využitelný a udržovatelný kód, jenž urychluje vývoj aplikací, díky tomu, že poskytuje programátorovi nástroje a knihovny zjednodušující běžné úkoly spojené s vývojem webových aplikací. Aplikace naprogramované za užití frameworku umožňují růst v průběhu času, protože jsou škálovatelné. Tyto aplikace mají pevně danou strukturu a to jak mezi adresáři, tak přímo v kódu. Díky tomu se mohou programátoři webových aplikací specializovat na konkrétní frameworky a spoléhat na rychlé zorientování při práci na novém projektu v daném frameworku. Moderní frameworky jsou naprogramované za užití nejmodernějších technologií umožňující správu routování adres, snadnou interakci s databázemi, důraz na zabezpečení proti webových útokům, či zabudované uživatelské ověřování. Mezi nejznámější a nejpoužívanější PHP frameworky v současnosti patří Laravel, Symfony a Zend Framework. U nás je ale nejpopulárnější český Framework Nette.

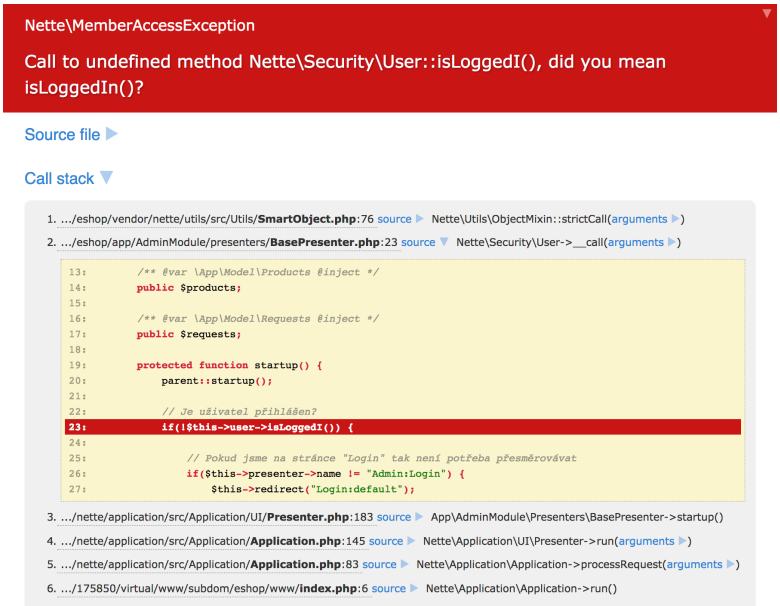
Všechny frameworky jsou vytvořené za užití softwarové architektury MVC, která rozděluje aplikaci na tři propojené části – model, view a controller. Toto rozdělení je pro oddělení interních reprezentací informací od způsobů, jakými jsou informace uživateli předkládány a přijímány. Návrhový vzor MVC odděluje tyto hlavní komponenty, což umožňuje efektivní použití kódu a paralelní vývoj. [19]

3.3.1.5 Nette

Nette je český server-side framework, jenž stejně jako ostatní PHP frameworky usnadňuje programování a správu webových aplikací, díky množství zabudovaných a užitečných tříd. Stejně jako ostatní velké frameworky, i tento funguje na bázi MVC, kdy každá část této softwarové architektury je uložená ve vlastním adresáři. Konkrétně modelová část je řešená v adresáři „Model“. V ní jsou uložené jednotlivé třídy – modely, ve kterých je obsažena veškerá aplikační logika. Jedná se o třídy, jenž slouží hlavně ke komunikaci s databází. Jednotlivé modely jsou kompletně odděleny od zbytku aplikace a komunikují pouze s presentery. View část MVC se nachází v Nette v adresáři „Templates“. Zde se nachází jednotlivé šablony pro veškeré stránky webu. Tyto šablony jsou řešené pomocí „Latte“ šablonovacího systému, který zabezpečuje výstup před zranitelnostmi jako je XSS (metoda narušení WWW stránek využitím bezpečnostních chyb ve skriptech) a umí ušetřit čas, neboť nabízí plno funkcí zkracujících a zpřehledňujících kód oproti běžnému spojení PHP a HTML jazyka. Latte je navíc oproti klasickému PHP zápisu rychlé, neboť překládá své šablony do nativního PHP kódu, který pak ukládá do mezipaměti na disk. Poslední část MVP architektury – Controller, je řešen v adresáři „Presenters“. Jedná se o řadič, který zpracovává požadavky uživatele a na jejich základě pak zavolá patřičnou aplikační logiku (model) a poté požádá View o vykreslení dat pomocí latte šablon. Presenter třídy se skládají z jednotlivých renderovacích metod, které předávají data z databáze do šablon. Každá stránka vytvořená v Nette tedy musí mít svou renderovací metodu a příslušné šablony.

Nette si umí poradit i s rychlým odhalováním a opravováním chyb. K tomu slouží jeho laděnka zvaná Tracy. V případě, že je v Nette aplikaci nastavené vývojové prostředí (režim, jenž programátorovi vypisuje citlivé informace o kódu), zobrazuje se na každé stránce malá lištička zobrazující přehledné informace o aktuálním načteném presenterua aktivní render metodě. Dále je zde výpis použitých SQL příkazů a statistiky týkající se rychlosti načtení dané stránky a využití paměti a procesoru. V případě, že během spuštění dojde k PHP chybě, vypíše Nette přehlednou chybovou stránku (oproti obyčejnému, strohému PHP výpisu), na které je jasně vidět v jakém řádku, a v jakém souboru došlo k chybě. [20]

Obrázek 3 - Chybový výpis v Nette frameworku



Zdroj: Vlastní zpracování

3.3.2 MySQL

MySQL je velmi rychlý, robustní, relační systém pro správu databází. Jeho databáze umožňují efektivně ukládat, vyhledávat, třídit a získávat data. Server MySQL řídí přístup k datům, aby se zajistilo že přístup k nim bude rychlý, dále aby se zajistilo že s nimi může pracovat současně více uživatelů, a aby k datům měli přístup pouze oprávnění uživatelé. Proto je MySQL více vláknový server s podporou práce pro více uživatelů. Používá standardizovaný dotazovací jazyk SQL. MySQL je veřejně přístupná od roku 1996, ale historie vývoje se datuje do roku 1979. Patří mezi nejpopulárnější open-source databáze na světě. Hlavními konkurenty MySQL mezi relačními databázemi je PostgreSQL, Microsoft SQL Server a Oracle. MySQL si mezi těmito konkurenty nevede špatně neboť mezi jeho silné stránky patří:

- Vysoký výkon
- Nízká cena – MySQL je k dispozici pod licencí open-source nebo za nízkou cenu při užití komerční licence. V případě aplikace, která využívá MySQL serveru a zároveň není pod licencí open-source, je potřeba pořídit komerční licenci MySQL. Aplikace,

které nejsou distribuovány (typické pro většinu webových aplikací), či aplikace pod free nebo open-source licencí nemusí používat placenou licenci.

- Snadná konfigurace a používání – Většina moderních databází používá SQL jazyk, takže i pro programátory využívající jiného typu správce relačních databází není problém při změně na toto prostředí. MySQL je také jednodušší nakonfigurovat a připravit pro používání než už u jiných, podobných produktů (například pro instalaci na lokálním počítači stačí pouze nainstalovat softwarový balíček XAMPP, či WAMPP – které se již o instalaci serveru postarají sami).
- Přenositelnost – MySQL lze používat na mnoha různých systémech UNIX i v systému Microsoft Windows.
- Dostupnost zdrojového kódu – Stejně jako v případě PHP se u MySQL dá získat a upravit jeho zdrojový kód. Tato vlastnost není pro většinu uživatelů důležitá, ale v případě nouze dává uživatelům možnost si vše dle sebe poupravit. V současnosti existuje pro MySQL několik vyvíjených nástupnických větví (forků) jako je například MariaDB, která je vyvíjena původními autory MySQL.
- Dostupnost podpory – MySQL je dceřinou společností Oracle corporation, která nabízí podporu, školení, poradenství a certifikaci.

  [13]

3.3.2.1 SQL

SQL (Structured Query Language) je databázový programovací jazyk určený pro správu dat v relačních databázových systémech. Počítač, který řídí databázi se nazývá systém řízení báze dat nebo DBMS. Při získávání dat z databáze se může použít jazyk SQL k vytvoření tohoto požadavku. DBMS následně zpracuje SQL požadavek, načte požadovaná data a vrátí je. Tento proces vyžádání dat z databáze a následné vrácení výsledků se nazývá databázový dotaz.

SQL je mnohem víc než jen dotazovací nástroj, i když to byl jeho původní účel, tak je získávání dat stále jednou z jeho nejdůležitějších funkcí. SQL se používá k ovládání všech funkcí, které DBMS poskytuje svým uživatelům včetně:

- Definice dat – SQL umožňuje uživateli definovat strukturu a organizaci uložených dat a vztahů mezi uloženými datovými položkami.
- Načítání dat – SQL umožňuje uživateli nebo aplikaci načíst uložené data z databáze a následně je používat.
- Manipulace s daty – SQL umožňuje uživateli nebo aplikačnímu programu aktualizovat databázi přidáním nových dat, odebráním starých dat a úpravou dříve uložených dat.
- Ovládání přístupu – v SQL lze omezit schopnost uživatele načítat, přidávat a měnit data a chránit uložená data před neoprávněným přístupem.
- Sdílení dat – SQL se používá ke koordinaci sdílení dat s uživateli.
- Integrita dat – SQL definuje omezení integrity v databázi, chrání ji od poškození v důsledku nekonzistentních aktualizací nebo při selhání systému.

SQL není ve skutečnosti úplný počítačový, programovací jazyk jako je COBOL, C, C++ nebo Java, poněvadž neobsahuje žádné IF příkazy pro vytváření podmínek, ani žádné GOTO, DO a FOR příkazy pro vytváření cyklů v kódu. Místo toho je SQL jazykem obsahujícím přibližně 40 příkazů specializovaných na úkoly spojené se správou databází. Tyto příkazy mohou být použity i v jiných jazycích, jako je například C nebo PHP, a tím rozšířit jejich vlastnosti o schopnost práce s databází. Alternativně mohou být SQL příkazy explicitně odeslány pro zpracování do systému správy databáze prostřednictvím CLI (Call level interface – rozhraní definující jakým způsobem má program odeslat dotazy do systému řízení báze dat) za užití jazyků C, C++ či Javy nebo prostřednictvím zpráv odesílaných přes počítačovou síť.

SQL není příliš strukturovaný jazyk v porovnání se strukturovanými jazyky jako je C, Pascal nebo Java. Místo toho SQL příkazy připomínají anglické věty, doplněnými pomocnými slovy, díky kterým výsledný dotaz vypadá přirozeně a jeho funkce dává rychle smysl. SQL je nedílnou součástí systému správy databází, jazykem a nástrojem pro komunikaci se systémem báze dat. DBMS je zodpovědné za skutečné strukturování, ukládání a načítání dat v databázi. Přijímá SQL požadavky od ostatních DBMS komponent, z uživatelských aplikací, či z jiných počítačových systémů. SQL má v systému řízení báze dat několik rolí:

- SQL je interaktivní dotazový jazyk – Uživatelé zadávají SQL příkazy do interaktivní aplikace využívající SQL pro načtení dat a jejich následné zobrazení na obrazovce. Tím poskytuje pohodlný a snadno použitelný nástroj pro databázové dotazy.
- SQL je programovací databázový jazyk – Programátoři vkládají SQL příkazy do svých aplikačních programů pro přístup k datům v databázi. Uživateli psané programy a databázové služby (jako například nástroje pro zadávání dat) používají tuto techniku pro přístup k databázím.
- SQL je jazyk pro správu databází – Administrátor databáze odpovědný za správu minipočítače nebo databázového mainframu používá SQL k definování databázové struktury a řízení přístupu k uloženým datům.
- SQL je jazyk typu klient/server – Programy osobních počítačů používají SQL k síťové komunikaci s databázovými servery, které ukládají sdílená data. Tento klient/server styl se stal velmi oblíbený mezi podnikovými aplikacemi.
- SQL je jazyk pro přístup k datům na internetu – Internetové webové servery, které operují s firemními daty a internetovými aplikacemi používají SQL jako standartní jazyk pro přístup k firemním databázím.
- SQL je distribuovaný databázový jazyk – Distribuované systémy pro správu databází používají SQL k distribuci dat přes spoustu připojených počítačových systémů. Software systému řízení báze dat používá v každém systému SQL pro komunikaci s dalšími systémy odesíláním požadavků pro přístup k datům.
  [21]

4 Praktická část

4.1 Vytvoření webu pro správu produktů

Prvním krokem k přípravě tohoto webu je instalace Nette. Framework se může buď stáhnout z oficiálních stránek Nette, či za užití composeru a příkazu:

```
composer create-project nette/sandbox supply
```

Po instalaci je potřeba napojit projekt na databázi. To se provede vyplněním uživatelského jména, hesla a názvu databáze v souboru „app/config/config.local.neon“, kde jsou pro tyto hodnoty připravené příslušené pole. Nyní když je databáze napojená se může vytvořit vzhled administrace. Horní navigace, styly a skripty budou na všech stránkách webu pro správu produktů stejné, čili není potřeba toto vše vkládat několikrát pro každou stránku. Místo toho se dá využít již hotová latte šablona s názvem „@layout.latte“. Jedná se o takovou globální šablonu, jenž se spustí na každé stránce. V této šabloně se pro HTML hlavičku nadefinují jednotlivé meta tagy, kaskádové styly a fonty. Vzhled všech prvků na webu a responsivita budou vycházet z nejnovější čtvrté verze bootstrapu, proto se musí mezi kaskádové styly vložit i styl pro bootstrap. Ihned po konci hlavičky bude mít stránka výpis flashových zpráv. Díky nim uživatel pozná výsledek zpracování formulářů. Flashové zprávy jsou uložené v proměnné „$flashes“. Tato proměnná se skládá z pole s objekty, které obsahují vlastnosti „message“ (text zprávy) a „type“ (námi nadefinovaný typ zprávy). Pro jejich výpis se používá tak zvané „n:makro“, díky kterému se pomocí cyklu „foreach“ vypíší veškeré flashové zprávy v případě, že nějaké existují. Zápis přes n:makra není potřeba, ale oproti běžnému PHP zápisu výrazně šetří místo a dělá kód přehlednějším.
Navigace menu bude zkonstruována pomocí bootstrap tříd, díky čemuž získá pěkný, responsivní vzhled a umožní čitelnost i na menších zařízeních. Mezi jednotlivé položky v menu se vloží základní odkazy potřebné pro aplikaci – správa produktů, objednávek a klientů.

Jelikož bude samotný obsah stránek administrace pokaždé jiný, musí se nyní těsně po navigaci vložit latte blok „{include content}“, do kterého se bude vkládat obsah ostatních obsahových šablon. Nette poté v těchto šablonách pozná, která část kódu se má do bloku content vložit, neboť bude obalená blokem „{block content}“. Pro skripty se musí udělat to samé jako pro obsah, ale obráceně. Neboť budou skripty, které chceme mít na všech stránkách (jako například jQuery a bootstrap), zároveň ale stejně jako na jakémkoliv jiném webu existují skripty, které jsou jen na určitých stránkách. Je samozřejmě možné veškeré skripty dávat do patičky, ale to by časem mohlo zpomalit web. Proto se do patičky @layout.latte „{block scripts}“ vloží veškeré globální skripty pro web a v obsahových stránkách se poté v případě použití javascriptových skriptů vloží za {blok scripts} ještě blok {include parent}. Díky tomu se nejprve vloží skripty z @layout.latte a až poté skripty nadefinované na dané obsahové stránce.

4.1.1 Přihlašování

Tento web se bude skládat pouze z backendové části, do které se půjde dostat za pomocí přihlašovacího formuláře. Pro přihlašování má Nette připravené vlastní funkce. V první fázi je potřeba vytvořit presenter pro tuto stránku. Ten bude obsahovat pouze renderovací metodu „renderDefault()“ a komponentu „createComponentLogin()“ obsluhující formulář pro přihlašování. V renderovací metodě bude probíhat kontrola, zdali je uživatel přihlášen. Pokud ano, nemá cenu mu zobrazovat přihlašovací formulář a rovnou se přesměruje na domovskou stránku administrace. V této komponentě se nejprve musí nadefinovat jednotlivé inputy formuláře a následně jejich validace. Přihlašovací formulář se kromě odesílacího tlačítka skládá z textového inputu pro uživatelské jméno a inputu pro heslo, kde oba tyto inputy jsou povinné. To se v Nette provádí zavoláním metody „setRequired()“ na instanci Nette třídy „Form“, která v tomto frameworku pomáhá s prací s formuláři. O zpracování formuláře se stará metoda „LoginSucceeded()“, jenž se zavolá ihned po submitu tlačítka pro odesílání. Tato metoda se pokusí přihlásit za použití již v základním Nette hotového modelu „UserManager“. Ten v případě jakéhokoliv neúspěchu při přihlašování vrací výjimku, proto je potřeba celou tuto operaci „obalit“ PHP blokem „try“ a „catch“, jenž v případě neúspěchu podchytí výjimku a vypíše uživateli příslušnou chybu, díky které se nepřihlásil. O správu uživatelů se v Nette stará třída „Nette\Security\User“, která se dá v presenterech získat zavoláním metody „$this->getUser()“, čímž se získá objekt této třídy. Pomocí tohoto objektu se dá využít metoda „login()“, která jako parametry přijímá hodnoty z inputu pro uživatelské jméno a heslo a poté se pomocí nich pokusí ve zmíněném modelu přihlásit. Tabulka pro správce této administrace může být pro začátek velmi jednoduchá a bude uchovávat pouze id, uživatelské jméno a důkladně zašifrované heslo.

Tabulka 2 - Struktura tabulky s administrátory

Tabulka - admins		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
username	varchar(80)	Sloupec pro uživatelské jméno
password	varchar(300)	Sloupec pro heslo v zašifrovaném tvaru

Zdroj: Vlastní zpracování

Nyní se v modelu „UserManager“ vykoná SQL dotaz, jenž ověří že daný uživatel existuje, a že zadané heslo je správné. I pro hashování hesel má Nette vlastní třídu plnou užitečných metod, konkrétně třídu „Passwords“. Pomocí její metody „hash()“ se heslo zašifruje a vytvoří se tak dostatečně bezpečný hash. Jeho správnost se následně ověří metodou „verify()“. V případě, že heslo nebo uživatelské jméno nesedí, vyhodí se výjimka s konkrétní chybou, která se zobrazí za pomocí flash zprávy a přesměruje se přes metodu „redirect()“ opět na stránku s přihlašovacím formulářem. V opačném případě vrátí model takzvanou identitu, která v sobě nese ID uživatele, jeho roli a pole s dalšími daty. Nette identitu ukládá do session proměnné, takže je poté kdykoliv možné ověřit uživatelovu roli, či jestli je zrovna uživatel přihlášen. V posledním kroku se přesměruje na úvodní stránku administrace.

```
// Zpracování formuláře pro přihlašování
public function LoginSucceeded($form)
{
    $values = $form->getForm()->getValues();

    // Zkusíme přihlásit uživatele...
    try {
        $this->getUser()->login($values->username, $values->password);
        $this->getUser()->setExpiration('1 day', false);
        $this->flashMessage("Úspěšně přihlášeni.", "success");
        $this->redirect('Admin:');
    }

    catch(NetteSecurityAuthenticationException $e) {
        if ($e->getCode() == 2) {
            $this->flashMessage("Zadali jste špatné heslo.", "warning");
        }
        else {
            if ($e->getCode() == 1) {
                $this->flashMessage("Uživatel neexistuje.", "warning");
            }
        }

        $this->redirect("this”);
    }
}
```

4.1.2 Správa klientů

Po úspěšném přihlášení je nutné nastavit v globálním presenteru – „BasePresenter“ (ten se načítá před spuštěním běžného presenteru) podmínku, která vždy zkontroluje, jestli je uživatel zrovna přihlášený. Toho se docílí metodou „isLoggedIn()“ na instanci objektu třídy Nette/User. V opačném případě se přesměruje zpět na přihlašovací formulář – tím se zabezpečí přístup do administrace, do které může jen člověk s ověřením. Aby byla možná komunikace s databází, tak se musí napojit BasePresenter na jednotlivé modely, čímž presenteru vznikne „závislost“ na daném modelu. Správa objednávek, klientů a produktů bude mít pro přehlednost každá vlastní model. Ty se dají na BasePresenter napojit několika způsoby. Nejlehčí a nejkratší z nich je pomocí anotace „@inject“ u proměnné s typem přístupu public. Po napojení modelu pro klienty je nutné vytvořit samostatnou tabulku pro klienty. Tato tabulka bude uchovávat nejdůležitější informace o každém klientovi.

Tabulka 3 - Struktura tabulky s klienty

Tabulka - clients		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
name	varchar(80)	Název klienta
email	varchar(40)	Email klienta
api_key	varchar(250)	Unikátní, vygenerovaný klíč k identifikaci klienta při užití REST API
status	varchar(15)	Status značící jestli je klient aktivní, blokovaný nebo dlužník
created	datetime	Datum a čas, ve kterém byl klient vytvořen
updated	datetime	Datum a čas, ve kterém byl klient updatován

Zdroj: Vlastní zpracování

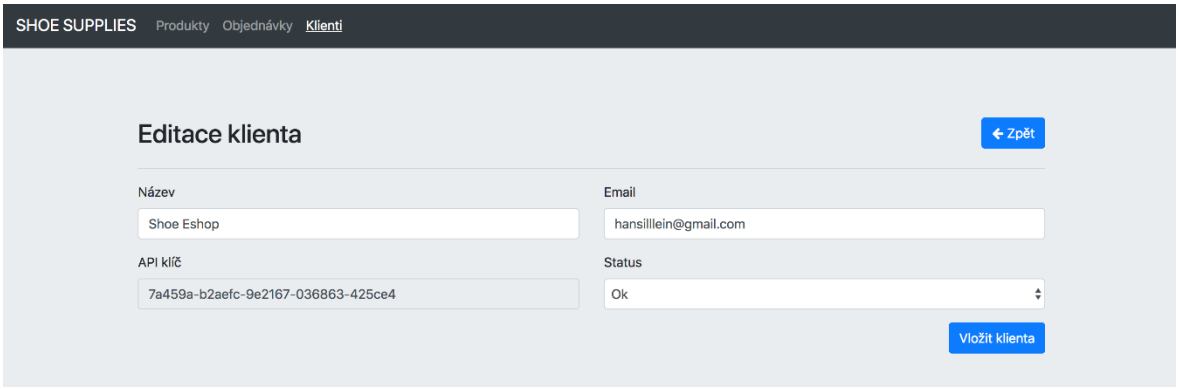
Další krokem po vytvoření tabulky je vytvoření přidávání a editace klientů. To se spravuje pomocí formuláře, takže je nutné v presenteru pro klienty vytvořit komponentu „createComponentEditClient()“ a patřičné renderovací metody pro přidávání a editaci - „renderAddClient()“ a „renderEditClient()“. V komponentě createComponentEditClient() se nastaví jednotlivé inputy pro všechny sloupce tabulky s klienty kromě id, které se nastavuje samo automaticky při vložení nového řádku do tabulky. Po nastavení správné validace pro email a nastavení požadovaných inputů je nutné v této komponentě nastavit, aby při editaci klienta byly inputy předvyplněné jeho hodnotami z databáze. Toho se docílí tím, že bude stránka pro editaci klienta uchovávat informaci o jeho id v URL adrese. V nette se tato věc nastaví jednoduše přidáním parametru do renderEditClient() metody. Nyní stačí přidat do komponenty podmínku, která pomocí kódu „if($this->getParameter("id"))“

zkontroluje zdali se uživatel zrovna edituje nebo ne. V případě editace stačí pak zavolat patřičnou metodu v modelu pro klienty, která vrací řádek uživatele z databáze na základě id a nastavit mu defaultní hodnoty inputů. API klíč klienta se generuje ajaxově a je pouze nutné přes atribut „readonly“ nastavit, aby jeho input při editaci nešel znovu nastavit. Metoda starající se o výsledek vyplněného formuláře se nazývá „EditClientSucceeded()“. Uvnitř této metody probíhá zpracování jednotlivých hodnot inputů a jejich uložení. Zde je opět nutné ověřit jestli se klient přidává nebo edituje a podle toho pak zavolat konkrétní metodu, která klienta v databázi upraví. Při přidání nového klienta se mu na nastavený email pošle zpráva, jenž jej upozorní na přidání do systému a dá mu informaci o jeho unikátním API klíči, který bude potřebovat při využívání funkcí REST API. V emailu také klient uvidí odkaz na stránku s přehledem veškerých REST API funkcí a s detailním popisem jejich fungování. Pro používání nette funkcí na vytváření a odesílání emailů je potřeba za užití „use“ operátoru vložit do klientského modelu třídy „Nette\Mail\Message“ a „Nette\Mail\SendmailMailer“. Při vyplňování základních informací potřebných pro odeslání emailu se musí nastavit i obsah emailu. Ten se přidává za užití metody „setHtmlBody()“ na instanci vložené třídy Nette\Mail\Message. Obsah se může jednoduše vložit jako textový řetězec v parametru této metody, ale nejideálnější způsob je do parametru vložit odkaz na latte šablonu a pomocí metody „renderToString()“ tuto šablonu převést do textové podoby. Díky tomu se může email lépe formátovat pomocí HTML a nemusí se nešikovně psát do uvozovek značících textový řetězec. Po vytvoření, či úpravě klienta se pošle uživatelovi oznámení o úspěchu operace přes flash zprávu a přesměruje se na výpis klientů. Stejně jako pro ostatní renderovací metody, tak i pro renderAddClient a renderEditClient se musí do adresáře Templates přidat patřičné latte šablony. Ty se budou skládat pouze z formuláře nadefinovaného v metodě createComponentEditClient().

Pro generování API klíče je potřeba přidat vedle textového inputu tlačítko a jemu pomocí jQuery přiřadit událost, která po kliknutí na tlačítko spustí jQuery funkci „ajax“. Té je potřeba přiřadit cestu na PHP skript, který se poté ajaxově vykoná. V nette se o zpracování ajaxových požadavků starají v presenterech „handle“ metody, takže v tomto případě je potřeba vytvořit v klientském presenteru metodu „handleGenerateApiKey()“ a přiřadit jí cestu. Uvnitř této metody se nejprve musí zkontrolovat, zdali se jedná o ajaxový požadavek pomocí metody služby zapouzdřující HTTP požadavek „$this->isAjax()“. Pokud se jedná o ajaxový požadavek spustí se metoda v modelu pro správu klientů, která vygeneruje

alfanumerický, unikátní řetězec. Ten se poté přes jQuery vloží do textového pole pro API klíč.

Obrázek 4 - Formulář pro editaci klienta v centrálním skladě bot



Zdroj: Vlastní zpracování

4.1.3 Výpis klientů

Tato stránka se skládá z jednoduchého výpisu jednotlivých klientů. Administrátor webu má možnost základního, ajaxového filtrování mezi klienty a pro větší přehlednost a snížení náročnosti má výpis klientů i stránkování. Výpis klientů je technicky trošku složitější oproti stránce s editací a přidáním klientů, neboť tyto stránky se v podstatě skládaly pouze z formuláře. Nejprve je potřeba klasické vytvoření renderovací metody pro stránky, v tomto případě se metoda jmenuje „renderListOfClients()“. Jelikož má stránka s výpisem klientů filtry a stránkování, je potřeba aby se přenášely potřebné informace v URL adrese za užití GET proměnných. Toho se stejně jako v případě editace klienta docílí vložením parametrů do renderovací metody renderListOfClients(). Na této stránce se pro stránkování přenáší informace o aktuální stránce a pro filtrování se přenáší počet vypsaných klientů na stránce, název klienta a status klienta. V případě budoucího rozšíření filtrace stačí pouze doplnit další parametr do této metody. Pokud se uživatel dostane na tuto stránku, nebude v URL adrese žádný GET parametr, neboť ještě nebylo použito filtrování a uživatel se nachází na první stránce výpisu. První věcí, která se tedy musí nastavit v metodě renderListOfClients() jsou defaultní hodnoty jejích parametrů (pokud již nějakou hodnotu neobsahují), kde číslo

aktuální stránky bude mít hodnotu 1 a ostatní „null“. Po nastavení defaultních hodnot se může přejít k samostatnému předání klientů z presenteru do šablony. Pro získání klientů se musí opět kontaktovat databáze, proto se musí vytvořit nová metoda „getAllClients()“ v modelu pro klienty. Parametry této metody budou stejné parametry co jsou v renderovací metodě renderListOfClients(). Uvnitř getAllClients() se sestavuje SQL dotaz na základě jejich parametrů, kdy se v první fázi nejprve vyberou všichni klienti a poté se dotaz díky podmínkám rozšiřuje a vybírají se jen konkrétní klienti splňující podmínky nastaveného filtrování. Stránkování se řeší pomocí SQL funkce OFFSET, kdy se například pro druhou stránku výpisu při nastavení pěti klientů na stránku vyberou záznamy 6 – 10. Po sestavení celého SQL vrátí tato metoda objekt zvaný „Selection“, který se v šabloně vypíše přes cyklus foreach.

```
// Vrátí klienty
public function getAllClients($clients_per_page, $actual_page, $name,
$status)
{
    $clients = $this->database->table(self::TABLE_CLIENTS);

    // Vyfiltrujeme výsledky dle názvu...
    if ($name AND $name != "")
    {
        $clients->where("LOWER(" . self::COLUMN_NAME . ") LIKE ?",
       "%" . mb_strtolower($name, "UTF-8") . "%");
    }

    // Vyfiltrujeme pouze klienty s daným statusem
    if ($status AND $status != "all")
    {
        $clients->where(self::COLUMN_STATUS . " = ?", $status);
    }

    // Vrátíme všechny výsledky, či jen část...
    if ($actual_page)
    {
        $offset = ($actual_page - 1) * $clients_per_page;
        $clients->limit($clients_per_page, $offset);
    }

    return $clients;
}
```

Pro co největší komfort administrátora při filtrování klientů je tato funkce řešena ajaxově, aby se stránka nemusela při každé změně refreshovat. Jelikož se ale stále jedná o formulář, je proto opět nutné vytvořit novou komponentu –

„createComponentFilterClients()“. V ní se klasicky nadefinují jednotlivé inputy, které filtrování klientů má, konkrétně tedy počet klientů na stránku, název klienta a status. Jako defaultní hodnoty se jim nastaví hodnoty v GET proměnných, pokud tedy nějaké existují. V metodě zpracovávající tento formulář - „filterClientsSucceeded()“ přichází ke změně oproti předchozím situacím. Stránka se musí přes ajax překreslit novými daty vyhodnocenými z formuláře. To se dá v nette udělat několika způsoby, jedním z nich jsou tak zvané snippety, kdy se v latte šabloně obalí potřebné HTML, které se bude měnit, pomocí latte bloku {snippet}. V metodě, která má poté ajaxově něco překreslit se pak musí znovu poslat aktualizovaná data do šablony a zavolat metoda „redrawControl()“, která překreslí veškerá data novými uvnitř {snippet} bloků. V metodě filterClientsSucceeded() se tedy znovu kontaktuje model a za použití funkce getAllClients() se opět vytvoří nové SQL s parametry získanými z hodnot navolených z formuláře pro filtrování. Tyto data se poté pošlou do šablony a ajaxově překreslí.

Samotný výpis klientů je tedy řešen cyklem foreach, kde jsou jednotliví klienti vypisováni v řádcích tabulky. Každý řádek obsahuje tyto informace: název klienta, jeho unikátní id, email, API klíč, status klienta a nezaplacený dluh. Na konci každého řádku jsou dvě akce, které může administrátor využít. Těmito funkcemi jsou editace klienta a smazání a jsou symbolizovány ikonkami koše a tuštičky. Editace klienta je pouhý odkaz na již hotovou renderovací metodu renderEditClient() s patřičným id klienta. O smazání klienta se stará ajaxový formulář, jenž funguje podobně jako formulář pro filtrování. Ten je ukryt v modálním, bootstrapovským okně, které se spustí pomocí jQuery při události kliknutí na košík. Jelikož jsou data na této stránce neustále překreslována a HTML se zde dynamicky mění, je potřeba událost na kliknutí řešit přes jQuery funkci „on()“, a ne přes „click()“. Zároveň se musí v parametru pro selektor zadat kompletní cesta ke košíku, začínající od začátku dynamicky překreslovaného HTML, u kterého je použit latte blok snippet. Formulář obsahuje pouze tlačítko pro potvrzení smazání klienta a skrytý input obsahující id tohoto klienta, jenž se nastaví automaticky v události při kliknutí na košík jQuery metodou „val()“. Komponenta v presenteru starající se o mazání klientů se nazývá „createComponentDeleteClient()“ a díky metodám nette objektu třídy Form - „addHidden()“ a „addSubmit()“, dává frameworku informaci, že tento formulář obsahuje pouze tlačítko a skrytý input. Metoda zpracovávající tento formulář „deleteClientSucceeded()“ obsahuje funkci „deleteOrdersByClient()“, jenž smaže jeho řádek v tabulce klientů a zároveň i veškeré

relace, které klient má v tabulce s objednávkami. Po vykonání této funkce je potřeba opět překreslit stránku novými daty. Toho se docílí úplně stejným stylem jako u filtrování, čili se nejprve za pomoci metody getAllClients() získají klienti. Ti se následně odešlou do šablony a metodou redrawControl() se vše ajaxově v mžiku překreslí bez obnovení stránky.

4.1.4 Správa produktů

Stránka správy produktů se velmi podobá správě uživatelů, jde v podstatě pouze o stránku s formulářem, a proto je i kód velice podobný. V první řadě je potřeba vytvořit presenter ProductPresenter, neboť už se nepracuje s klienty ale s produkty. Uvnitř tohoto presenteru se musí nadefinovat renderovací metody pro přidávání a editaci produktů – „renderAddProduct()“ a „renderEditProduct()“, která má stejně jako metoda renderEditClient() parametr přenášející informaci o aktuálně editovaném produktu. Data produktů jsou uchována ve třech tabulkách: products, variants a categories. Kde tabulka products schraňuje veškerá data o produktu a tabulka variants zase o jednotlivých variantách. Mezi těmito tabulkami jsou nastavené relace pomocí cizích klíčů, zamezujícím mimo jiné například smazání konkrétního produktu, pokud je k němu přiřazená varianta. Tyto tři tabulky vypadají následovně:

Tabulka 4 - Struktura tabulky s produkty

Tabulka - products		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
name	varchar(80)	Název produktu
html_name	varchar(80)	Název produktu bez interpunkce a mezer
description	text	Detailní popis produktu
product_code	varchar(8)	Unikátní označení produktu
price	float	Cena produktu

sex	varchar(1)	Pohlaví pro které je produkt určen. Možné hodnoty: M, Ž
category_id	int	Cizí klíč uchovávající primární klíč tabulky „categories“
image	varchar(120)	Absolutní cesta k obrázku produktu
image_thumb	varchar(120)	Absolutní cesta k náhledovému obrázku produktu
created	datetime	Datum a čas, ve kterém byl produkt vytvořen
updated	datetime	Datum a čas, ve kterém byl produkt updatován

Zdroj: Vlastní zpracování

Tabulka 5 - Struktura tabulky s variantami

Tabulka - variants		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
product_id	int	Cizí klíč uchovávající primární klíč tabulky „products“.
name	varchar(30)	Název varianty produktu
supply	int	Nezáporná číselní hodnota značící počet skladových zásob dané varianty. Defaultní hodnota nastavená na 0

Zdroj: Vlastní zpracování

Tabulka 6 - Struktura tabulky s kategoriemi

Tabulka - categories		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
name	varchar(30)	Název kategorie

Zdroj: Vlastní zpracování

O formulář starající se o přidávání a editaci produktu a jeho variant se v ProductPresenteru stará komponenta „createComponentEditProduct()“. Její kód je velmi podobný komponentě createComponentEditClient(), čili jsou zde opět zaznamenané jednotlivé inputy odpovídající sloupcům v tabulkách pro produkty a v případě, že se edituje produkt, tak se jednotlivým inputům přednastaví defaultní hodnoty. Toto všechno platí na všechny sloupce tabulek až na inputy pro varianty. Jelikož každý produkt může mít libovolný počet variant, nedá se přesně v komponentě nadefinovat kolik textových polí budou varianty zrovna mít. Tudíž se tato informace v komponentě vynechá a výsledné varianty se vytáhnou za pomoci proměnné $_POST, ve která se běžně ukládají výsledné hodnoty formulářů. Metoda zpracovávající formulář pro přidávání a editaci produktů se nazývá „EditProductSucceeded()“. V ní se v závislosti na typu operace vykoná buď funkce „editProduct()“, či „addProduct()“.

Mezi hodnotami získanými z formuláře může být i obrázek produktu, který není mezi povinnými inputy, proto se musí při přidávání/editaci produktu zkontrolovat přes funkci „isOk()“, jestli vůbec existuje a zdali se správně načetl. Pokud vrátí tato funkce hodnotu TRUE, provede se metoda „addImageToProduct()“. Ta má jako parametry uploadovaný obrázek, název obrázku a název náhledového obrázku. Poslední dva parametry jsou díky operátoru „&“ referencemi. Díky tomu se těmto dvěma předaným proměnným nastaví v metodě addImageToProduct() hodnoty a ty jim pak zůstanou i mimo tuto funkci. Pro práci s obrázky má nette třídu „Nette\Utils\Image. Pomocí objektu této třídy se dají s obrázkem dělat různé úpravy. U většího obrázku bude stačit pouze nastavení defaultních rozměrů, přes

funkci „resize()“ s parametry „1920, 1080, Image::FIT“. Díky tomu bude mít větší varianta obrázku maximálně rozměry 1920x1080 a uživatel tak nemusí kontrolovat, jak velký obrázek zrovna nahrává, poněvadž se to na straně serveru samo vyřeší. Náhledový obrázek se zmenší přes funkci resize() s parametry „300, 300, Image::EXACT“. Menší obrázek tak bude mít pokaždé rozměry 300x300 (co přesáhne tyto rozměry, to se ořízne). Takto zmenšený obrázek se za užití funkce „sharpen()“ malinko doostří pro lepší vzhled a uloží společně s velkým obrázkem za užití funkce „save()“ do adresáře pro obrázky produktů.

Uložené varianty se mohou získat klasicky přes $_POST, ale data v nich by se musely ještě důkladně ošetřit, aby nedošlo k SQL injection. Druhou možností jak varianty získat je pomocí nette metody „getHtttpData()“, která přijímá jako parametr název inputu (v tomto případě se jedná o pole inputů „variant_name[]“ značící názvy variant a „variant_supply[]“ značící počet variant). Hodnoty těchto polí vrátí metoda getHtttpData() v očištěném a zabezpečeném tvaru. Podobně jako u ostatních hodnot produktu, tak i u variant jsou v modelu pro produkty nadefinované metody pro přidávání a editaci – „addVariants()“ a „editVariants()“. Ty mají jako přijímají jako parametry id produktu, kterému se mají varianty přiřadit a pole názvů variant a jejich skladové kusy. Uvnitř funkce addVariants() probíhá pouze SQL příkaz INSERT přidávající novému produktu varianty specifikované v parametru. Funkce editVariants() projíždí veškeré varianty v poli variant a u každé varianty nejprve přes SQL příkaz SELECT zkontroluje, jestli již náhodou není v databázi u tohoto produktu. V případě že se tato varianta u produktu nachází, tak se provede SQL příkaz UPDATE. V opačném případě se použije příkaz INSERT. V obou případech se ukládá id varianty do vytvořeného pomocného pole. Na závěr se smažou veškeré varianty u daného produktu, které v tomto pomocném poli nejsou.

```
// Smažeme varianty, co nejsou v poli...
if (!empty($used_ids))
{
    $this->database->query("
      DELETE FROM " . self::TABLE_VARIANTS . "
      WHERE " . self::COLUMN_VARIANTS_ID . "
      NOT IN (" . implode(",", $used_ids) . ") AND
      " . self::COLUMN_VARIANTS_PRODUCT_ID . " = ?", $id);
}
```

V závěru metody EditProductSucceeded() se odešle administrátorovi flash zpráva s oznámením úspěšné editace produktu a přesměruje se zpět na výpis produktů. Latte šablona

této stránky se skládá z formuláře definovaného v komponentě createComponentEditProduct(), ve kterém jsou v sekci s variantami vypsány veškeré varianty tohoto produktu za užití latte příkazu:

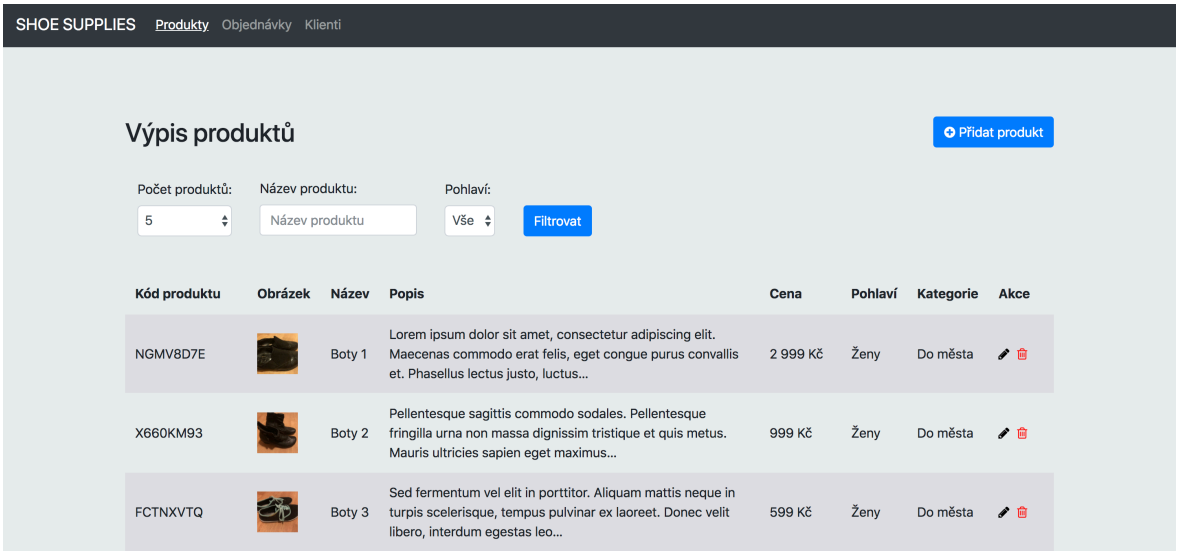
```
{foreach
      $product->related('variants.product_id')->order("variants.id")
      as $variant
}
```

Příkaz v cyklu foreach projede objekt třídy Selection, ve kterém jsou uchovány veškeré informace spojené s editovaným produktem. Za užití metody „related()“ provede nette samo SQL příkaz JOIN na tabulku variants, ve kterém je nadefinována relace na cizí klíč „product_id“. Každá varianta se skládá ze dvou inputů, kdy jeden z nich slouží pro název a druhý pro počet kusů. Vedle těchto dvou inputů se nachází ikonka košíku, na kterou je přiřazena jQuery událost při kliknutí. Při spuštění této události se provede pouze jQuery funkce „remove()“, jenž celý řádek varianty odstraní. Samotné smazání varianty z databáze se řeší až pomocí PHP, kde se z formulářem vrácených variant pozná, která chybí. Pro přidání nové varianty se nachází ve formuláři speciální tlačítko, kterému je opět přiřazena jQuery událost na kliknutí přes metodu click(). Po kliknutí na toto tlačítko se poté za užití jQuery metody „append()“ vloží na konec třídy, jenž je nadefinovaná v selektoru této metody – dva textové inputy a košík pro novou variantu.

4.1.5 Výpis produktů

Výpis produktů je opět velmi podobný výpisu klientů (jak vzhledově, tak uvnitř kódu), poněvadž není moc potřeba dělat tuto věc jiným způsobem. Opět se zde využívají filtry, tentokrát ale pro: počet produktů na stránku, název produktu a pohlaví pro které je produkt určen. Toto lze opět v budoucnu rozšířit i o další filtry, ale prozatím tyto základní stačí. Zpracování filtrování a stránkování funguje v ProductPresenteru úplně stejným způsobem jako v ClientsPresenteru. Ve výpisu produktů se nachází hodnoty pro kód produktu, obrázek (zde se zobrazuje náhledová zmenšenina obrázku, pro rychlejší načítání stránky), název produktu, popis, cena, pohlaví a kategorie produktu. Ve sloupci „Akce“ se opět zobrazují ikonky tuštičky a košíku pro editaci a mazání produktu.

Obrázek 5 - Výpis produktů v centrálním skladě bot



Zdroj: Vlastní zpracování

4.1.6 Výpis objednávek

Objednávky se v administraci skládají pouze z jednoduchého výpisu podobnému výpisu klientů a produktů. Jelikož se jedná opět o jinou entitu, tak se musí pro pořádek vytvořit odpovídající presenter a model. Data objednávek jsou uchována ve dvou tabulkách: orders a orders_detail. Tabulka orders schraňuje základní informace jako je cena objednávky, či informace jestli je objednávka zaplacená, či ne. Na základě této informace se pak počítá klientům ve výpisu klientů dluh. Informace o klientovi, kterému tato objednávka patří je uchována ve sloupci „clients_id“, který je propojen s primárním klíčem, díky relaci s tabulkou clients. Jelikož se v API této aplikace nachází i metoda umožňující klientovi vrátit libovolné produkty, zaznamenává se tento příznak do sloupce „type“ buď pod hodnotou „Vrácené zboží“, či v běžném případě jako „Objednávka“. Tabulka pro objednávky vypadá následovně:

Tabulka 7 - Struktura tabulky s objednávkami

Tabulka - orders		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
clients_id	int	Cizí klíč uchovávající primární klíč tabulky „clients“
price	float	Celková cena objednávky včetně DPH
price_without_dph	float	Celková cena objednávky bez DPH
type	varchar(30)	Typ značící jestli se jedná o objednání, či vrácení zboží
status	varchar(30)	Status značící jestli je objednávka zaplacená, či jestli se čeká na zaplacení
date	datetime	Datum a čas přijetí objednávky

Zdroj: Vlastní zpracování

Tabulka orders obsahuje pouze základní informace každé objednávky, ale již neobsahuje co konkrétně si klient objednal. K těmto účelům slouží tabulka „order_detail“. Vyjma sloupce „supply“ (ta zaznamenává objednaný počet kusů dané varianty) jsou veškeré sloupce klíče s patřičnými relacemi na příslušné tabulky. Každý řádek v této tabulce obsahuje informaci o jakou variantu se jedná, kterému produktu tato varianta patří a s jakou objednávkou je tento detail objednávky spojen.

Tabulka 8 - Struktura tabulky s detaily objednávek

Tabulka – order_detail		
Název sloupce	Datový typ	Význam
id	int	Unikátní primární klíč s nastavenou vlastností „AUTO_INCREMENT“
order_id	int	Cizí klíč uchovávající primární klíč tabulky „orders“
product_id	int	Cizí klíč uchovávající primární klíč tabulky „products“
variant_id	int	Cizí klíč uchovávající primární klíč tabulky „variants“
supply	int	Počet objednaných kusů této varianty

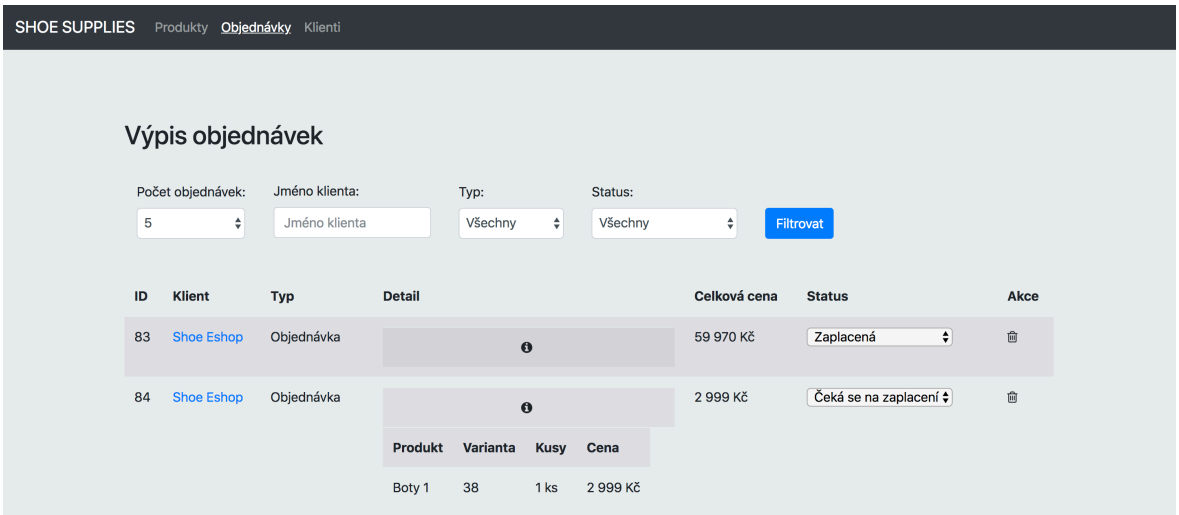
Zdroj: Vlastní zpracování

Samotný výpis objednávek se skládá opět ze stránkování a filtrování, ve kterém se dá zpřehlednit výpis dle jména klienta, typu objednávky a dle stavu zaplacení. Výpis je sestaven z tabulky, kde každý řádek reprezentuje jednu objednávku, či vrácené zboží. Objednávky mohou obsahovat i několik desítek položek a kvůli nim by pak řádky objednávek mohly být zbytečně vysoké na výšku. Proto je vhodnější mít položky v tabulce defaultně schované přes CSS vlastnost „display: none;“ a nechat na administrátorovi webu, aby si sám rozklikl detail objednávky sám. Zobrazování a schovávání detailů objednávek se řeší pomocí jQuery funkce on(“click“), neboť se opět díky filtrování jedná o dynamické HTML prvky. Uvnitř této funkce probíhá díky jQuery metodě „toggle()“ schovávání a zobrazování detailu po každém kliku na ikonku detailu.

Sloupec v tabulce pro výpis objednávek řešící, zdali je objednávka zaplacená, či jestli se čeká na zaplacení je řešená pomocí HTML inputu pro výběr hodnot select. Při úpravě

hodnoty tohoto selectu se poté zavolá pomocí ajaxu handle metoda - „handleChangeOrderStatus()“, v níž se volá funkce v modelu pro objednávky, jenž díky SQL příkazu updatuje hodnotu tabulky orders ve sloupci status. Aby si byl administrátor jistý, že změna vskutku proběhla, objeví se poté po úspěšném dokončení ajaxové metody vedle select inputu zelená ikonka značící úspěch. Tato funkcionalita by šla vyřešit i pomocí klasického formuláře, kdy by se po změně hodnoty v selectu obnovila stránka a vykonalo se klasické zpracování formuláře. Ale tento způsob by mohl působit rušivě pro uživatele, jelikož samotný výpis je již řešen ajaxově a vše se v něm načítá za běhu a bez nějakého rušení, proto je vhodnější vyřešit ajaxovým způsobem i tuto operaci. Na konci řádku se nachází klasicky ikonka košíku, jako i v ostatních výpisech na webu. Při kliku na ikonku se přes ajax spustí handle metoda přiřazená této operaci a ta smaže data objednávky i její relace.

Obrázek 6 - Výpis objednávek v centrálním skladě bot



Zdroj: Vlastní zpracování

4.2 Vytvoření REST API rozhraní pro web se správou produktů

REST API rozhraní bude fungovat (stejně jako i jednotlivé výpisy produktů, klientů a uživatelů) na své vlastní stránce s vlastními funkcemi, proto je potřeba vytvořit presenter s názvem „ApiPresenter“. Do něj se na začátek presenteru musejí naimplementovat public vlastnosti, které budou v budoucnu potřeba ve všech částech presenteru. Mezi tyto vlastnosti patří stavový kód, chybová zpráva, API klíč klienta a řádek z tabulky clients obsahující

informace o klientovi. Jednotlivé metody API, které mohou klienti využívat mají společné zabezpečené ověřování. Kód starající se o ověřování se může vložit na začátek jednotlivých renderovacích metod každé API funkce, ale nejideálnější je vytvořit v presenteru tak zvanou „startup()“ metodu, která se zavolá ihned po vytvoření presenteru, takže se spustí před renderovací metodou. Tímto zápisem není potřeba vkládat jednu věc opakovaně, a díky tomu je v kódu větší pořádek.

Klienti mají dle návrhu pro správné odeslání požadavku za úkol povinně odesílat v hlavičce „X-Auth-Token“ svůj vygenerovaný API klíč, jenž jim přišel emailem. Prvním krokem v metodě startup() tedy bude jeho získání. Pro práci s HTTP požadavky má nette svou třídu plnou užitečných funkcí – „Nette\Http\Request“. Po jejím vložení do ApiPresenteru a vytvořením objektu této třídy se dá přes metodu „getHeader()“ získat hodnota hlavičky uvedené v parametru (v tomto případě X-Auth-Token). První fází ověření, že API využívá oprávněná osoba tedy bude kontrola, jestli vůbec nějaká hlavička X-Auth-Token přišla. V opačném případě se nadefinované public proměnné „status_code“ nastaví hodnota 401 (značící chybu při ověřování) a proměnné „error_message“ se přiřadí textový popis této chyby. Pokud se v hlavičce HTTP požadavku API klíč nachází, tak je potřeba jej ověřit a najít jej v databázi v tabulce s klienty. K tomuto účelu slouží metoda v modelu clients s názvem – authenticateClient(). Ta na základě parametru s API klíčem vykoná SQL příkaz, za pomocí kterého najde klienta s daným klíčem a vrátí selection objekt s řádkem konkrétního klienta. Selection se poté uloží do již vytvořené public proměnné $client, poněvadž údaje o klientovi se budou určitě v budoucnu hodit a při nastavení public proměnné ve startup() metodě, je pak možné tuto proměnnou využívat kdekoliv v kódu presenteru. V případě, že klient s daným API klíčem neexistuje, tak se opět do public proměnných status_code a error_message přiřadí hodnoty značící neúspěch při autentizaci.

Na závěr metody startup() probíhá kontrola, jestli ověření klienta proběhlo v pořádku a zdali je stavový kód rovný jeho defaultní hodnotě 200. Pokud podmínka není splněná, tak se odešle do šablony chybový výpis v JSON formátu a pomocí nette metody „sendTemplate()“ se zamezí zavolání konkrétní renderovací metody a rovnou se zavolá šablona. Klient díky tomu uvidí, co přesně udělal špatně a co má přesně ve svém kódu upravit, aby se chybě při příštím volání vyvaroval. Pokud má public proměnná uchovávající stavový kód jako svoji hodnotu stále 200, zavolá se konkrétní renderovací metoda vykonávající klientem zvolenou operaci.

4.2.1 Metoda pro získání produktu

První důležitou metodou v REST API rozhraní je získání veškerých informací o konkrétním produktu. K tomu bude sloužit metoda s názvem – „get-product“. Ta pomocí GET požadavku obsahujícím v parametrechech kód hledaného produktu a bool hodnotu označující, jestli mají být ve vráceném výpisu i varianty produktu s informacemi o jejich skladových zásobách – vrátí v těle zprávy odpovědi vypsané kompletní informace nacházející se v databázi v přehledném JSON formátu, z kterého si pak klient může vytáhnout potřebná data. Veškeré informace ohledně REST API metod a jak je používat jsou zaznamenané na speciální stránce, o které se klient dozví v emailu odeslaném při jeho vložení do systému. Na této stránce jsou pro metodu get-product mimo jiné tyto informace:

URL požadavku

```
GET /api/get-product
```

Tabulka 9 - Parametry požadavku metody get-product

Parametry požadavku		
Název	Typ	Popis parametru
product_code	string	Kód hledaného zboží
with_variants	bool	V případě hodnoty true budou vrácené informace o produktu obsahovat i informace o jeho veškerých variantách

Zdroj: Vlastní zpracování

Tabulka 10 - Parametry odpovědi get-product

Parametry odpovědi		
Název	Nabývané hodnoty	Popis parametru
product_code	NGMV8D7E	Kód zboží
name	Boty 1	Název produktu
html_name	boty-1	Název produktu v osekaném tvaru bez háčků a čárek
description	Lorem ipsum dolor sit amet...	Popis produktu
price	2999	Velkoobchodní cena produktu
image	http://www.supply.honza-jahoda.cz/products/1821195d372119f6c1d669fc3998ddeb.jpg	Cesta k obrázku produktu
image_thumb	http://www.supply.honza-jahoda.cz/products/1821195d372119f6c1d669fc3998ddeb-thumb.jpg	Cesta k náhledovému obrázku produktu
category	Do města/Sandály/Do hor	Kategorie produktu
sex	Muži/Ženy	Pohlaví, pro které je produkt určen
variants	[název varianty => počet ks]	Varianta produktu v poli ve tvaru [název varianty => počet ks]

Zdroj: Vlastní zpracování

Pro vytvoření jednoduché GET metody get-product stačí ve zkratce vytvořit renderovací metodu „getProduct()“, která v šabloně vypíše v JSON formátu daný produkt specifikovaný v parametru a jeho varianty (v případě, že v parametru with_variants se bude nacházet hodnota TRUE). Pro získání produktu dle produktového kódu stačí naprogramovat metodu v již vytvořeném modelu pro produkty, která jej vytáhne z databáze v selection objektu. Poté je potřeba dostat tento objekt do tvaru určeném ve specifikaci odpovědi a převést výsledné pole produktu do JSON formátu za užití PHP funkce „json_encode()“ a tento JSON následně vypsat v šabloně. Ve výpisu JSONU v šabloně se nesmí zapomenout,

aby se jako typ zdroje uvedl „application/json“, a ne klasický: „text/html“. V nette pro toto nastavení stačí pouze na začátek šablony vložit tento latte kód:

```
{contentTypeapplication/json}
```

4.2.2 Metoda pro získání specifických produktů

Metoda „get-specific-products“ umožňuje vypsání všech produktů uložených v systému. Stejně jako v případě metody get-product i tato metoda používá pro vzdálenou komunikace dotazovací metodu GET. Produkty se dají i filtrovat, aby klient získal zboží dle jeho potřeb. Klient může filtrovat produkty dle názvu, velikosti bot, pohlaví pro které jsou boty určeny a skladové dostupnosti. Zároveň může klient určit dle jakého sloupce produkty řadit a může i ovlivnit počet vypsaných produktů. Tyto všechny vlastnosti filtru jsou uložené stejně jako u výpisu produktů jako parametry renderovací metody. Funkce pro vyfiltrování produktů getSpecificProducts() již v aplikaci existuje, díky stránce s výpisem produktů, takže je díky tomu tato API metoda z větší části hotová. Po vytvoření renderovací metody „renderGetSpecificProducts()“ se tedy pouze zavolá funkce getSpecificProducts(), která stejně jako na stránce s výpisem vrátí v objektu selection veškeré produkty i s jejich variantami, které projdou filtrovacími podmínkami. V posledním kroku je opět potřeba vytvořit ze selectionu produktů datový typ pole a to převést přes funkci json_encode() na řetězec v JSON formátu. Ten se následně musí opět odeslat šabloně, ve které se vypíše. V dokumentaci tohoto REST API se klient o metodě get-specific-products doví:

URL požadavku

```
GET /api/get-specific-products
```

Tabulka 11 - Parametry požadavku metody get-specific-products

Parametry požadavku		
Název	Typ	Popis parametru
product_per_page	int	Počet vrácených produktů
actual_page	int	Offset hodnota. Pokud se například zadá počet produktů 5 a offset 10 => vypíšou se produkty 11-15
name	string	Název produktu
size	int	Velikost bot
sex	string	Pohlaví, pro které je produkt určen
available	bool	Pokud TRUE, vypíše pouze produkty, které jsou skladem
order	string	Uvádí, dle čeho produkty řadit. Možné hodnoty: name, size

Zdroj: Vlastní zpracování

Parametry odpovědi jsou totožné jako u REST API metody get-product. Opět se v JSONU nacházejí jednotlivé vlastnosti produktu, jako je třeba název produktu, či cesta k obrázku.

4.2.3 Metoda pro objednání produktu

Tato POST metoda s názvem „send-order“ – získá z pole POST jednotlivé objednané produkty a jejich konkrétní varianty a následně tyto data zaznamená v databázi v tabulce orders a orders_details a na závěr odešle klientovi email se shrnutím jeho objednávky. V REST API dokumentaci se k této metodě klient doví:

URL požadavku

```
POST /api/send-order
```

Tabulka 12 - Parametry požadavku metody send-order

Parametry požadavku		
Název	Typ	Popis parametru
product_code	string	Unikátní kód produktu
variants	array	Varianta produktu v poli ve tvaru [název varianty => počet ks]

Zdroj: Vlastní zpracování

Tabulka 13 - Parametry odpovědi metody send-order

Parametry odpovědi		
Název	Nabývané hodnoty	Popis parametru
status	Success/Fail	Vrací informaci, zdali akce proběhla v pořádku, či nikoliv

Zdroj: Vlastní zpracování

První akcí v nově vytvořené renderovací metodě „renderSendOrder“() bude kontrola, jestli nějaká POST data vůbec existují. K ověření zdali je pole prázdné se v PHP používá funkce „empty()“. V případě, že je pole s POST daty prázdné, vypíše se klientovi v JSON formátu informace o neodeslání dat správnou formou. Jinak se spustí v modelu „orders“ metoda „addOrder()“ jenž vykoná uložení objednávky. Vnitřek této metody je obalen bloky try a catch, které v případě jakékoliv chyby ukončí proces ukládání objednávky a zaznamenaná chyba se pak následně rovnou zobrazí v JSON výpisu klientovi. V bloku try se nejprve uloží do tabulky orders nový záznam, ve kterém se vyplní údaje co se zatím o objednávce ví, čili id klienta, status objednávky (Čeká se na zaplacení) a datum. Výsledná cena objednávky se musí propočítat v každém cyklu při projíždění POST dat. Pole objednaných produktů z POST proměnné se projíždí pomocí cyklu foreach, kde se v každém cyklu uloží do tabulky order_detail záznam o objednané variantě a počtu kusů. V případě, že by během projíždění pole a ukládání objednaných variant nastala z jakéhokoliv důvodu chyba, zůstaly by v databázi nekompletní data, které by mohly činit potíže (například ve výpisu objednávek). Proto je zde potřeba využít tak zvaných SQL transakcí, které po

úspěšném provedení metody addOrder() potvrdí veškeré SQL příkazy příkazem „commit()“. Zároveň v případě jakékoliv chyby dokáží transakce vrátit databázi do stavu před jejím provedením pomocí příkazu „rollback()“, který se nachází v bloku catch. Díky tomu se tedy databáze naplní pouze požadovanými daty. V průběhu projíždění objednaných produktů je potřeba ohlídat, že klient neobjednává víc skladových zásob než ve skutečnosti jsou na skladu (díky REST API metodě get-product a get-specific-products by klient měl moc dobře vědět kolik zásob jednotlivé varianty mají), dále se musí ohlídat, že objednaný produkt a varianta existují. V opačném případě se musí metoda ukončit a vyhodit výjimka, která předá JSONU konkrétní chybu, aby klient věděl co se nepovedlo. Po ukončení cyklu zbývá ještě přes SQL příkaz UPDATE uložit nyní již známou kompletní cenu objednávky. Při úspěchu celé této operace se klientovi odešle email potvrzující objednávku a dále se v něm vypíší jednotlivé objednané položky. Data ve výpisu jsou obsažená v tabulkách orders, order_detail, products a variants, ale díky nastaveným databázovým relacím stačí pouze do latte šablony odeslat selection objekt s konkrétní objednávkou (řádek z tabulky orders) a poté za užití metod related() a ref() využít nastavených relací a získat data z ostatních tabulek. Výpis tedy v latte šabloně vypadá tedy nakonec takto jednoduše:

```
<h2>Souhrn objednávky</h2>
<table>
   <thead>
      <tr>
         <th>Název produktu</th>
         <th>Počet</th>
         <th>Cena<br>bez DPH</th>
         <th>Cena s<br>DPH</th>
         <th>Cena<br>celkem s DPH</th>
      </tr>
   </thead>
   <tbody>
    {var $order_details = $order->related('order_detail')}
    {foreach $order_details as $order_detail}
    <tr>
      {var $product = $order_detail->ref('products', 'product_id')}
      {var $variant = $order_detail->ref('variants', 'variant_id')}
      <td>{$product->name} - {$variant->name} </td>
      <td>{$order_detail->supply}</td>
      <td>
            {number_format(
                  round(
                        $product->price - (($dph / (100 + $dph)) *
                        $product->price)), 0), 0, ',', ' '
                  )
            } Kč
      </td>
```

```
      <td>{number_format(round($product->price, 0), 0, ',', ' ')} Kč</td>
      <td>
      {number_format(round($order_detail->supply * $product->price),
      0, ',', ' ')} Kč
      </td>
    </tr>
    {/foreach}
  </tbody>
</table>
```

Na závěr renderSendOrder() se opět do latte šablony odešle JSON s potvrzení úspěchu REST API metody a s datem dokončení této operace.

4.2.4 Metoda pro vrácení produktu

Tato metoda s názvem „return-product“ projíždí pole produktů, které chce klient vrátit centrálnímu skladu, uloženém v těle požadavkové metody PUT. Tvar pole je totožný s polem pro odeslání objednávky, čili „array(product_code => array(název varianty => počet ks))“, a dokonce se oboje data zaznamenávají do stejných databázových tabulek orders a order_detail, pouze vrácení produktů má ve sloupci „type“ příznak „Vrácené zboží“ místo „Objednávka“. Ostatní věci ale zůstávají stejné, takže se poté dá jednoduše zjistit, co všechno klient vrátil a kolik musí za vrácené zboží zaplatit. V REST API dokumentace centrálního skladu bot jsou o metodě return-product vypsané tyto informace:

URL požadavku

```
PUT /api/return-product
```

Tabulka 14 - Parametry požadavku metody return-product

Parametry požadavku		
Název	Typ	Popis parametru
product_code	string	Unikátní kód produktu
variants	array	Varianta produktu v poli ve tvaru [název varianty => počet ks]

Zdroj: Vlastní zpracování

Tabulka 15 - Parametry odpovědi metody return-product

Parametry odpovědi		
Název	Nabývané hodnoty	Popis parametru
status	Success/Fail	Vrací informaci, zdali akce proběhla v pořádku, či nikoliv

Zdroj: Vlastní zpracování

Pro vrácení produktů je potřeba vytvořit renderovací metodu „renderReturnProduct“, ve které se stejně jako u renderSendOrder nejprve kontroluje, jestli klient pro volání použil správný typ dotazovací metody, v tomto případě PUT. Pokud kontrola proběhla v pořádku, zavolá se metoda v modelu orders, která se nazývá „returnProduct()“. Ta je podobně jako addOrder() obalená bloky try a catch pro zachycení výjimek a uvnitř opět probíhá cyklus foreach, který projíždí jednotlivé produkty pro vrácení a zaznamená informace o variantách do tabulky order_detail. Jediným rozdílem oproti addOrder() je kontrola v průběhu cyklu, zdali klient nevrací více zboží než si vůbec přes toto REST API objednal. K tomu je potřeba znát:

- kolik variant daného produktu chce klient vrátit
- kolik variant daného produktu si klient celkem již objednal
- kolik variant daného produktu klient již vrátil

První hodnotu (počet kusů varianty produktu, které chce klient vrátit) posílá klient jako informaci v těle požadavku. Zbylé dvě hodnoty se musejí vytáhnout z databáze. K tomu slouží metoda „getCountOfVariant()“, jenž jako parametry přijímá id klienta, id produktu id varianty a řetězec nabývající hodnot „Objednávka“ nebo „Vrácené zboží“ pro typ objednávky. Uvnitř této funkce se pak cyklem projedou všechny objednávky, a poté objednávky s vráceným zbožím konkrétního klienta. Z tabulky order_detail se mezitím ze sloupce „supply“ získá suma skladových zásob za daný typ objednávky. Nyní když jsou známy všechny tři potřebné hodnoty se konečně může provést tato kontrola, která v případě úspěchu vyhodí chybnou výjimku:

```
// Pokud klient vrací víc zboží, než celkem objednal => vrátíme chybu
if ($supply > ($countOfOrderedVariant - $countOfReturnedVariant)) {
    throw new Exception(
      "U produktu: $product_code vracite vice variant
      (typu: $size), nez jste si u nas objednali.");
}
```

Zbytek metody probíhá v duchu funkce addOrder(), kdy se na závěr po spočtení celkové ceny, kterou má klient uhradit vykoná SQL příkaz UPDATE, který cenu uloží do tabulky orders. Po úspěšném skončení této metody se klientovi opět odešle rekapitulační email, tentokrát s informacemi o vráceném zboží.

Obrázek 7 - Email s potvrzením objednávky

Shoe supplies - Potvrzení objednávky pro vrácení zboží: 93

Dobrý den,

zasíláme oznámení o přijetí Vaší objednávky pro vrácení zboží

Souhrn objednávky

Název produktu	Počet	Cena bez DPH	Cena s DPH	Cena celkem s DPH
Boty 4 - 38	15	1 239 Kč	1 499 Kč	22 485 Kč
Boty 4 - 43	15	1 239 Kč	1 499 Kč	22 485 Kč

Cena bez DPH: 37 165 Kč
Celkem k úhradě: 44 970 Kč

Částku nám zašlete na účet **123123123/0100**
Jako variabilní číslo uveďte: **2018000093**.

Produkty nám vrattě (do 14ti dnů prosím) na adresu Zelená 14, Praha 6.

V případě dotazů nás kontaktuje na telefonu +420 123 123 123,
či na naší emailové adrese info@shoesupplies.cz.

Pěkný den přeje **Shoe Supplies!**

Zdroj: Vlastní zpracování

4.3 Vytvoření administrace Eshopu

Administrace eshopu, který využívá REST API metody centrálního skladu bot funguje na podobném principu jako administrace centrálního skladu bot. Obě administrace mají totožný výpis produktů (díky ProductsPresenteru a jeho modelu Products), ve kterém mohou

administrátoři přidávat, editovat a mazat jednotlivé produkty a spravovat jejich varianty. Přihlášení do administrace je také stejné jako u centrálního skladu bot, neboť přihlášení na tomto webu již bylo dostatečně zabezpečené a proto to není nutné na webu s eshopem řešit jiným způsobem. Jediný, čím se od sebe tyto dva weby liší je presenter s názvem – „SuppliesPresenter“ a jeho model – „Requests“. Tyto dva soubory se starají o HTTP komunikaci s REST API centrálního skladu. Aby byla možná tato komunikace, je potřeba mít proto nějaký prostředek. PHP má proto vlastní metody, které ale potřebují řadu úprav, aby byla komunikace zcela bezpečná. Snadnější variantou je stáhnutí již hotové knihovny, která se již tyto všechny detaily má vyřešené. Nejznámější knihovnou pro HTTP komunikaci je Guzzle a jeho pro nette integrovaná podoba „Guzzlette“. Tato knihovna se instaluje pomocí composeru jednoduchým příkazem:

```
composer require matyx/guzzlette eshop-project
```

Tímto příkazem se do nette nainstalují veškeré potřebné třídy a závislosti potřebné ke správnému fungování Guzzle.

4.3.1 Výpis produktů z centrálního skladu bot

Eshop má kromě správy vlastních produktů i možnost vypsání produktů z centrálního skladu, které si pak následně může klient z tohoto výpisu rovnou objednat. Výpis je vzhledově podobný klasickému výpisu produktů, jen se jednotlivé produkty neberou z databáze eshopu, ale za užití již vytvořené REST API metody get-specific-products. Nejprve se pro výpis produktů musí opět vytvořit renderovací metoda – renderListOfProducts(), která má jako parametry jednotlivé filtry, podle kterých si pak může klient vyhledávat konkrétní produkty. Dále je potřeba získat produkty z centrálního skladu. Proto je potřeba v modelu Requests vytvořit metodu „getProductsFromShoeSupplies()“, u které se jako parametry předají filtry z renderovací metody. Dalším krokem je nastudování správného tvaru parametrů požadavku a hlaviček z API dokumentace. Je potřeba vytvořit pole, které bude mít na indexu „query“ pole s jednotlivými parametry požadavku a jejich hodnotami a v indexu „headers“ potřebné hlavičky – v tomto případě se musí odeslat API

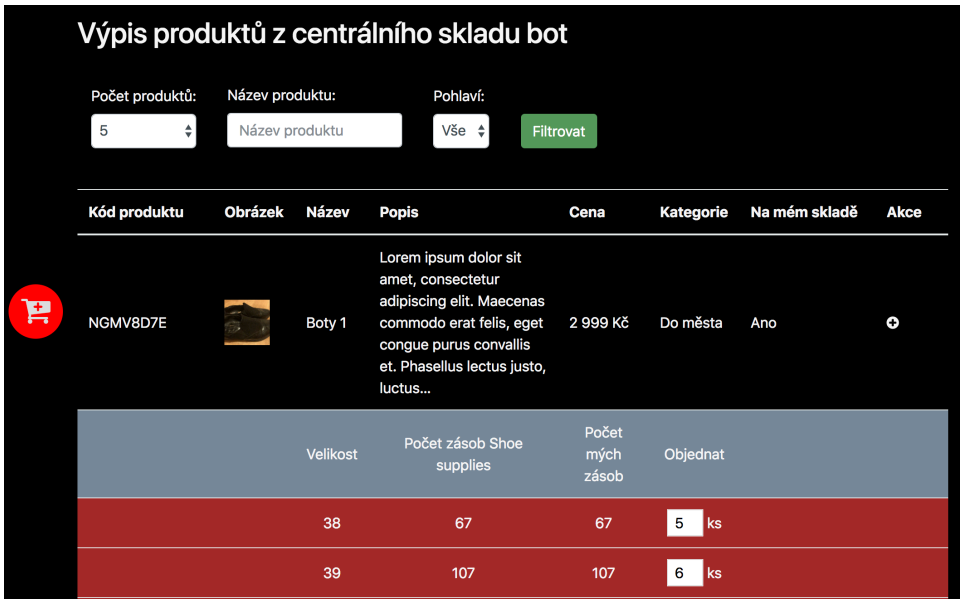
klíč klienta v hlavičce „X-Auth-Token“. Oba názvy těchto indexů jsou definované v dokumentaci Guzzle pro správné odeslání požadavku. Po vytvoření tohoto pole jej stačí společně s URL požadavku odeslat za užití Guzzle funkce pro GET dotazovací metody – „get()“. Tato metoda vrátí odpověď z REST API centrálního skladu bot ve formátu JSON, který je potřeba pro lepší práci překonvertovat do podoby PHP pole. Výsledná metoda pro získání produktů přes vzdálenou komunikaci vypadá takto:

```
// Vrátí produkty dle filtru
public function getProductsFromShoeSupplies(
$products_per_page, $actual_page, $name, $sex) {
    $client = new GuzzleHttpClient();
    $url = "honza-jahoda.cz/subdom/supply/api/get-specific-products";

    // Vložíme parametry v požadovaném formátu
    $params = array(
        'query' => array(
            'products_per_page' => $products_per_page,
            'actual_page' => $actual_page,
            'name' => $name,
            'size' => false,
            'sex' => $sex,
            'available' => false,
            'order' => false
        ) ,
        'headers' => ['Content-Type' => 'application/json',
        'X-Auth-Token' => 'api-key ' . $this->my_api]
    );

    // Odeslání požadavku a získání odpovědi
    $response = $client->get($url, $params);

    // Převod JSON formátu do pole
    $response_in_array = json_decode($response->getBody() , true);

    return $response_in_array;
}
```

Vrácené pole obsahuje potřebná data pro výpis, ale u všech variant chybí ještě informace, zdali je tato varianta na skladě eshopu a popřípadě kolik kusů se na skladě zrovna nachází. Pro tyto informace je potřeba vrácené pole z REST API metody projet cyklem a pomocí kódu produktu najít zboží v databázi eshopu. V případě nalezení tohoto produktu v databázi je pak potřeba k němu následně připsat z tabulky variants skladové kusy. Tím se vrácené pole doplní o hodnoty z databáze eshopu. Samotný výpis je opět tvořen tabulkou, kde jednotlivé řádky představují konkrétní produkt. Oproti běžnému výpisu produktů tu je navíc sloupec značící, zdali je produkt na skladu eshopu či ne. Zároveň se pod řádkem

každého produktu nachází řádky zobrazující informace o jednotlivých, dostupných variantách na centrálním skladě bot. Tyto řádky variant obsahují informace o velikosti boty, počtu skladových zásob na centrálním skladě a počtu zásob na eshopu. V posledním sloupci každé varianty se nachází textový input, do kterého může administrátor napsat počet skladových kusů, které potřebuje na eshop doobjednat. Tyto řádky variant produktu jsou defaultně schované pro lepší přehlednost a dají se zobrazit kliknutím na poslední sloupec řádku produktu, kde se nachází ikonka znaménka plus. Na tuto ikonku je nastavená přes jQuery událost, která po kliknutí zobrazí, či schová řádky s variantami produktu. Produkty se v tomto výpisu vypisují ve stránkování a dají se i klasicky filtrovat, jen se musí využít nově vytvořená metoda getProductsFromShoeSupplies(), namísto původní getSpecificProducts(). Výpis produktů a jeho variant vypadá na eshopu takto:

Obrázek 8 - Výpis produktů pro objednání na eshopu



Zdroj: Vlastní zpracování

4.3.2 Objednání produktů z centrálního skladu bot

Při objednávání jednotlivých variant produktů se administrátorovi zobrazí nalevo od tabulky s výpisem ikonka košíku, která mu po kliknutí zobrazí přehled objednaného zboží a umožní rovnou vybrané zboží objednat za pomoci REST API metody send-order. Ikonka košíku se mu zobrazí, pokud zaklikl alespoň jeden kus zboží. Toto se zjišťuje za pomoci

jQuery funkce on() s parametry „change paste keyup“ (značícím, že k vyvolání události může dojít při vložení dat do inputu, změně inputu či při stisku klávesy) a selektorem (opět nutné zadat cestu se všemi potomky, neboť je výpis dynamicky generován ajaxem). Při splnění podmínek této funkce probíhá kontrola, jestli je objednána alespoň jedna položka. Nejprve se to otestuje na inputu, na kterém proběhla funkce on(), kdy se pomocí funkce val() zjistí, zdali je hodnota v tomto inputu větší jak 0. Pokud ano, zobrazí se ikonka košíku přes funkci fadeIn(). V opačném případě je potřeba funkcí each() projekt veškeré inputy, do kterých se zadává počet kusů, a opět zkontrolovat, jestli je jejich hodnota větší jak 0 a popřípadě zobrazit, či schovat ikonku košíku. Další vhodnou věcí před odesláním požadavku centrálnímu skladu bot, je kontrola zdali administrátor neobjednává více zboží než je na skladě skladu bot. Ten má toto sice ve svém API ošetřené a vrátil by na eshop chybovou hlášku v odpovědi, ale je lepší chybě zabránit již v počátku. Proto se na úplném konci metody on() kontroluje, zdali hodnota v právě vyplněném inputu pro počet kusů není větší než textová hodnota ve sloupci „Počet zásob Shoe supplies“ u stejné varianty. Pokud se snaží administrátor objednat více kusů než může, zobrazí se mu výstražná hláška s chybou (za pomoci javascriptové funkce „alert()“) a input pro počet kusů se mu nastaví na maximální možný počet skladového množství dané varianty.

Zobrazený košík spustí po kliku modální, bootstrapové okno obsahujícím veškeré produkty s variantami, které si administrátor eshopu přeje objednat. Ty se do modálního okna vypíší při události click() na ikonku košíku. Při této události se projedou všechny inputy pro počet kusů, které mají větší hodnotu jak 0 a jejich informace se zaznamenají administrátorovi pro přehled objednávky do modálního okna. Modální okno je zároveň formulářem. Aby bylo vidět při vyhodnocování tohoto formuláře, co administrátor objednal, tak se při projíždění inputů za pomoci jQuery funkce append() přidávají uvnitř formuláře skryté inputy, jenž mají v atributu „name“ pole ve tvaru „pole[produktový kód][id varianty]“ a jeho hodnotami je skladové množství dané varianty.

Komponenta pro odesílání objednávek se nazývá „createComponentOrderProducts()“. Ta má pro formulář nadefinovaný pouze odesílací tlačítko, neboť skryté inputy obsahující jednotlivé objednané varianty se do formuláře přidávají dynamicky přes jQuery. Odesílací tlačítko spouští metodu „orderProductsSucceeded()“, uvnitř které probíhá proces odesílání objednávky přes REST API metodu send-order. Ta obsahuje v modelu Requests funkci s názvem „saveProducts()“,

jenž má jako parametry objednané varianty z formuláře. Tato metoda projíždí jednotlivé objednané varianty a každou z nich uloží do databáze eshopu. Celý tento proces ukládání opět běží za pomocí transakcí a výjimkových bloků try a catch, aby v případě chyby, byly data v databázi odstraněny. Jelikož ale byly ve skrytých inputech uloženy pouze kódy produktu, varianty a počet kusů, tak je potřeba zjistit i ostatních informace o produktu (název, popisek, obrázek...). Proto se v každém cyklu před uložením produktu, zavolá REST API metoda get-product, která vrátí veškeré informace o produktu. K tomu slouží metoda „getProductDetails()“, jenž má jako parametry kód produktu a bool proměnnou značící, jestli se mají ve vrácené JSON odpovědi objevit i varianty produktu, které ale nejsou v tomto případě potřeba (čili FALSE). Tělo metody je podobné metodě getProductsFromShoeSupplies(), kdy jde opět o GET dotazovací metodu, které stačí předat do těla dotazu požadované parametry a hlavičky o odeslat pomocí Guzzle funkce get();

Po získání informací o produktu a jejich následném uložení do databáze, společně s nově objednanými skladovými kusy se provede REST API metoda send-order, čímž se oznámí centrálnímu skladu bot o proběhnuté objednávce. Tento krok probíhá stále uvnitř výjimkového bloku try, díky kterému se v případě neúspěchu zaznamenaná data v tabulkách s produkty a variantami smažou. REST API metoda send-order se volá z funkce sendOrder(). Jejím parametrem je pole skrytých inputů z objednávajícího formuláře obsahujícím jako klíče kód produktu a jako hodnotu pole s klíčem, ve kterém je název varianty a jeho hodnotou je počet kusů. Toto pole se už nemusí nikterak upravovat, poněvadž přesně v tomto tvaru se má odeslat dle dokumentace k této metodě. Tato dotazovací metoda již není typu GET, ale POST. Tato skutečnost ale pro odeslání požadavku v kódu nic nemění, pouze se místo metody get() použije metoda „post()“, jenž má opět jako parametry hlavičky (API klíč klienta) a parametry požadavku (pole s objednanými produkty). Funkce sendOrder() vypadá takto:

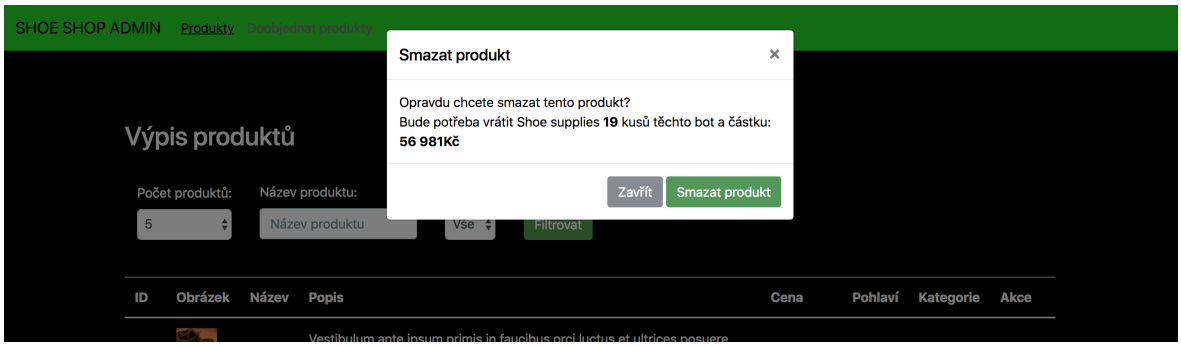
```
// Odešle objendávku
public function sendOrder($products) {
    $client = new GuzzleHttpClient(
      ['headers' =>
            [
                  'Content-Type' => 'application/json',
                  'X-Auth-Token' => 'api-key ' . $this->my_api
            ]
      ]
    );
    $url = "honza-jahoda.cz/subdom/supply/api/send-order";

    // Vložíme parametry v požadovaném formátu
    $params = array(
        'form_params' => [json_encode($products, JSON_NUMERIC_CHECK) ]
    );
    $response = $client->post($url, $params);
    $response_in_array = json_decode($response->getBody() , true);

    if ($response_in_array["status"] === "Success") {
        return true;
    }
    else {
        return $response_in_array["errorMessage"];
    }
}
```

V případě úspěchu této funkce se potvrdí transakce a pomocí flash zprávy se uživateli vypíše oznámení o úspěšném objednání. V opačném případě se transakce nepotvrdí a veškerá data uložená za běhu funkce saveProducts() se smažou. Přes flash zprávu se zároveň klientovi zobrazí chybová zpráva, kvůli které byla objednávka neúspěšná.

4.3.3 Vrácení produktů z centrálního skladu bot

Poslední REST API metodou, kterou může zatím klient centrálního skladu bot využívat je – „return-product“, jenž předá centrálnímu skladu informaci o vrácení zboží. Tato REST API metoda se zavolá při smazání produktu ve výpisu zboží v administraci eshopu. Aby tento proces byl přehledný a uživatel omylem nesmazal produkt při kliknutí na ikonku koše v řádku s produktem, tak se mu nejprve zobrazí modální, bootstrapové okno, ve kterém uvidí přehledně, kolik bude muset centrálnímu skladu zaplatit a kolik objednaných kusů musí vrátit.

Obrázek 9 - Modální okno s formulářem pro smazání produktu z eshopu



Zdroj: Vlastní zpracování

Veškeré tyto informace se administrátorovi vypisují ajaxově za užití metody „handleGetInfoAboutReturnedProduct()“, která z databáze získá počet kusů u daného, objednaného produktu a vynásobí jej cenou za tento produkt. V případě, že má administrátor eshopu opravdu zájem produkt vrátit, tak svůj záměr potvrdí klikem na tlačítko v modálním okně odesílající formulář. Tento formulář je řešen v ProductsPresenteru v komponentě „createComponentDeleteProduct()“. V ní se pro formulář definuje skrytý input, obsahující kód produktu, jenž se má smazat (jeho hodnota se naplní přes jQuery při události kliknutí na ikonku koše u produktu) a tlačítko odesílající formulář.

Metoda zpracovávající tento formulář se nazývá „deleteProductSucceeded()“. V ní se pomocí funkce „returnProduct()“ volá REST API metoda return-product. Pro její úspěšné volání je potřeba ji naplnit správnými parametry obsahujícími informace o kódu produktu a konkrétními variantami, které se mají vrátit (v tomto případě, kdy se maže celý produkt se jedná o všechny varianty produktu). Pro volání metody return-product slouží Guzzle funkce „put()“, neboť se jedná o dotazovací metodu typu PUT. Ta stejně jako ostatní REST API metody vrátí úspěšnou, či neúspěšnou odpověď, na základě které se administrátorovi výpíše příslušené oznámení ve flash zprávě. Výsledná funkce returnProduct vypadá takto:

```php
// Vrátí produkt zpátky do Shoe supplies
public function returnProduct($product) {
    $client = new GuzzleHttpClient(
      ['headers' =>
            [
                  'Content-Type' => 'application/json',
                  'X-Auth-Token' => 'api-key ' . $this->my_api
            ]
      ]
    );
    $url = "honza-jahoda.cz/subdom/supply/api/return-product";
    $order = array();
    foreach($product->related("variants", "product_id") as $variant) {
       $order[$product->product_code][$variant->name] = $variant->supply;
    }

    // Vložíme parametry v požadovaném formátu
    $params = array(
        'form_params' => [json_encode($order, JSON_NUMERIC_CHECK) ]
    );

    // Budeme aktualizovat skladové zásoby na Shoe supplies
    // proto metoda "PUT"
    $response = $client->put($url, $params);
    $response_in_array = json_decode($response->getBody() , true);
    if ($response_in_array["status"] === "Success") {
        return true;
    }
    else {
        return $response_in_array["errorMessage"];
    }
}
```

5 Závěr

V diplomové práci na téma B2B řešení pro správu skladových zásob za užití REST API byly probrány v teoretické části detailní informace o restovém rozhrání, jeho funckionalitách, historii a pravidlech potřebných pro správné vytvoření jeho API. Dále jsou zde zmíněné jednotlivé programovací nástroje, potřebné pro vytvoření aplikací poskytující a využívající REST API. Výstupem práce jsou celkem dvě webové aplikace.

Prvním výstupem je centrální sklad bot, sloužící jako administrace skladu bot, jenž umožňuje administrátorům tohoto webu správu nad veškerými potřebnými funkcemi každého B2B eshopu, jako je správa a výpis klientů, objednávek a produktů. Vývoj této části je důkladně popsán v kapitole 4.1. V kapitole 4.2 je popsán vývoj REST API části této webové aplikace, jenž umožňuje svým řádně registrovaným klientům objednávání produktů za užití REST API metod. O správném užití jednotlivých metod, se klienti dozví z webové stránky s dokumentací k API tohoto B2B eshopu. K dokumentační stránce se klient dostane skrze registrační email. Mimo jiné zde může klient nalézt informace o metodě pro výpis dostupných produktů a o metodě pro vrácení produktů zpátky do centrálního skladu.

Posledním výstupem této práce je administrace menšího eshopu, jenž po přihlášení umožňuje administrátorovi správu produktů na skladě eshopu a hlavně objednávání zboží za užití REST API metod centrálního skladu. Veškeré funkcionality ohledně vývoje backendu tohoto eshopu a HTTP komunikace přes REST API se společně s ukázkami kódu nacházejí v kapitole 4.3.

Výsledné webové aplikace s implementovaným REST API rozhraním se podařilo úspěšně zhotovit. B2B eshop nabízí svým klientům API, přes které mohou klienti při správně vytvořeném HTTP požadavku objednávat produkty do svých eshopů, jak lze vidět v kapitole 4.3.2. Prototyp B2B eshopu je tedy funkční a schopný fungovat ve skutečném provozu, ale obsahuje části, které by se na něm mohly ještě vylepšit. Například ověřování klientů probíhá pomocí vložení unikátního API klíče klienta do speciální HTTP hlavičky, přitom se v dnešní době tyto situace řeší pomocí protokolu OAuth 2.0, který autentizaci zvládá ještě bezpečněji. Samotné REST API nyní nabízí 4 základní metody pro objednávání a výpis produktů. Jejich počet by se mohl v budoucnu rozšířit například o metodu vracející veškeré kategorie produktů, či metodu obsahující informace o klientových objednávkách.

6 Seznam použitých zdrojů

1. GOURLEY, David. a Brian. TOTTY. *HTTP: the definitive guide*. Sebastopol, CA: O'Reilly, 2002. ISBN 978-1565925090.
2. ROUSE, Margaret. *World Wide Web (WWW)* [online]. 2017 [cit. 2018-03-23]. Dostupné z: http://whatis.techtarget.com/definition/World-Wide-Web
3. MEDLEY, Joe, Stephanie HOBSON a Chris MILLS. *HTTP headers* [online]. 1.1. 2018 [cit. 2018-03-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
4. MASSÉ, Mark. *REST API design rulebook*. Sebastopol, CA: O'Reilly, 2012. ISBN 9781449310509.
5. RICHARDSON, Leonard a Michael AMUNDSEN. *RESTful Web APIs*. Beijing: O'Reilly, 2013. ISBN 978-1449358068.
6. FREDRICH, Todd. *Using HTTP Methods for RESTful Services* [online]. 2018 [cit. 2018-03-23]. Dostupné z: http://www.restapitutorial.com/lessons/httpmethods.html
7. FIELDING, Roy a Julian RESCHKE. *HTTP/1.1 Semantics and Content* [online]. 2014 [cit. 2018-03-23]. Dostupné z: https://tools.ietf.org/html/rfc7231
8. FIELDING, Roy. *Hypertext Transfer Protocol -- HTTP/1.1* [online]. 2004 [cit. 2018-03-23]. Dostupné z: https://www.w3.org/Protocols/rfc2616/rfc2616.html
9. ROUSE, Margaret. *URI (Uniform Resource Identifier)* [online]. 2016 [cit. 2018-03-23]. Dostupné z: http://searchmicroservices.techtarget.com/definition/URI-Uniform-Resource-Identifier
10. RODRIGUEZ, Alex. *RESTful Web services: The basics* [online]. 2015 [cit. 2018-03-23]. Dostupné z: https://www.ibm.com/developerworks/webservices/library/ws-restful/
11. JAMES, Geordy. *Creating a simple REST API in PHP* [online]. 2017 [cit. 2018-03-23]. Dostupné z: https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php
12. LEONARD RICHARDSON ET SAM RUBY a Hervé Soulard TRADUCTION DE SANDRINE BURRIEL. *Services Web RESTful*. Paris: O'Reilly, 2007. ISBN 9782841774487.

13. WELLING, Luke a Laura THOMSON. *PHP and MySQL web development*. Fifth edition. Hoboken, NJ: Addison-Wesley, 2017. Developer's library. ISBN 978-0321833891.
14. MURACH, Joel a Ray. HARRIS. *Murach's PHP and MySQL: training & reference*. 2nd edition. Fresno, CA: Mike Murach and Associates, 2014. ISBN 978-1890774790.
15. SOLAR VILARINO, Pablo a Carlos PEREZ SANCHEZ. *PHP Microservices*. Packt, 2017. ISBN 9781787125377.
16. DE LA NUEZ, Tony. *PHP - Consuming REST APIs with cURL or file_get_contents* [online]. 2017 [cit. 2018-03-24]. Dostupné z: https://tonydelanuez.com/PHP-Consuming-REST-APIs-with-CURL-or-file_get_contents/
17. DOWLING, Michael. *Guzzle Documentation* [online]. 2015 [cit. 2018-03-24]. Dostupné z: http://docs.guzzlephp.org/en/stable/
18. C. ARNTZEN, Thies, Rasmus LERDORF, Stig BAKKEN a Sascha SCHUMANN. *PHP documentation* [online]. 2001 [cit. 2018-03-24]. Dostupné z: http://php.net/docs.php
19. MILLS, Chris a Vidhi BAGADIA. *Server-side web frameworks* [online]. 2017 [cit. 2018-03-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks
20. ŠULC, Milan a David GRUDL. *Nette dokumentace* [online]. 2008 [cit. 2018-03-24]. Dostupné z: https://doc.nette.org
21. WEINBERG, Paul N., James R. GROFF a Andrew J. OPPEL. *SQL, the complete reference*. 3rd ed. New York: McGraw-Hill, c2010. ISBN 978-0071592550.

7 Přílohy

7.1 Obsah CD

- /centralni-sklad – Kompletní zdrojový kód centrálního skladu bot
- /eshop – Kompletní zdrojový kód pro klienta centrálního skladu bot