FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Backend a CI administrace e-shopu
Student:	Bc. Tomáš Hojek
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je ve spolupráci se studentem Aloisem Koubou realizovat funkční backend webové aplikace administrace e-shopu. Frontend vytváří Martin Dvořák a Jan Babák. Důraz je kladen na týmovou spolupráci a vhodnou metodiku vývoje.

Postupujte v těchto krocích:
1. Analyzujte současné řešení e-shopové administrace a předešlé práce věnující se vývoji daného projektu.
2. Společně s ostatními členy navrhněte vhodnou metodiku vývoje a spolupráce. Při návrhu přihlédněte k možný problémům týmové spolupráce.
3. Vhodným způsobem proveďte rozdělení potřebné práce a realizujte dílčí návrhy částí budoucího backendu.
4. Konzultujte návrh s frontendovou částí týmu.
5. Připravte prostředí pro vývoj backedové části aplikace a vhodně nastavte CI (Continuous integration).
6. Implementujte výsledný návrh.
7. Při realizaci nezapomeňte na důkladné testování, volte také vhodné testování při návrzích rozhraní pro frontendovou část projektu.
8. Zhodnoťte výsledné řešení, navrhněte úpravy do budoucna.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 21. ledna 2022 v Praze.

Diplomová práce

BACKEND A CI ADMINISTRACE E-SHOPU

Bc. Tomáš Hojek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
22. června 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu, panu Ing. Jiřímu Hunkovi, za vedení této diplomové práce a za ochotu a pomoc při jejím psaní. Zároveň bych chtěl poděkovat Ing. Janu Matouškovi za konzultace a poskytnuté informace o současném systému pro administraci, protože bez jeho podpory a znalostí současného systému by tato práce možná ani nevznikla. Dále bych chtěl poděkovat Aloisovi Koubovi, který byl mým týmovým kolegou při návrhu a vývoji systému, s nímž jsem mohl diskutovat mnoho problémů, které se v průběhu implementace objevily. Také bych chtěl poděkovat dvojici Martin Dvořák a Jan Babák, kteří pracovali na klientské aplikaci. V neposlední řadě bych rád poděkoval své rodině za podporu během celého mého studia.

Prohlášení

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

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

V Praze dne 22. června 2022 .....

Abstrakt

Tato diplomová práce má za cíl vytvořit novou serverovou aplikaci (backend) pro administraci e-shopů. Na začátku je popsán současný systém od společnosti Jagu, s. r. o. Následně je popsán návrh, který byl vytvořen na základě provedené analýzy. Největším omezením pro nově navrhovaný systém je již existující databáze, kterou je nutno respektovat a neměnit příliš její strukturu, neboť současná administrace nesmí být změnami negativně ovlivněna. Návrh byl implementován v jazyce PHP a frameworku Symfony. Implementované části systému byly řádně otestovány. V rámci této práce nebyl navržen a vytvořen kompletní systém, který by nahradil současné řešení pro administraci. Hlavním cílem bylo položit základy a implementovat některé z podstatných funkcionalit budoucího systému pro administraci e-shopů.

Klíčová slova e-shop, administrační systém, backend, PHP, Symfony

Abstract

This diploma thesis aims to create a new server side of an application (backend) for e-shop administration. At the beginning there is described the current system developed by company Jagu, s. r. o. Subsequently, the design that was created on the basis of the analysis is described. The most important constraint for the newly designed system is the existing database, that must be respected and not changed too much in its structure. Any changes must not negatively affect the current system. The design was implemented in PHP language and Symfony framework. The implemented parts of the system were properly tested. In this thesis, the complete system has not been designed and developed to replace the current administration system. The main goal was to lay the foundation for the new system and develop some essential functionalities of the future e-shop administration system.

Keywords e-shop, administration system, backend, PHP, Symfony

Seznam zkratek

API	Application Programming Interface
CI	Continuous integration
CORS	Cross-Origin Resource Sharing
DIČ	Daňové identifikační číslo
DPH	Daň z přidané hodnoty
GDPR	General Data Protection Regulation
GIF	Graphics Interchange Format
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IČO	Identifikační číslo osoby
JPEG	Joint Photographic Experts Group
JSON	Javascript Object Notation
JWK	JSON Web Key
JWKs	JSON Web Key Set
JWT	JSON Web Token
OAS	OpenAPI Specification
ORM	Object–relational mapping
OS	Operating system
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
PNG	Portable Network Graphics
REST	Representational state transfer
SCM	Source Code Management
SDLC	Software Development Life Cycle
SEO	Search engine optimization
SQL	Structured Query Language
SRS	Software Requirements Specification
TCP	Transmission Control Protocol
TO	Transport object
UC	Use case
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VCS	Version Control Systems
YAML	YAML Ain't Markup Language
2FA	Two-factor Authentication

Úvod

Obchod je součástí lidského života od nepaměti. Nejdříve se uplatňoval takzvaný směnný obchod, tedy prostá výměna jednoho předmětu za jiný. Později se začaly pro směnu používat vzácné kameny, mušle, až se nakonec začaly používat peníze. Zároveň již od rané fáze obchodování začaly vznikat obchodní cesty, po kterých cestovali obchodníci a v městech a vesnicích po cestě nabízeli své zboží. Vznikaly také kamenné obchody, kam přicházeli především stálí zákazníci a nakupovali různé předměty nebo služby.

Od počátku obchodu již uplynulo mnoho let a mnohé se za tu dobu změnilo. Jedním z nesporných milníků je příchod počítačů a především internetu. S touto érou se začínají kamenné obchody vylidňovat a narůstá popularita něčeho, co se nazývá e-shop. E-shop je označení pro internetový obchod a zákazníci jeho prostřednictvím mohou vybírat a nakupovat zboží odkudkoliv na světě. Obchodníkům však tímto přibývá jedna velká starost navíc. Musí se kromě svého kamenného obchodu starat i o internetový obchod, popř. mají pouze internetový obchod. Řada platforem pro tvorbu e-shopů již dnes umožňuje vytvářet graficky zajímavé a poutavé e-shopy. Stejně jako u kamenného obchodu je totiž podstatné zákazníka okouzlit krásným, přehledným a intuitivním vzhledem.

Obchodník se však musí o svůj internetový obchod starat a spravovat ho. K tomuto účelu se používá takzvané administrativní rozhraní. A jelikož tato část e-shopu je jen pro obchodníka, nikoliv zákazníka, tak je kvalita administrativního rozhraní o poznání horší. Velice často se ukazuje, že je backend e-shopu velice komplikovaný a neintuitivní. A především chybí možnost komponenty jednoduše přidávat, odebírat a upravovat. Toto je veliký problém, neboť každý obchod je jiný a má jiné požadavky. Kvůli tomu je velice často nutné dělat úpravy před i po spuštění e-shopu.

Jednou z aplikací pro administraci je administrační rozhraní od společnosti Jagu, s. r. o. Tato aplikace je postavena na platformě OpenCart. Naneštěstí během rozvoje této aplikace se objevily problémy spojené s rozšiřitelností a dlouhodobou udržitelností. Zmíněná společnost měla dříve obdobný problém s jejich skladovým systémem, který úspěšně vyřešili vytvořením nového skladového systému. Proto následně přišla myšlenka vytvořit nové rozhraní také pro administraci e-shopu.

Cílem této práce bude analyzovat současné řešení e-shopové administrace. Na základě provedené analýzy bude vypracován návrh, jenž bude konzultován s kolegy, kteří se na vývoji nového systému budou rovněž podílet. Jedná se o Martina Dvořáka, Jana Babáka a Aloise Koubu. Vypracovaný návrh bude implementován a následně vhodně otestován.

Kapitola 1
Analýza

V rámci této kapitoly bude popsán současný systém administrace e-shopu. Jelikož je však systém velmi obsáhlý, tak na tomto tématu bude spolupracovat tým několika lidí. Autor této práce, Bc. Tomáš Hojek, a Alois Kouba budou spolupracovat na tvorbě serverové části nového systému pro administraci. Martin Dvořák a Jan Babák budou spolupracovat na tvorbě klientské části aplikace. Nejprve budou představeny možné metodiky týmového vývoje aplikace.

1.1 Metodiky vývoje

V této podkapitole budou popsány hlavní metodiky vývoje, jejich výhody a nevýhody. Někdy se používá zkratka SDLC, která pochází z anglického spojení Software Development Life Cycle, tj. životní cyklus vývoje softwaru. Toto označení je výstižné, neboť proces vývoje softwaru se vždy skládá z několika etap, které jsou definovány v rámci dané metodiky.

Jedna z metodik bude později vybrána pro vývoj aplikace, na kterou se soustředí tato práce. Výběr metodiky je velmi důležitý, neboť metodika má následně značné dopady na vývoj aplikace, provedení změnových požadavků a další.

1.1.1 Waterfall model

Waterfall model, nebo též vodopádový model, je přístup projektového řízení. Tento model má několik fází, které na sebe navazují a stejně jako u vodopádu je možné postupovat pouze dopředu (či dolů, pro lepší paralelu s vodopádem). Na začátku projektu se vytvoří obsáhlý a kompletní plán, který se následně realizuje a výsledný produkt se předá zákazníkovi. Celý proces realizace je lineární a nepředpokládá se změna či úprava zadání. [1]

Jak již bylo dříve zmíněno, waterfall model je rozdělen do několika striktně oddělených etap. Jednotlivé etapy dle [1] jsou níže vyjmenovány a popsány.

Požadavky a analýza je počáteční fáze sloužící pro sběr business požadavků a jejich následnou analýzu.

Návrh zahrnuje volbu technologií, plánování softwarové architektury a vytváření diagramů.

Implementace zahrnuje psaní kódu a řešení různých problémů.

Testování je etapa, ve které se ověřuje funkčnost, tedy zda aplikace dělá to, pro co byla navržena.

Provoz aplikace zahrnuje nasazení aplikace do produkčního prostředí a následnou údržbu.

Tento model řízení má dle [2] a [1] několik výhod. Největší výhodou je, že jsou jasně odděleny jednotlivé fáze tohoto modelu, díky čemuž je snazší pochopit tento model i výsledný sestavený plán. Mezi další výhody patří:

- konzistentní a úplná dokumentace hned na začátku,
- snazší sledování pokroku v projektu (postup je lineární a každá fáze projektu je naplánovaná),
- efektivní využití času členů týmu (každý dopředu ví, co přesně bude dělat, a proto jsou odhady přesnější),
- klient ví, co přesně očekávat (zná časovou osu, co dostane a kolik to bude stát),
- vše je naplánováno na začátku a v průběhu projektu není potřeba konzultací s klientem (nebo jen výjimečně),
- lepší návrh, protože rozsah a funkce jsou známé a nemění se v průběhu projektu.

Waterfall model má dle [2] a [1] i celkem značné nevýhody. Především musí být kladen velký důraz na první fázi projektu, protože potencionální chyba v analýze se přenese i do dalších fází. Dále je zásadním problémem striktní lineárnost projektu, z čehož pramení několik problémů:

- zdržení jedné fáze znamená zdržení celého projektu (musí se vždy čekat na dokončení předchozí fáze),
- obtížný návrat zpět (pokud je některá fáze dokončena, tak je velmi náročné a drahé vrátit se zpět),
- testování a ověření kvality je až po vývoji (nemožné dělat větší opravy a úpravy),
- klient vždy nedostane to, co potřebuje (klient obvykle neví přesně, co potřebuje a zjistí to až ve chvíli, kdy si aplikaci sám vyzkouší, což je v případě tohoto modelu teprve po vyvinutí celé aplikace),
- neočekávané problémy (obvykle se nepodaří odhalit všechny problémy při analýze či návrhu a později může být obtížné je vyřešit).

1.1.2 Iterativní model

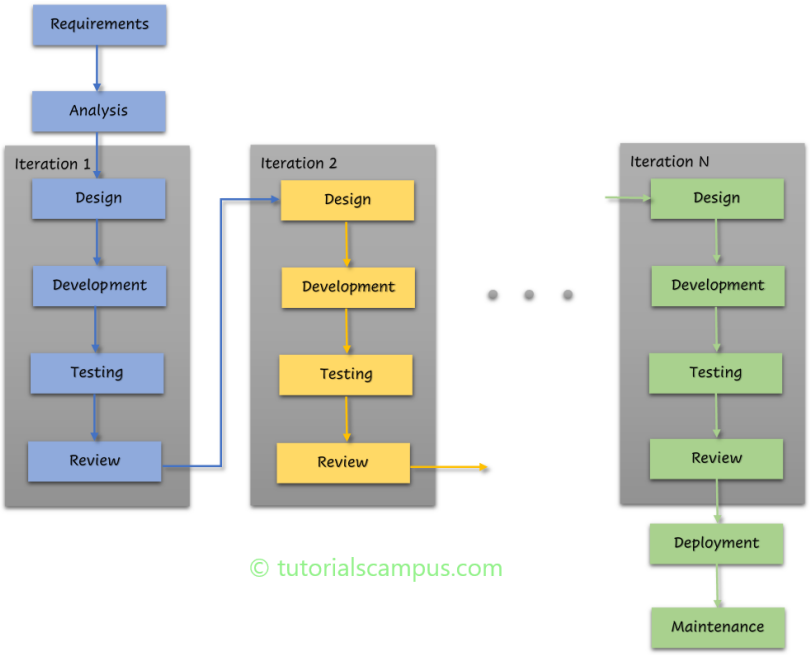
Iterativní model počítá nejprve s implementací malé sady požadavků. Následně se iterativně, s každou verzí, přidávají další funkcionality. Jakmile se dosáhne stanoveného cíle a systém je kompletní, aplikace je připravena na nasazení. [3]

Iterativní model je vlastně posloupnost několika vodopádů. Nejedná se tedy o jeden velký vodopád, ale několik menších. Grafické znázornění iterativního modelu je na obrázku 1.1.

Iterativní vývoj obsahuje následující kroky [5]:

- požadavky a analýza,
- návrh,
- vývoj,
- testování,
- vyhodnocení,
- provoz aplikace.

Obrázek 1.1 Grafické znázornění iterativního modelu vývoje softwaru. Obrázek převzat z [4].



Jelikož první čtyři fáze byly již dříve popsány v předcházející podkapitole u waterfall modelu, tak jejich popis je zde vynechán. Pouze je nutné zdůraznit, že sběr požadavků a analýza se provádí na začátku projektu. Později může v rámci fáze návrhu probíhat další (dodatečná) analýza týkající se aktuálně zpracovávané funkcionality.

Vyhodnocení je etapa, která zakončuje každou iteraci a vyhodnocuje se v ní vyvinutá funkcionalita. Tato fáze umožňuje celému týmu a zadavateli (klientovi) posoudit stav projektu a v případě potřeby upravit analýzu dle zjištěných kritérií. Dále se v této fázi plánuje, co se bude dělat v další iteraci. [5]

Etapa provozu aplikace, zahrnující nasazení a údržbu, byla již popsána v podkapitole 1.1.1.

Nyní je čas vyjmenovat několik výhod, které jsou charakteristické pro iterativní vývoj. Nejzásadnější změnou oproti předchozímu waterfall modelu je, že je možné dělat změny v návrhu, protože vývoj se skládá z několika menších vodopádů. Další výhody dle [5] jsou:

- první funkce nové aplikace jsou vyvinuty rychle v rané fázi projektu,
- testování je při menších iteracích jednodušší (neprovádí se až v závěru projektu, ale průběžně),
- snadná koordinace práce a snáze se plánuje paralelní vývoj,
- průběžně je možné získávat zpětnou vazbu od klienta a provádět změnové požadavky,
- první etapa, zahrnující požadavky a analýzu, nemusí být zcela kompletní při vstupu do další fáze (stále však existuje jasně definovaný plán),
- identifikace a řešení rizik je v rámci iterace jednodušší a může být jednodušší řízení rizik (rizikovější části aplikace se například vyvinou dříve),
- celkem dobrá predikovatelnost (čas, rozsah, cena).

Iterativní vývoj má dle [5] i několik nevýhod. Mezi hlavní nevýhody iterativního vývoje se řadí:

- stále nejsou možné časté změnové požadavky (ačkoliv jsou možné oproti waterfall modelu, tak nejsou dodány rychle),
- celý proces je náročnější na řízení a vyžaduje více zdrojů (například testery i vývojáře je nutné zajistit na celou dobu vývoje aplikace),
- nevhodnost pro malé projekty,
- limitovaný čas na dokumentaci a návrh,
- běh projektu je vysoce závislý na analýze rizik, která musí být provedena v rané fázi projektu (nutno chápat, co se chce, již na začátku),
- čas dokončení projektu nemusí být znám (analýza na začátku není kompletní a záleží také na změnových požadavcích v průběhu projektu).

1.1.3 Agilní model

Agilní model vývoje softwaru je kombinací iterativního modelu se zaměřením na adaptaci a uspokojení zákazníka. Tento model prosazuje rychlé doručení funkčního softwaru zákazníkovi, a to pomocí něčeho, co je nazýváno release (česky *uvolnění*). Každý release se dále ještě dělí na menší části, které nazýváme iterace. V závislosti na projektu a použití může release a iterace splývat a obecně je tedy možno říci, že vývoj probíhá ve verzích, ale ne každá verze je produkční. [6][7]

Typicky každá iterace trvá jeden až čtyři týdny. Znázornění agilního vývoje je na obrázku 1.2. Iterace se dle [7] skládá z následujících etap:

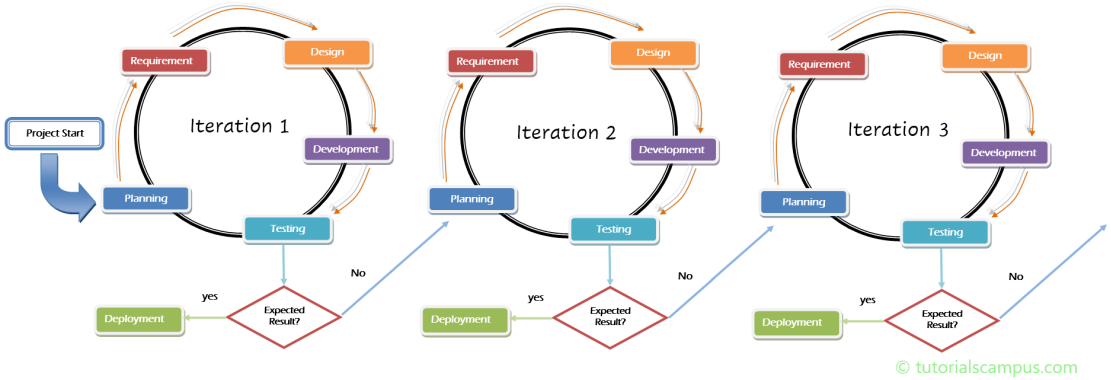
- plánování a shromažďování požadavků,
- analýza požadavků,
- návrh,
- vývoj,
- testování a vyhodnocení.

Na základě testování a vyhodnocení dochází k uvolnění verze (release) nebo k přesunu do další iterace (uvolnění verze je odloženo). Popis jednotlivých etap je zde vynechán, neboť se shoduje s dřívějšími popisy ekvivalentních etap. Hlavní změnou u tohoto modelu je to, že jsou zákazníkovi pravidelně a často dodávány jednotlivé funkční verze. Zákazník má tedy přehled nad probíhajícím vývojem a díky častým iteracím je možné rychle reagovat na změnové požadavky, které mohou mít velký dopad na podnikání zákazníka. [7]

Dle [7] má agilní model mnoho výhod. Mezi ty hlavní patří především:

- eliminace nedorozumění a přepisování vytvořených funkcionalit díky časté komunikaci se zákazníkem,
- rychlý vývoj a zpětná vazba od zákazníka,
- šetří náklady a je efektivní,
- snižuje potřebu práce na dokumentaci,
- zvyšuje zaměření na výslednou kvalitu produktu,

Obrázek 1.2 Grafické znázornění agilního modelu vývoje softwaru. Obrázek převzat z [8].



- není nutné mít všechny požadavky na začátku projektu.

Hlavními nevýhodami dle [7] jsou:

- není vhodné pro projekty s mnoha závislostmi,
- méně dokumentace,
- zákazník musí vědět, co chce a kam projekt směřovat (nutný silný business vlastník),
- klade vyšší nároky na tým a spolupráci v rámci týmu,
- nutná velmi častá (každý den) komunikace se zákazníkem,
- pouze zkušení vývojáři nebo experti mohou dělat kritická rozhodnutí,
- je dražší než předchozí modely (ale na druhou stranu zákazník dostane přesně to, co chce).

Agilní model si získal velkou popularitu díky své přizpůsobivosti a flexibilitě. Populární modely vycházející z agilního modelu jsou například:

- Scrum,
- Extrémní programování,
- Feature Driven Development (česky *vývoj řízený vlastnostmi systému*). [7]

1.2 Sdílení a správa souborů

Jelikož vývoj bude probíhat v týmu, tak je nutné se podívat, jaké jsou možné metody pro sdílení zdrojového kódu. Tomuto tématu se věnuje takzvaný Source Code Management (SCM). SCM každou změnu (nebo spíše seskupení změn) uloží jako verzi. Jelikož tyto systémy pracují s verzemi, tak mohou být také označovány jako VCS (Version Control Systems). Daný proces se označuje jako verzování. [9]

Existují dva systémy pro verzování - centralizované a distribuované řešení. Oba tyto přístupy budou později popsány v následujících podkapitolách.

Verzovací systémy mají typicky nějaký mechanismus, jak vyřešit spolupráci několika lidí nad stejným souborem, tedy když dva lidé chtějí upravit stejný soubor. Existují dva typické přístupy: zamykání (*lock*) a slučování změn (*merge*). [10]

VCS, které používají strategii slučování, fungují na principu kopie – úpravy – sloučení. Tento postup lze dle [10] popsat následujícím způsobem.

1. Uživatel provádí změny ve své pracovní kopii souboru.
2. Uživatel dokončí úpravy a chce, aby tyto změny byly uloženy ve vzdáleném repositáři.
3. Systém nechá uživatele udělat sloučení jeho verze s verzí, která je aktuálně na serveru (ta na serveru se může lišit od verze, ze které uživatel vycházel, když začínal dělat úpravy).
4. Předchozí krok je většinou proveden automaticky verzovacím systémem. Systém porovnává změny, které provedl uživatel, se změnami, které provedli ostatní uživatelé od posledního slučování.
5. Pokud se v předchozím kroku změny nepřekrývají, provede se automatické sloučení. Pokud se naopak změny překrývají, musí uživatel provést ruční úpravy a rozhodnout, jak má soubor po sloučení vypadat.

Naopak VCS, které používají strategii zamykání, fungují na principu zamknutí – úpravy – odemknutí. To dle [10] používají spíše starší verzovací systémy a postup je přibližně popsán v následujícím seznamu.

1. Všechny soubory jsou automaticky pouze ke čtení.
2. Pokud uživatel chce zapisovat, musí o toto oprávnění požádat server.
3. Verzovací systém oprávnění na zápis vydá pouze jednomu uživateli v jednu chvíli. Tedy jeden uživatel zapisuje a ostatní uživatelé mohou pouze číst.
4. Když uživatel dokončí úpravy, odešle soubor a tím ukončí i zápis.
5. Verzovací systém odemkne soubor.

1.2.1 Centralizovaný verzovací systém

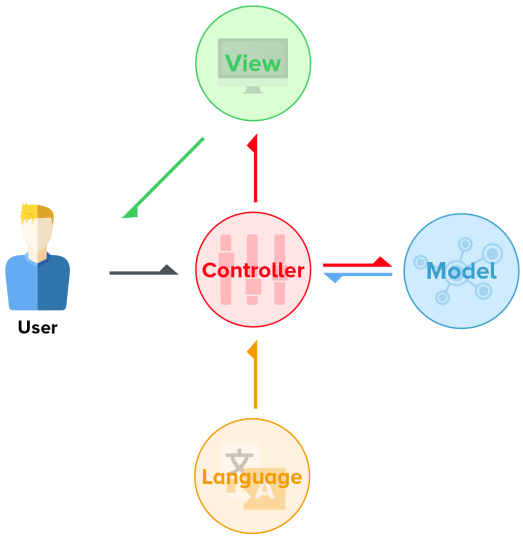
Tento systém předpokládá množinu počítačů a jeden server, který funguje jako centrální prvek. Pokud je na některém z počítačů provedena změna, tato změna je uložena do úložiště (vzdáleného repositáře) na serveru. Pracovní kopie souborů jsou tedy dostupné na počítači, ale každá změna se zapisuje přímo do vzdáleného repositáře na serveru. Jinými slovy, centrální server spravuje verze. Pokud dojde k chybě na serveru, nebo nefunguje internetové připojení, vývojář nemá přístup k verzím. Bez přístupu k serveru tedy vývojář nemůže pracovat, což je velkým záporem tohoto systému. [9]

1.2.2 Distribuovaný verzovací systém

Distribuovaný verzovací systém opět předpokládá množinu počítačů a server. Server spravuje vzdálený repositář a data uložená v něm. Zároveň však každý počítač má i vlastní lokální kopii repositáře. Díky tomu při selhání serveru nebo výpadku připojení je nadále možné měnit kód (mění se lokální kopie). Každý repositář zaznamenává lokální historii změn (verze). Na příkaz je pak možné lokální repositář:

- aktualizovat (neboli stáhnout aktuální verzi repositáře),
- nahrát lokální změny na server,
- provést takzvaný merge (česky *spojení*), které se provádí, pokud došlo ke změně na stejném místě ve vzdáleném i lokálním repositáři. [9]

Obrázek 1.3 Vizuální znázornění MVC-L architektury. Obrázek převzat z oficiální dokumentace [14].



1.3 Současný stav

Současná administrace od Jagu, s. r. o., je implementována pomocí open-source platformy OpenCart.

"OpenCart je volně dostupná open-source platforma pro obchodování po internetu (e-shopy). OpenCart poskytuje profesionální a spolehlivý základ pro vybudování úspěšného e-shopu. Tento základ je použitelný pro širokou skupinu uživatelů, od zkušených webových vývojářů hledajících uživatelsky přívětivé rozhraní až po vlastníky obchodů, kteří právě své podnikání poprvé rozšiřují o online prostředí. OpenCart poskytuje mnoho možností úprav e-shopu dle přání zákazníka. S nástroji OpenCart můžete pomoci svému e-shopu dosáhnout jeho maximálního potencionálu."[1] [11, překlad autor]

Dle [12] je aktuální verze OpenCart, v.3.0.2.0, založená na PHP (minimální verze 5.4) a doporučuje se použití této platformy ve spojení s databází MySQL a webovým serverem Apache.

Platforma OpenCart poskytuje open-source frontend i backend e-shopu. Systém je možné rozšiřovat pomocí již hotových modulů, které lze získat z OpenCart extension storu. Díky open-source licenci je možné kód i libovolně upravovat a rozšiřovat tím funkčnost aplikace.

Dle [13] moduly OpenCart platformy používají architekturu MVC-L, tedy model, view, controller, language. Toto rozdělení je také vizuálně znázorněno na obrázku 1.3. OpenCart tedy definuje rozdělení aplikace do zmíněných čtyř vrstev, díky čemuž je aplikace trochu členěná, ale stále se jedná o více či méně monolitickou aplikaci. Například přechod k microservice architektuře není možný. Největším nedostatkem je zřejmě přímé propojení mezi backend a frontend částí, jinak řečeno backend nevystavuje žádné API, což přináší značná omezení a v dnešní době je to značná slabina.

OpenCart platforma má zásadní vliv na podobu databáze. Současná administrace (i z historických důvodů, viz dále) trpí několika nedostatky:

[1] OpenCart is free open source e-commerce platform for online merchants. OpenCart provides a professional and reliable foundation from which to build a successful online store. This foundation appeals to a wide variety of users; ranging from seasoned web developers looking for a user-friendly interface to use, to shop owners just launching their business online for the first time. OpenCart has an extensive amount of features that gives you a strong hold over the customization of your store. With OpenCart's tools, you can help your online shop live up to its fullest potential.

- Databáze obsahuje prázdné tabulky a prázdné sloupce. Tato skutečnost znepřehledňuje databázi.
- V databázi nejsou cizí klíče a výjimečně chybí i primární klíče. Neexistence cizích klíčů vede k situaci, kdy je nutné závislosti kontrolovat na úrovni aplikace, čímž aplikace nahrazuje jednu z primárních funkcí databází.

1.3.1 Historie

Firma Jagu, s. r. o., působí na trhu již řadu let, byla založena 26. 7. 2008 [15]. Dle [16] během svého působení pracovala na řadě projektů s různorodým zaměřením a mezi její produkty patří například:

- Stylka – designový a stylový e-shop s domácí elektronikou, který se zaměřuje na zdraví, péči o tělo a domácnost,
- Můj Remington – e-shop s výrobky společnosti Remington.

Všechny e-shopy jsou založeny na již zmíněné platformě OpenCart (verze 1.1.).

Pro účely této práce není konkrétní historie příliš podstatná a je zřejmé, že firma má za sebou určitou technologickou historii a z ní je nutné vycházet, neboť tyto okolnosti budou ovlivňovat i tuto práci.

V několika následujících podkapitolách bude provedena analýza současného systému, která bude zaměřena na popis existujících funkcí v systému a popis databáze. Popsány budou ty části aplikace, které budou následně zpracovány v nové administraci, tedy:

- výrobci,
- kategorie,
- produkty,
- objednávky.

1.3.2 Účastníci

Současná administrace umožňuje definovat uživatelské skupiny a přiřazovat do nich uživatele. Každá uživatelská skupina má definována přístupová práva a práva na editaci. Tyto parametry lze dynamicky měnit a skupiny uživatelů tedy nejsou fixně definovány.

Pro účely tohoto textu budou výše zmíněné skutečnosti zanedbány a bude předpokládána pouze jediná uživatelská skupina - administrátor (zkráceně též admin).

1.3.3 Výrobci

Tato podkapitola popisuje stávající funkce související s výrobci, kteří dodávají produkty nabízené v e-shopu. Jedná se o jednodušší téma, kde není mnoho závislostí na další části sytému.

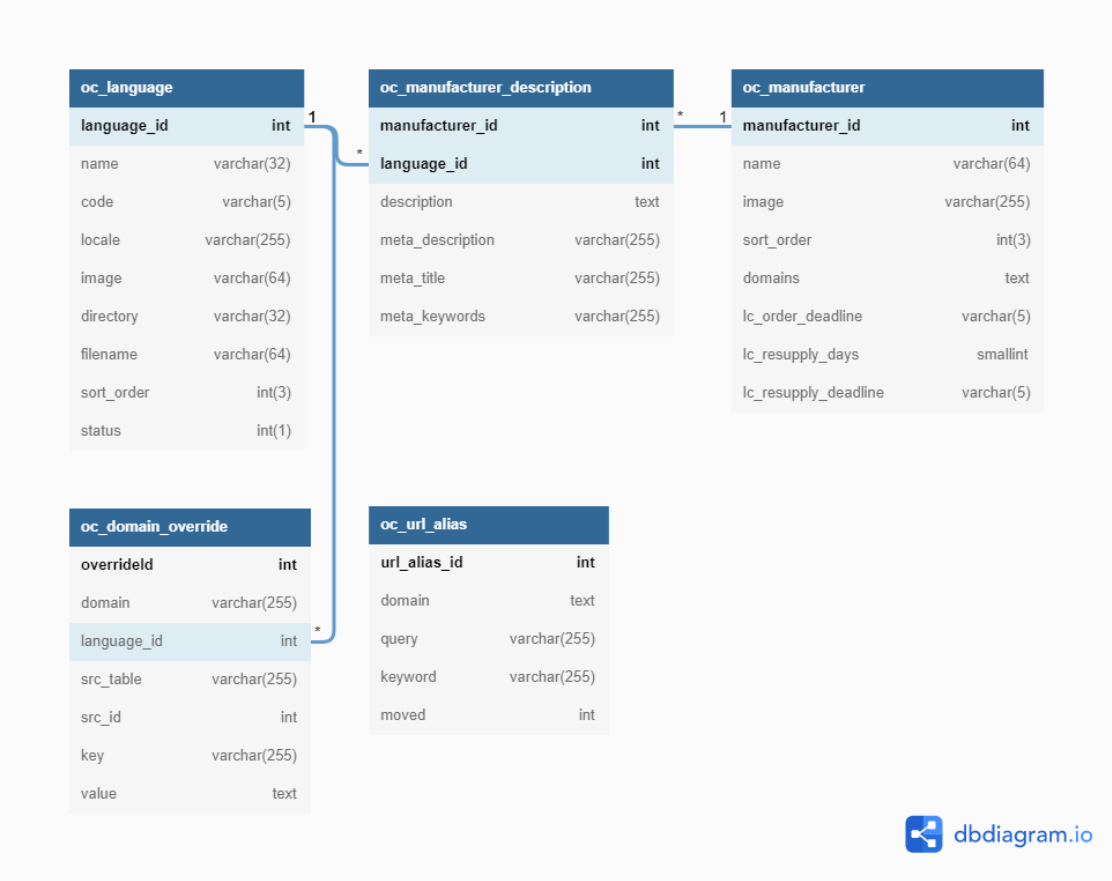
Aktuální podoba databáze týkající se výrobců je znázorněna na obrázku 1.4. Význam a popis jednotlivých tabulek je uveden níže.

oc_manufacturer obsahuje základní informace o výrobci.

oc_manufacturer_description obsahuje další informace o výrobci (především delší texty, popisy).

oc_domain_override obsahuje upřesňující texty pro konkrétní domény. Když neexistuje text v této tabulce, použije se výchozí text definovaný ve výše uvedených tabulkách.

Obrázek 1.4 Schéma databáze zachycující tabulky týkající se výrobce.



oc_url_alias je tabulka, která obsahuje takzvaný URL slugs, které jsou vždy pro danou doménu unikátní (bližší vysvětlení tohoto termínu níže). Každý výrobce by měl mít definovaný URL slug.

oc_language obsahuje seznam jazyků, jejich zkratky, kódy, obrázky (vlajky) a další. Tabulka `oc_manufacturer_description` může pro každého výrobce obsahovat více záznamů v závislosti na jazyku (používalo se historicky, nyní se nepoužívá).

URL slug je seskupení znaků, které je součástí URL adresy a vyskytuje se za posledním zpětným lomítkem [17]. Příkladem může být adresa

$$https://www.example.com/notebook/img-5,$$

kde URL slug je výraz $img-5$.

V předcházejícím výčtu byl uveden pojem *doména*. Slovo doména je zkratka, správné označení zní *doménové jméno*, a jedná se o unikátní adresu na internetu. V kontextu administrace je význam stejný. Jedna administrace může spravovat více domén, které samozřejmě sdílí i databázi. Proto, jak už bylo zmíněno, může pro některou doménu existovat takzvaný *override*, který se použije místo výchozí hodnoty (může se jednat například o jazykovou mutaci). Příkladem domén jsou *www.ukazka.cz* a *www.ukazka.sk*.

Funkce související s výrobci budou popsány v následujících podkapitolách pomocí případů užití. Případy užití jsou taktéž zachyceny na diagramu A.1.

Případ užití (anglicky *use case*) je množina akcí, které vedou k nějakému výsledku a typicky jsou vykonány systémem. Ten, kdo se systémem pracuje, je označován jako účastník a jedná se obecně o roli mimo systém, ve kterém se akce vykonávají. Pro účastníka je důležitý výsledek akce. Případy užití se používají pro sběr požadavků na systém a v tomto případě především k analýze již existujícího systému, protože mnoho těchto požadavků bude později převzato pro návrh nového systému. [18]

1.3.3.1 UC1 Zobrazit seznam výrobců

Administrátor může zobrazit seznam výrobců, kteří jsou v e-shopu definovaní. Systém u každého výrobce zobrazuje jeho jméno a číslo, kterým je dána výrobci priorita pro řazení. Výrobce je možné seřadit podle jména nebo podle priority.

1.3.3.2 UC2 Smazat výrobce

Administrátor může označit jednoho či více výrobců v seznamu výrobců a označené záznamy smazat.

1.3.3.3 UC3 Přidat nového výrobce

Administrátor může přidávat nové výrobce kliknutím na tlačítko *Přidat*, které se nachází nad seznamem všech výrobců. Systém následně uživateli zobrazí formulář s následujícími položkami:

- Název výrobce (povinný) (*),
- SEO klíčové slovo (*),
- Obrázek,
- Řazení,
- V doménách,
- Title (*),
- Klíčová slova (*),
- Meta-popis, nebo též Meta description (*),
- Popis (*).

U položek označených pomocí *(*)* je možné nastavit hodnotu pro konkrétní doménu. Bližší popis s příkladem je uvedený v kapitole 1.3.3.5.

SEO klíčové slovo je ekvivalent, alespoň pro účely tohoto textu, pro dříve uvedený výraz URL slug. Hodnota této položky se vždy ukládá do tabulky `oc_url_alias` (v případě výchozí hodnoty i hodnoty pro konkrétní doménu). Hodnota, která se uloží do databáze do sloupce `query`, bude odpovídat formátu `manufacturer_id=?`, kde bude místo `?` identifikátor výrobce.

1.3.3.4 UC4 Upravit existujícího výrobce

Administrátor může libovolně upravovat informace u již existujících výrobců. Editovat je možno stejné položky, které systém umožňuje zadat při vytváření nového výrobce.

1.3.3.5 UC5 Nastavit hodnotu pro konkrétní doménu

Pro položky, které jsou označeny pomocí *(*)* v předcházející kapitole 1.3.3.3, je možné přidat specifickou hodnotu v závislosti na doméně (viz tabulka `oc_domain_override`).

Pro lepší představu je použití znázorněno na následujícím příkladu.

- Výrobce má nastaveny v poli *V doménách* hodnoty *domena1.cz* a *domena2.com*.
- Výrobce bude mít nastaveno klíčové slovo *příklad*.
- Výrobce bude mít nastaveno klíčové slovo *example* pro doménu *domena2.com*.
- Pro doménu *domena1.cz* je tedy platná hodnota klíčového slova *příklad*.

1.3.4 Kategorie

V této podkapitole budou popsány stávající funkce související s kategoriemi, které se používají pro kategorizaci produktů. Kategorie pomáhají uživatelům snáze najít to, co hledají, a definují určité společné vlastnosti produktů, které se v nich nachází. Příkladem může být filtrování podle zadaných kritérií, která budou pravděpodobně různá v závislosti na kategorii.

Aktuální podoba databáze týkající se kategorií je znázorněna na obrázku 1.5. Význam a popis jednotlivých tabulek je následující.

oc_category obsahuje základní a výchozí informace o kategorii.

oc_category_description obsahuje doplňující informace o kategorii (především delší texty a popisy).

oc_category_banner obsahuje informace o reklamních bannerech, které jsou vždy vázány na určitou kategorii.

oc_category_cache obsahuje předpočítané údaje pro kategorie. Hlavním cílem této tabulky je zrychlení sestavení podoby výsledné kategorie pro danou doménu.

oc_feature obsahuje seznam parametrů. Jedná se například o barvu či jiné vlastnosti produktu. Z historických důvodů má tabulka sloupec `category_id`, ale to se neosvědčilo, a proto se nepoužívá (docházelo k duplikování parametrů).

oc_feature_category obsahuje definici vazby mezi parametrem a kategorií, jedná se tedy o vazební tabulku (vazba M:N). Díky této tabulce nedochází k duplikaci parametrů a zároveň je možné nastavovat výchozí parametry pro kategorie.

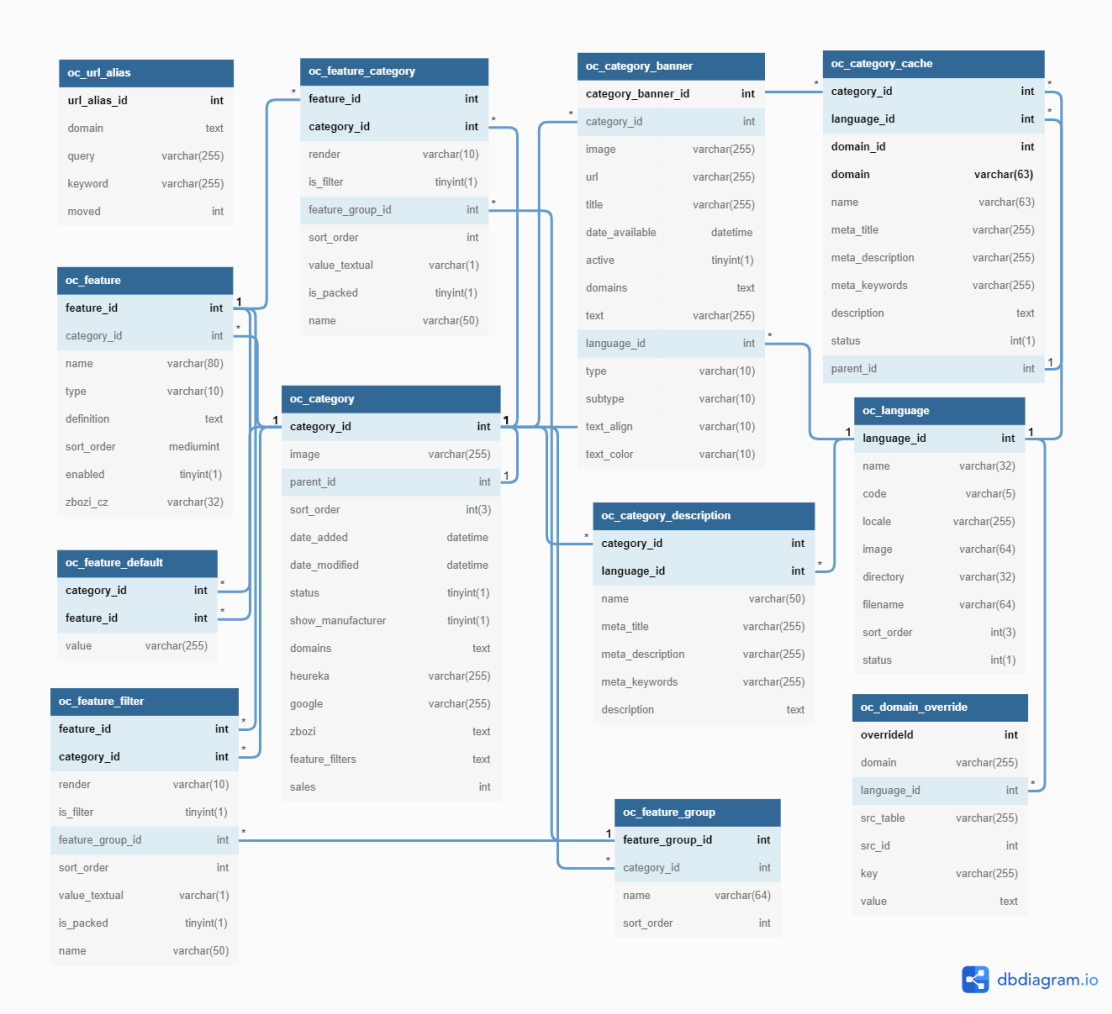
oc_feature_filter obsahuje definici filtrů pro kategorie. Filtr má vazbu na kategorii a parametr. Podle parametru se budou produkty v rámci dané kategorie filtrovat.

oc_feature_group obsahuje skupiny filtrů pro kategorie. Jedná se o seskupení filtrů, kde se každý filtr váže na parametr, který nabývá logické hodnoty pravda (`true`) nebo nepravda (`false`). Příkladem skupiny filtrů může být filtr *Ostatní* se zaškrtávacími možnostmi *Kontrolka provozu*, *Ukazatel množství tekutiny*, *Eko výroba*.

oc_feature_default obsahuje výchozí nastavení filtrů pro danou kategorii.

oc_language obsahuje seznam jazyků a jejich zkratky. Některé tabulky ve schématu používají jazyk jako součást složeného klíče, což se používalo historicky a nyní se už nepoužívá (nebo používá jen minimálně).

Obrázek 1.5 Schéma databáze zachycující tabulky týkající se kategorie.



oc_domain_override obsahuje upřesňující texty pro konkrétní domény. Když neexistuje text v této tabulce, použije se výchozí text definovaný v tabulkách výše (konkrétně se jedná o tabulky `oc_category` a `oc_category_description`).

oc_url_alias je tabulka, která obsahuje takzvané URL slugs. Bližší vysvětlení termínu i tabulky je uvedeno v kapitole 1.3.3.

Funkce související s kategoriemi budou popsány v následujících podkapitolách pomocí případů užití. Případy užití byly kvůli přehlednosti seskupeny do určitých skupin. Každá skupina má i vlastní diagram případů užití uvedený v přílohách. Skupiny jsou:

- základní operace s kategoriemi – diagram A.2,
- bannery – diagram A.3,
- parametry – diagram A.4,
- filtry – diagram A.5.

1.3.4.1 UC6 Zobrazit seznam kategorií

Administrátor může zobrazit seznam kategorií, které jsou v systému vytvořeny. Systém u každé kategorie zobrazuje domény, název kategorie a číslo, kterým je kategorii dána priorita pro řazení. Uživatel může kategorie seřadit podle libovolné z výše uvedených hodnot.

1.3.4.2 UC7 Smazat kategorii

Administrátor může označit jednu či více kategorií v zobrazeném seznamu a označené záznamy smazat.

1.3.4.3 UC8 Přidat novou kategorii

Administrátor může přidávat nové kategorie kliknutím na tlačítko *Přidat*, které se nachází nad zobrazeným seznamem kategorií. Systém následně uživateli zobrazí formulář s následujícími položkami:

- Název kategorie (povinný) (*),
- Title (*),
- Klíčová slova (*),
- Meta Tag Description (*),
- Popis,
- Nadřazená kategorie,
- SEO klíčová slova (*),
- Heuréka kategorie,
- Google kategorie,
- Zbozi.cz kategorie,
- Obrázek,
- Zobrazovat výrobce v názvech,
- Řazení,
- V doménách,
- Status.

Položky označené pomocí *(*)* souvisí s nastavením hodnoty pro konkrétní doménu. Bližší popis je uveden v kapitole 1.3.4.5.

SEO klíčové slovo je ekvivalent, alespoň pro účely tohoto textu, pro dříve uvedený výraz URL slug. Hodnota této položky se vždy ukládá do tabulky `oc_url_alias` (v případě výchozí hodnoty i hodnoty pro konkrétní doménu). Hodnota, která se uloží do databáze do sloupce `query`, bude odpovídat formátu `category_id=?`, kde bude místo `?` identifikátor kategorie.

1.3.4.4 UC9 Upravit existující kategorii

Administrátor může libovolně upravovat informace zobrazené kategorie. Systém umožňuje zobrazit libovolnou kategorii ze seznamu kategorií. V detailu kategorie je možné editovat stejné položky, které je možné zadat při vytváření nové kategorie.

1.3.4.5 UC10 Nastavit hodnotu pro konkrétní doménu

Tato funkčnost je stejná jako pro výrobce, kteří byli popsáni již dříve. Popis s příkladem je k nalezení v kapitole 1.3.3.5.

1.3.4.6 UC11 Zobrazit bannery zvolené kategorie

Administrátor může zobrazit seznam bannerů pro danou kategorii. Pro každý banner jsou v seznamu zobrazeny všechny jeho nastavitelné položky (atributy), jejichž seznam je uveden dále v kapitole 1.3.4.8.

1.3.4.7 UC12 Smazat banner

Administrátor může libovolný banner v zobrazeném seznamu smazat pomocí tlačítka *Odebrat*, které je zobrazeno u každého banneru.

1.3.4.8 UC13 Přidat nový banner

Administrátor může přidávat nové bannery kliknutím na tlačítko *Přidat*, které se nachází pod zobrazeným seznamem bannerů. Systém po kliknutí na tlačítko přidá do seznamu bannerů nový prázdný záznam s následujícími položkami:

- Jazyk (povinný)
- Umístění (povinný),
- Soubor, neboli obrázek,
- Text,
- Odkaz,
- Zarovnání (povinný),
- Barva (povinný),
- Cíl,
- Domény,
- Stav (povinný).

1.3.4.9 UC14 Upravit banner

Administrátor může libovolně upravovat data bannerů přímo z přehledu se seznamem všech bannerů. Editace jednotlivých položek banneru je ekvivalentní s nastavováním banneru napoprvé, tj. po jeho přidání.

1.3.4.10 UC15 Zobrazit parametry zvolené kategorie

Administrátor může zobrazit všechny parametry, které jsou přiřazené k dané kategorii. Tyto parametry poté dědí produkty, které jsou v dané kategorii.

1.3.4.11 UC16 Odebrat parametr z dané kategorie

Administrátor může odebrat parametr z dané kategorie. Parametr touto akcí nezaniká, pouze je smazán z databáze z vazební tabulky. Úplné smazání parametru není v současné administraci umožněno.

1.3.4.12 UC17 Přidat parametr do dané kategorie

Administrátor může přiřadit již existující parametr k dané kategorii. Tento parametr je již vytvořený a nelze editovat, lze jej pouze přiřadit do zvolené kategorie.

1.3.4.13 UC18 Přidat či upravit parametr

Přidat nový parametr je možné podobně jako přiřadit již existující parametr do dané kategorie. Místo výběru existujícího parametru administrátor napíše jméno parametru, zvolí jeho druh, jednotku a data uloží.

Editovat již existující parametr není v současné administraci systémem umožněno.

1.3.4.14 UC19 Zobrazit filtry zvolené kategorie

Administrátor může zobrazit seznam všech filtrů, které jsou nastaveny pro danou kategorii. V seznamu jsou zobrazeny položky: název parametru, název filtru, způsob zobrazení daného filtru a výchozí stav. Výchozí stav označuje, zda je banner ve výchozím stavu rozbalený či sbalený.

1.3.4.15 UC20 Smazat filtr nebo smazat všechny filtry

Administrátor může libovolný filtr smazat pomocí tlačítka *Zrušit*. Systém také umožňuje smazání všech filtrů pomocí tlačítka *Smazat všechny filtry*. Tyto změny je nutné ještě potvrdit pomocí tlačítka pro uložení změn.

1.3.4.16 UC21 Přidat nový filtr nebo přidat skupinu filtrů

Administrátor může přidat nový filtr pomocí tlačítka *Přidat filtr*. Následně je nutné vyplnit název parametru, jméno filtru, způsob zobrazení a výchozí stav. Výchozí stav udává, zda je filtr rozbalený či sbalený.

Systém dále umožňuje přidat skupinu filtrů, u které se definuje název filtru a výchozí stav. Následně je pak možné do této skupiny přidávat filtry.

1.3.4.17 UC22 Upravit filtr

Nastavení filtrů je možné editovat. Administrátor může editovat libovolnou položku filtru (viz položky vyjmenované v předchozí kapitole). Dále je možné měnit pořadí filtrů pomocí tlačítek *Nahoru* a *Dolů*.

1.3.4.18 UC23 Zkopírovat všechny filtry z jiné kategorie

Poslední funkce, kterou současná administrace poskytuje, je kopírování filtrů z jiné kategorie. Systém umožňuje zkopírování všech filtrů jedné kategorie do jiné kategorie. Cílem této funkce je usnadnit administrátorovi práci, aby nemusel opakovaně zadávat stejná data.

1.3.5 Produkty

Tato podkapitola bude vynechána, neboť toto téma zpracovává kolega Alois Kouba ve své bakalářské práci. Autor této práce přišel do kontaktu s tímto tématem a účastnil se několika schůzek týkajících se tohoto tématu, avšak na analýze či návrhu se nijak významně nepodílel.

1.3.6 Objednávky

V této podkapitole budou popsány funkce současné administrace, které mají přímou souvislost s objednávkami. Objednávky jsou základním kamenem každého e-shopu, ale zároveň se jedná o velmi komplexní téma, které má mnoho přesahů - skladovací systém, sledování zásilek, správa faktur, dobropisy a samozřejmě samotná správa objednávek.

Aktuální podoba databáze týkající se objednávek je znázorněna na obrázku 1.6. Význam a popis jednotlivých tabulek je uveden níže. Některé tabulky neobsahují kompletní seznam atributů (daná tabulka obsahuje na svém konci text *a další ...*). Atributy byly vynechány u tabulek, které byly uvedeny pouze pro účely ilustrace vazby, nebo u tabulek, které mají atributů příliš mnoho a kompletní seznam nebylo možno uvést.

oc_order obsahuje hlavní informace o objednávce (doručovací a fakturační údaje, údaje o zákazníkovi a další). Kvůli přehlednosti jsou doručovací a platební informace ve schématu sloučeny pomocí znaku *. Jedná se například o doručovací/fakturační jméno a příjmení a adresu, platební metodu, způsob dopravy a další.

oc_order_product obsahuje seznam objednaných produktů a informace o nich.

oc_order_option obsahuje informace o objednaných produktech, které odpovídají tabulce `oc_product_option_value` (viz níže).

oc_order_status obsahuje seznam stavů, ve kterých se může objednávka nacházet (čeká na uhrazení, k expedici atd.).

oc_order_history obsahuje historii stavů objednávky.

oc_order_product_move obsahuje informace o jednotlivých produktech v rámci objednávky (počet produktů v interním skladu a počet produktů v externím skladu).

oc_order_total obsahuje konečné ceny pro objednávku (cena s DPH, cena bez DPH apod.).

oc_coupon obsahuje kupóny (použité i nepoužité).

oc_setting obsahuje různá nastavení a informace o způsobech doručení a plateb.

oc_domain_setup obsahuje nastavení pro domény a informace o způsobech doručení a plateb.

oc_language obsahuje seznam jazyků a jejich zkratky.

oc_currency obsahuje seznam měn.

oc_review obsahuje hodnocení zákazníků.

oc_customer obsahuje informace o zákaznících.

oc_product obsahuje informace o produktech.

oc_product_option_value obsahuje doplňující informace o produktu (zcela typickým příkladem jsou velikosti bot).

Funkce související s objednávkami budou popsány v následujících podkapitolách pomocí případů užití. Případy užití jsou taktéž zachyceny pomocí diagramů v přílohách – diagramy A.6 a A.7.

Obrázek 1.6 Schéma databáze zachycující tabulky týkající se objednávky.

1.3.6.1 UC24 Zobrazit seznam objednávek

Systém umožňuje zobrazit seznam objednávek, které jsou v systému vytvořeny. U každé objednávky v seznamu jsou uvedeny následující informace:

- identifikátor objednávky,
- jméno zákazníka,
- číslo faktury,
- doména,
- stav objednávky,
- způsob dopravy,
- datum vytvoření,
- zda byla objednávka již uhrazena,
- celková cena a zisk,
- odkud zákazník na stránky e-shopu přišel,
- zda jsou všechny požadované produkty skladem,
- zda zákazník udělil souhlas s GDPR.

Objednávka má pouze omezený počet stavů, kterých může nabývat. Systém umožňuje tyto stavy editovat, přidávat či mazat. Příklady možných stavů jsou: Čeká na uhrazení, Odesláno dopravcem, Nevyřízeno, K expedici.

1.3.6.2 UC25 Filtrovat objednávky podle zadaných kritérií

Systém umožňuje filtrovat objednávky (v seznamu objednávek) podle všech kritérií uvedených v předchozím případu užití 1.3.6.1 (kromě informace, zda dal uživatel souhlas s GDPR). Konkrétně je možno předvyplnit údaje, podle kterých se má filtrovat, a následně stisknout tlačítko *Filtrovat*.

1.3.6.3 UC26 Smazat objednávky

Administrátor může označit jednu či více objednávek v zobrazeném seznamu objednávek a označené záznamy smazat.

1.3.6.4 UC27 Zobrazit detail objednávky

Systém umožňuje zobrazit detail objednávky, kde jsou uvedeny další údaje, například seznam objednaných produktů. Dále je zde umožněno upravovat objednávku (viz dále).

1.3.6.5 UC28 Zobrazit fakturu zvolené objednávky

Administrátor může v seznamu objednávek označit jednu či více objednávek a následně kliknout na tlačítko *Faktury*. Systém následně v novém okně zobrazí faktury pro vybrané objednávky.

1.3.6.6 UC29 Zobrazit a vytisknout produkty v určeném stavu

Administrátor může v seznamu objednávek vybrat z rozbalovacího seznamu jeden z možných stavů, který může objednávka nabývat. Systém následně v novém okně zobrazí všechny produkty, jejichž objednávka je aktuálně v požadovaném stavu. Formátování stránky je vhodné pro tisk, který je možno vyvolat přes pravé tlačítko myši v prohlížeči (možnost tisku je nabízena prohlížečem).

1.3.6.7 UC30 Zobrazit a upravit základní informace o objednávce

Po otevření detailu objednávky systém zobrazí základní informace – podrobnosti o objednávce, kontaktní údaje, adresy. Podrobnosti o objednávce zahrnují identifikátor, datum vytvoření, způsob zaplacení, způsob dopravy, číslo faktury a název domény. Mezi kontaktní údaje patří email, telefon a fax. Adresy jsou dvě - fakturační adresa a adresa pro doručení. Obě adresy se skládají z běžných položek jako je město, ulice atd.

Systém umožňuje kontaktní údaje a adresy upravovat. Naopak vyjmenované podrobnosti o objednávce upravovat nelze.

1.3.6.8 UC31 Zobrazit a upravit seznam produktů v objednávce

V detailu objednávky se zobrazí kompletní seznam produktů, které byly objednány. Dále je uvedena cena bez DPH, kolik činí DPH a součet těchto dvou položek (tedy cena včetně DPH).

Produkty je možno do objednávky přidávat nebo je z objednávky mazat. Systém také umožňuje upravovat produkty v objednávce (název produktu, délku záruky, počet objednaných kusů aj.)

1.3.6.9 UC32 Zobrazit historii stavů objednávky

V detailu objednávky se zobrazí kompletní historie stavů objednávky. Každý záznam má datum, volitelný komentář, stav objednávky a zda byl zákazník informován o daném stavu objednávky.

1.3.6.10 UC33 Přidat nový stav objednávky

V detailu objednávky je možno přidat nový stav objednávky (například změnit stav z *Nevyřízeno* na *K expedici*). Při změně stavu je možno určit, zda má být zákazník informován. Dále je ke změně možno přidat doprovodný komentář.

1.3.6.11 UC34 Zobrazit zásilky (balíčky)

Systém v detailu objednávky zobrazuje zásilky (nebo též balíčky), které byly odeslány k zákazníkovi (nebo odeslány budou).

1.3.6.12 UC35 Zobrazit nebo vystavit fakturu pro objednávku

U každého detailu objednávky se nachází tlačítko pro zobrazení faktury nebo pro její vystavení. Takto zobrazenou fakturu je možno vytisknout.

1.3.6.13 UC36 Označit objednávku jako zaplacenou

Systém umožňuje v detailu objednávky nastavit objednávku jako zaplacenou.

1.3.6.14 UC37 Dobropisovat vybrané produkty objednávky

Administrátor může libovolný z objednaných produktů dobropisovat. Jelikož je označení *dobropis* spíše lidový výraz, bylo by lepší používat označení *opravný daňový doklad*. Jelikož je však označení *dobropis* vžité a v systému běžně používané, bude užíváno i v tomto textu.

1.4 Formulace požadavků

Analýza současného řešení, která byla provedena v přecházejících kapitolách, není rozhodně kompletní, neboť současná administrace je velice rozsáhlá a nebylo předmětem této práce pokrýt celou její funkčnost. Proto byla analyzována pouze část administrace, která poskytuje zásadní funkčnost, a pravděpodobně na tuto analýzu později naváže další práce.

Na základě provedené analýzy je vhodný čas formulovat požadavky, které budou kladeny na nový systém pro administraci. Cílem této kapitoly je definovat jasně a přesně funkce, které bude nový systém poskytovat. To se provádí pomocí specifikace požadavků, což někdy bývá označováno zkratkou SRS. Specifikace požadavků se dělí na funkční a nefunkční požadavky. Tyto požadavky budou omezeny pouze na serverovou část aplikace, neboť klientská část aplikace je řešena odděleně kolegy Janem Babákem a Martinem Dvořákem. [19]

Požadavky bývají často popisovány pomocí modelů MoSCoW a FURPS. A i v této práci budou použity tyto klasifikační modely. Nejprve je však vhodné tyto modely představit a popsat.

1.4.1 FURPS model

Označení této klasifikační metody je zkratkou z několika anglických slov, která budou uvedena později. Tento model byl poprvé představen společností Hewlett-Packard a od prvního použití se stal velmi populární ke klasifikaci požadavků. [20]

Dle [20] je význam jednotlivých písmen ve zkratce následující:

- **F**unkcionality (funkčnost) – popisuje hlavní provozní funkce systému,
- **U**sability (použitelnost, vhodnost k použití) – definuje přístupnost, aneb jak budou se systémem interagovat jiné systémy, procesy či lidé (v pozici uživatele),
- **R**eliability (spolehlivost) – popisuje robustnost a odolnost systému proti chybám, případně možnosti obnovení provozu a zotavení se z výpadku,
- **P**erformance (výkon) – popisuje výkonnost a efektivitu, například rychlost odezvy systému, zatížení síťového provozu,
- **S**upportability (udržitelnost a rozšiřitelnost) – popisuje údržbu systému, jeho modularitu, rozšířitelnost a jak často bude nutné komponenty systému aktualizovat.

1.4.2 MoSCoW model

Název tohoto modelu je rovněž složen z několika anglických slov, která budou uvedena později. Tento model byl původně součástí jiné metodologie pro vývoj softwaru (*Dynamic System Development Method*). Cílem bylo určit priority požadavků pro vyvíjený systém. Tato klasifikace se však stala velmi oblíbenou a začala se používat pro stanovení priorit požadavků na systém. [20]

Dle [20] je význam jednotlivých písmen ve zkratce následující:

- Mo – Must have (musí mít) – definuje požadavky, které mají nejvyšší možnou prioritu,
- S –- Should have (mělo by mít) – definuje požadavky, které mají vysokou prioritu a měly by být součástí výstupu, pokud je to možné,

- Co -– Could have (co by bylo dobré, kdyby mělo) – definuje požadavky s nejnižší prioritou, které jsou žádoucí a bylo by dobré, kdyby byly součástí výstupu, pokud to neznamená příliš mnoho práce navíc a zvýšení nákladů,
- W – Won't have (zatím nebude mít) - definuje požadavky, které jsou do budoucna chtěné, ale zatím nebudou zahrnuty.

1.4.3 Funkční požadavky

"Funkční požadavky jsou takové vlastnosti a funkce výsledného produktu, které musí být implementovány, aby uživatel mohl provést požadovanou akci a dosáhnout tak svého cíle. Funkční požadavky tedy popisují chování systému za určitých podmínek a musí být jasně definovány, aby jim stejně rozuměl zadavatel i programátor."[2] [19, překlad autor]

Všechny požadavky uvedené v této podkapitole spadají v modelu FURPS do kategorie označené písmenem F, tedy popisují funkčnost systému. V hranaté závorce za funkčním požadavkem bude vždy uvedena priorita dle modelu MoSCoW.

1.4.3.1 Evidence výrobců

Systém bude umožňovat správu výrobců, kteří jsou v systému evidovaní a jejichž výrobky mohou být potencionálně v e-shopech nabízeny. Tato funkcionalita konkrétně zahrnuje:

- **F1** Přidání nového výrobce, smazání výrobce nebo editace existujícího výrobce [Mo]
- **F2** Získání seznamu všech výrobců [Mo]
- **F3** Získání informací o konkrétním výrobci (detail výrobce) [Mo]
- **F4** Možnost nastavení pořadového čísla daného výrobce [Mo]
- **F5** Automatický přepočet pořadí ostatních výrobců při změně pořadí jednoho výrobce [S]
- **F6** Nastavení obrázku výrobce podle domény (obrázek by se mohl lišit podle domény) [Co]

V rámci detailu výrobce je nutné, aby systém poskytoval minimálně stejné informace, jako poskytuje současný systém pro administraci, kapitola 1.3.3.3.

1.4.3.2 Evidence kategorií

Systém bude poskytovat rozhraní pro správu kategorií, které slouží pro třídění produktů do skupin. Systém bude rovněž poskytovat možnost práce s parametry, filtry a bannery.

- **F7** Získání seznamu všech kategorií [Mo]
- **F8** Získání informací o konkrétní kategorii (detail kategorie) [Mo]
- **F9** Přidání nové kategorie, smazání kategorie nebo editaci existující kategorie [Mo]
- **F10** Možnost nastavení pořadového čísla a změna rodičovské kategorie [Mo]
- **F11** Automatický přepočet pořadí ostatních kategorií při změně pořadí jedné kategorie [S]
- **F12** Nastavení obrázku kategorie podle domény (obrázek by se mohl lišit podle domény) [Co]

[2] Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks. So, it's important to make them clear both for the development team and the stakeholders. Generally, functional requirements describe system behavior under specific conditions.

V rámci detailu kategorie bude systém poskytovat stejné informace, jako poskytuje současný systém pro administraci, kapitola 1.3.4.3. V předcházejícím seznamu byl uveden pojem rodičovská kategorie. Znamená to, že kategorie je zařazena do jiné kategorie a mohou tudíž vznikat stromy kategorií. Například pro kategorii *Nářadí* může strom kategorií vypadat následovně:

- Nářadí
  - Kleště
  - Šroubováky
    - Sady
    - Torx
    - Příslušenství
  - Vrtačky

Požadavek na automatický přepočet pořadového čísla ostatních kategorií (při změně pořadového čísla jedné kategorie) je v případě kategorií komplikovanější, protože kategorie obsahují nadřazenou kategorii. Řadící algoritmus by musel tuto hodnotu zohledňovat, neboť kategorie se primárně řadí hierarchicky. Teprve kategorie na stejné úrovni, se stejným předkem, se řadí podle pořadového čísla.

1.4.3.3 Správa parametrů pro kategorie

Systém bude podporovat evidenci parametrů, a to ve vztahu ke kategoriím. Jeden parametr může být přiřazen k více kategoriím, aby nedocházelo ke zbytečné duplikaci parametrů. Parametrem se zde myslí například barva, velikost nebo jiná zobecněná vlastnost produktu.

- **F13** Získání přehledu všech dostupných parametrů a získání všech parametrů pro zvolenou kategorii [Mo]
- **F14** Upravení parametrů nějaké kategorie (přidání, smazání, editace) [Mo]

Každý parametr se skládá z identifikátoru, názvu, typu a jednotky. Typ parametru může nabývat pouze jedné z následujících hodnot:

text parametrem je textový řetězec, například *materiál*,

unit parametrem je jednotka, například *délka* nebo *hmotnost*,

bool parametrem je hodnota ano či ne, například *vodotěsný* nebo *zařízení s displayem*,

file parametrem je soubor, například *návod*,

system označení pro výchozí parametry, například *barva* či *cena*.

1.4.3.4 Správa filtrů pro kategorie

Filtr má přímý vztah k parametru a kategorii, je to způsob filtrování v rámci dané kategorie, které je založené na parametru přiřazenému dané kategorii.

- **F15** Získání všech filtrů pro danou kategorii [Mo]
- **F16** Přidání, smazání a editace filtru dané kategorie [Mo]
- **F17** Kontrola typů filtrů zařazovaných do skupiny (jediný povolený typ filtru je `bool`, viz dále) [S]

Každý filtr musí mít vazbu na parametr, kategorii, název, typ filtru a zda je ve výchozím stavu sbalený či rozbalený. Typ filtru je opět položka, která má pouze omezené spektrum hodnot, jichž může nabývat. Pro typ filtru také existují pravidla, která souvisejí s typem parametru.

range představuje rozsah od–do a vyžaduje vazbu na parametr typu `unit`,

enum představuje výčet možností, podle kterých je možné filtrovat, a vyžaduje vazbu na parametr typu `text` nebo `unit`,

bool představuje filtr, který se buď aplikuje, nebo ne, a vyžaduje vazbu na parametr typu `bool`,

system vyžaduje vazbu na parametr typu `system`.

Filtry mohou být organizovány ještě do skupin. Příkladem by mohla být skupina pojmenovaná *Ostatní*, ve které bude několik dalších filtrů typu `bool`. Kontrolu na typ filtru ve skupině filtrů současný systém pro administraci nemá.

1.4.3.5 Správa bannerů pro kategorie

Systém bude umožňovat správu bannerů. Banner bude vždy přidán konkrétní kategorii a doméně. Systém musí umožnit zobrazit bannery, jejich editaci i smazání.

- **F18** Získání všech bannerů pro danou kategorii [Mo]
- **F19** Přidání, smazání a editace bannerů dané kategorie [Mo]

Banner má ve stávajícím systému několik položek, které mohou nabývat pouze omezeného množství hodnot. Jedná se o stav, barvu, zarovnání, umístění. U těchto položek je nutností zachovat funkcionalitu, a tedy kontrolovat nastavované hodnoty.

1.4.3.6 Správa URL slug

Nechť je tato funkcionalita označena jako F20 a je klasifikována jako [Mo]. Systém bude u výrobců i kategorií kontrolovat nastavované hodnoty URL slug. Pravidla týkající se URL slug jsou uvedena v následujícím seznamu.

- Pokud není definována doména, tak je URL slug platný pro všechny domény.
- Pokud je definována doména, tak je URL slug platný pouze pro tuto jednu doménu.
- URL slug musí být vždy unikátní pro danou doménu (pokud doména není uvedena, tak musí být unikátní napříč všemi doménami).
- Při změně URL slug výrazu se záznam nepřepisuje, ale vytvoří se nový. Starý záznam dostane odkaz na nový URL slug. Toto pravidlo je nutností kvůli funkcionalitě přesměrovávání.

1.4.3.7 Správa obrázků a souborů

Vyvíjený systém by měl podporovat práci se soubory. Tato funkcionalita by měla být primárně zaměřena na práci s obrázky a PDF soubory. Dále všechny soubory budou samozřejmě zařazeny do složek. Tato funkcionalita se skládá z několika požadavků vyjmenovaných níže.

- **F21** Nahrávání a mazání souborů [Mo]
- **F22** Přesouvání a kopírování souborů [S]
- **F23** Vytváření, mazání složek [Mo]

- **F24** Přesouvání a kopírování složek [S]
- **F25** Vypsání seznamu všech složek a obsahu konkrétní složky [Mo]
- **F26** Stažení dříve nahraného souboru [Mo]
- **F27** Při přesunutí složky nebo souboru aktualizovat všechny reference na soubory [S]

Poslední uvedený požadavek se vztahuje například k výrobcům a kategoriím, kde je možné dané entitě nastavit obrázek. Pokud je tento obrázek přesunut, tak je nutné aktualizovat odpovídající záznam v databázi.

1.4.3.8 Párování externích kategorií

Z analýzy vyplývá, že v aktuální systému jsou tři externí kategorie - Heureka kategorie, Google kategorie a Zbozi.cz kategorie. Pro správu externích kategorií byly definovány následující požadavky.

- **F28** Manuální spuštění aktualizace kategorií [Mo]
- **F29** Automatické a pravidelné spuštění aktualizace kategorií [Co]
- **F30** Párování (propojování) externích kategorií s kategoriemi (a produkty) uvnitř systému [Mo]
- **F31** Automatická aktualizace vazeb externích kategorií se spárovanými kategoriemi [S]

Požadavek F31 je velmi žádoucí, neboť aktuální administrace tuto funkcionalitu nenabízí. Oprava by tak musela probíhat manuálně.

1.4.3.9 Evidence objednávek

Systém bude poskytovat správu objednávek, která se opět skládá z několika funkčních požadavků.

- **F32** Získání seznamu všech objednávek [Mo]
- **F33** Získání informace o konkrétní objednávce (detail objednávky) [Mo]
- **F34** Editace kontaktních údajů na zákazníka a jím vyplněné adresy [Mo]
- **F35** Editace produktů v objednávce (přidávat, odebírat, upravovat) [Mo]
- **F36** Přidání nového stavu objednávky (s komentářem) [Mo]
- **F37** Získání informací o možných způsobech dopravy a platby [S]

Detail objednávky by měl obsahovat stejné informace, jako poskytuje současný systém pro administraci popsaný v podkapitolách 1.3.6. Především se bude jednat o základní informace o objednávce, zákazníkovi kontaktní údaje, adresa doručení, fakturační adresa, seznam produktů, historie stavů objednávky. Seznam zásilek nový systém poskytovat nebude, neboť tuto funkcionalitu může poskytovat jiná aplikace vyvíjená společností Jagu, s. r. o.

Funkční požadavek F37 je důležitý především pro úpravu způsobu doručení a způsobu platby. Proto by měl systém poskytovat podpůrné informace o těchto službách.

1.4.3.10 Evidence dobropisů

Systém bude poskytovat seznam dobropisů pro danou objednávku s přehledem produktů obsažených v dobropisech. Také musí být umožněno produkty z objednávky dobropisovat. Dobropis by měl mít povinný komentář s odůvodněním.

- **F38** Získání seznamu všech dobropisů pro konkrétní objednávku [Mo]
- **F39** Získání seznamu všech dobropisovaných produktů z objednávky [Mo]
- **F40** Vytvoření či editace dobropisu [Mo]

1.4.3.11 Vytvoření a zobrazení faktury

Systém by také mohl poskytovat možnost pro práci s fakturami. Zobrazenou fakturu by mělo být možno vytisknout.

- **F41** Vytvoření (vygenerování) faktury [W]
- **F42** Zobrazení faktury [W]
- **F43** Nastavení, že je objednávka zaplacena [S]

1.4.4 Nefunkční požadavky

"Nefunkční požadavky nemají souvislost s funkčností systému. Spíše definují, jak má systém fungovat."[3] [19, překlad autor]

Tyto požadavky tedy popisují obecné charakteristiky systému, které se týkají například bezpečnosti, dostupnosti nebo výkonnosti systému. U těchto požadavků nebude specifikována klasifikace podle MoSCoW, neboť všechny požadavky mají vysokou prioritu. Klasifikace podle FURPS bude uvedena v hranatých závorkách za názvem požadavku.

1.4.4.1 N1 Architektura [S]

Aplikace bude rozdělena do několika vrstev na základě vhodného návrhového vzoru.

1.4.4.2 N2 Autentizace a autorizace [R/F]

Se systémem pro administraci e-shopu bude moci pracovat pouze přihlášená osoba. Daná osoba bude mít přidělené role a podle toho bude možné omezit její přístup k vybraným částem systému.

1.4.4.3 N3 Volba technologií [S]

V průběhu návrhu budou zohledněny technologie, které již firma Jagu, s. r. o., používá pro jiné vyvíjené informační systémy. Tento požadavek byl přijat pro snížení různorodosti používaných technologií, což obecně vede ke snazší správě portfólia používaných technologií.

1.4.4.4 N4 API rozhraní [U]

Aplikace bude vystavovat API rozhraní, pomocí kterého bude aplikace komunikovat s vnějším prostředím. Existence API bude umožňovat například přidání mobilní aplikace, která by toto rozhraní použila.

[3] Nonfunctional requirements, not related to the system functionality, rather define how the system should perform.

1.4.4.5 N5 Databáze [S]

Serverová část aplikace bude využívat již existující databázi a schéma. Schéma databáze bylo již popsáno v předchozích kapitolách analýzy a toto schéma je nutné využít. V rámci návrhu a implementace je možné přidat nové sloupce a tabulky, ale není možné měnit strukturu databáze ani zavádět DB omezení jako constraints nebo foreign key. Důvodem tohoto omezení je, že změny by mohly narušit funkčnost současné administrace.

1.4.4.6 N6 Dokumentace [U]

Rozhraní aplikace bude řádně zdokumentováno.

Kapitola 2
Návrh

V této kapitole bude popsán návrh nového systému pro administraci e-shopů. Návrh musí vycházet ze specifikace funkčních a nefunkčních požadavků, které jsou uvedeny v předešlé kapitole. Jelikož na systém bylo kladeno i několik požadavků, které souvisí s architekturou, tak prvním tématem návrhu je právě architektura nového systému.

V další kapitole již bude proveden návrh nového systému, následovaný problematikou autentizace/autorizace a celá tato kapitola bude zakončena seznamem a popisem zvolených technologií.

2.1 Architektura

Nejprve je nutné provést rozhodnutí týkající se architektury a rozdělit pomyslný systém do několika komponent na základě analýzy, především nefunkčních požadavků kladených na systém.

Z požadavku N4 na API (kapitola 1.4.4.4) vyplývá, že systém se bude dělit na serverovou aplikaci a klientskou aplikaci. Tento fakt je zřejmý a cílený již od samého začátku projektu. Systém bude používat architekturu klient–server implementovanou pomocí standardního webového řešení, tedy jeden server, který bude obsluhovat mnoho uživatelů. Systém pro administraci bude představovat serverovou část a klient bude jeho uživatel. Dále dle požadavku N6 na dokumentaci je nutné API zdokumentovat.

Systém pro administraci bude přistupovat k již existující databázi dle N5 (kapitole 1.4.4.5). Toto omezení je velmi důležité, neboť stávající databáze dle analýzy neobsahuje validaci dat, a proto tuto funkci musí zastávat vyvíjený systém. Do budoucna by databáze určitě měla převzít roli validace dat (alespoň částečné pomocí constraints, foreign key a podobných nástrojů, které jsou dnes již běžné), ale to v současné situaci není možné. Proto je tento problém ponechán k vyřešení do budoucnosti.

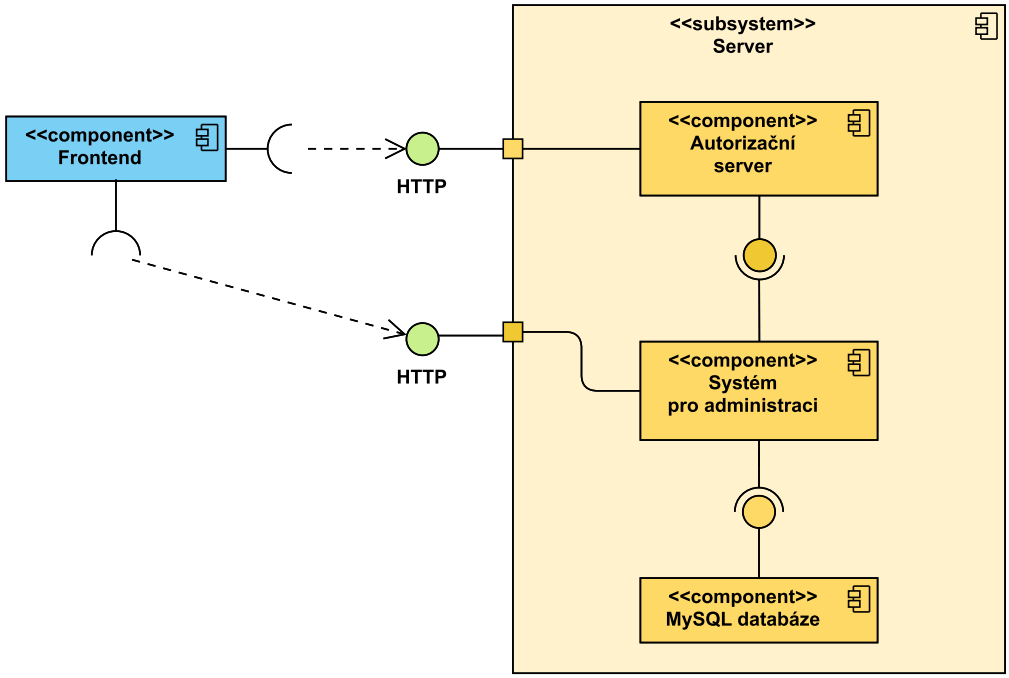
Posledním bodem, který zde bude popsán, je požadavek N2 na autentizaci a autorizaci (kapitola 1.4.4.2). Tento požadavek se týká bezpečnosti a bude splněn s využitím autorizačního serveru. Frontend provede autentizaci vůči autorizačnímu serveru a získá token, který pak přidá do posílaného požadavku na backend. Backend bude následně provádět validaci tokenu a autorizaci na základě rolí uživatele. Detailnější popis bude uveden později v příslušné kapitole 2.3.

Pro lepší představu je popsaná architektura zachycena na obrázku 2.1.

2.2 Systém pro administraci

V této kapitole bude představen návrh komponenty, která byla na obrázku nazvána jako Systém pro administraci. V následujících podkapitolách bude představen návrh, který pokrývá funkční

Obrázek 2.1 Návrh architektury systému



požadavky uvedené v kapitole 1.4.3. Nejprve bude představen návrh architektury pro tuto komponentu a později budou ukázány návrhy řešení pro výrobce, kategorie a tak dále.

2.2.1 Návrh architektury a docker

Tato komponenta bude realizována v jazyce PHP s využitím frameworku Symfony[1], formou REST API. Framework Symfony byl zvolen proto, že firma Jagu, s. r. o., tento framework již používá v jiných projektech. Zároveň je tento framework používaný a má velmi širokou základnu uživatelů [21]. Více informací v kapitole 2.4.

Nový systém bude komunikovat s autorizačním serverem, jehož výběr a popis bude uveden později v kapitole 2.3.

Dále bude komponenta komunikovat s již existující databází MySQL[2]. Přístup bude realizován pomocí objektově relačního mapování (ORM), který poskytuje například framework Doctrine[3]. Doctrine je nástroj, který je frameworkem Symfony doporučován. Více informací o těchto technologiích bude uvedeno v kapitole 2.4.

2.2.1.1 MVC architektura systému pro administraci

Vyvíjená aplikace bude vnitřně rozdělena na několik částí dle požadavku N1 na architekturu aplikace (kapitola 1.4.4.1). Framework Symfony je postaven na základech MVC architektury. MVC architektura se skládá ze tří vrstev:

- Model - představuje data, se kterými aplikace pracuje,
- View - představuje vrstvu, která zobrazuje data uživateli,

[1]Odkaz na webové stránky frameworku Symfony je https://symfony.com.
[3]Odkaz na webové stránky databáze MySQL je https://www.mysql.com.
[3]Odkaz na webové stránky frameworku Doctrine je https://www.doctrine-project.org.

- Controller - reaguje na uživatelské akce a odpovídajícím způsobem upravuje data. [22][23]

Vrstva View není náplní této práce, neboť klientská aplikace (frontend) není součástí této práce. Serverová aplikace bude pro ostatní aplikace poskytovat data ve formátu JSON. Tento formát se používá pro výměnu dat mezi serverovou a klientskou aplikací, která se stará o grafickou reprezentaci dat a interakci s uživatelem. [24]

Vrstva Controller bude rozdělena na dvě části. První část se bude skládat z objektů nazvaných jako Controller, které budou definovat přístupové body (nebo též endpointy), obsluhovat přijaté požadavky, konvertovat přijatá data na interní objekty a provádět základní validaci dat. Vykonání požadované akce budou zajišťovat objekty dále označované jako Service (*Služba*). Objekty Service tedy budou obsahovat takzvanou business logiku.

Vrstva Model bude také rozdělena na několik částí. Framework Symfony definuje Database Abstraction Layer (abstraktní vrstvu pro práci s databází), kterou pro tuto práci bude využívat již zmíněný framework Doctrine. Tyto frameworky definují objekty Entity a Repository. Entity je objekt, který představuje záznam uložený v databázi (příkladem může být jeden konkrétní výrobce). Repository je objekt, který umožňuje práci s databází pro konkrétní tabulku (například tabulku výrobců). Pomocí Repository je možné získat například konkrétní Entity objekt výrobce. [25]

Vrstva Model bude ještě rozšířena o objekty, které budou souhrnně označovány jako DTO (*Data Transport Object*, nebo jen TO). Důvodem pro zavedení těchto objektů je snaha vytvořit další vrstvu mezi objekty z databáze a vrstvou Controller. Cílem je, aby se přijatá data ve vrstvě Controller vždy konvertovala na transportní objekt. Tedy při směru od klienta se provede validace a konverze dat na DTO objekty. S těmito objekty budou následně pracovat Service objekty. V opačném směru, tedy ze systému ven ke klientovi, se data načtou z databáze, provede se konverze na transportní objekty a serializovaná data se odešlou klientovi.

Důvodem pro zavedení DTO objektů je snaha poskytnout výstupy, které jsou odstíněny od reprezentace v databázi, která aktuálně není ideální, a poskytnout vhodnější uspořádání dat. Tyto objekty budou později vždy popsány v samostatných podkapitolách nazvaných Doménový model. V těchto podkapitolách nebudou popsány třídy, na které se mapuje datový model, protože toto mapování přesně odpovídá datovému modelu a nepřineslo by čtenáři žádnou novou informaci. Naopak vztahy a popisy DTO objektů mohou být pro čtenáře užitečnější a zajímavější.

2.2.1.2 Docker

Docker je platforma pro oddělení jednotlivých komponent systému a poskytuje nástroje pro správu prostředí, ve kterém budou spuštěny. Více o této platformě bude uvedeno v kapitole 2.4, kde budou popsány všechny použité technologie.

Pro tuto práci se bude docker skládat ze tří hlavních kontejnerů:

- databáze - tento kontejner se může lišit v závislosti na tom, zda bude databáze lokální nebo vzdálená,
  - lokální databáze - kontejner bude obsahovat kompletní prostředí databázového systému,
  - vzdálená databáze - kontejner bude zprostředkovávat připojení k serveru, na kterém bude dostupná databáze,
- webový server pro zpracování HTTP požadavků,
- PHP - kontejner, který bude představovat modul s PHP.

2.2.2 Výrobce

V této podkapitole bude představen návrh zaměřující se na výrobce. Výrobci byli zpracováni jako první, neboť se jedná o jednodušší část systému, která nemá příliš vazeb na okolní části systému.

Tabulka 2.1 Specifikace API administrace pro výrobce

#	URI	Metoda	Popis
1	/manufacturers	GET	Vrátí seznam všech výrobců.
2	/manufacturers	POST	Vytvoří nového výrobce.
3	/manufacturers/{id}	GET	Vrátí informace o výrobci.
4	/manufacturers/{id}	DELETE	Smaže daného výrobce.
5	/manufacturers/{id}/image	PUT	Nastaví danému výrobci zadaný obrázek.
6	/manufacturers/{id}/order	PUT	Nastaví danému výrobci zadané pořadí a přepočítá pořadí ostatních výrobců.
7	/manufacturers/{id}/domains	POST	Přidá danému výrobci novou doménu se zadanými daty.
8	/manufacturers/{id}/domains/{domain}	GET	Vrátí informace o výrobci pro danou doménu.
9	/manufacturers/{id}/domains/{domain}	PUT	Nastaví danému výrobci pro danou doménu zadaná data.
10	/manufacturers/{id}/domains/{domain}	DELETE	Smaže pro daného výrobce danou doménu.

Hlavním účelem je zobrazení výrobců, kteří jsou evidovaní v systému, a umožnit základní CRUD operace (create, read, update, delete).

2.2.2.1 Datový model

Datový model pro výrobce v novém systému musí vycházet z datového modelu, který byl představen v rámci analýzy výrobců (obrázek 1.4). Tento model bude pro nový systém rozšířen o nový sloupec `default_domain` v tabulce `oc_manufacturer`.

Současná administrace u výrobce eviduje seznam domén a neeviduje nic, co by bylo možné označit jako výchozí doménu. Pro nový systém se nově tento pojem zavede a jedna doména bude vždy výchozí doménou. Pro existující záznamy bude určena jako výchozí doména ta, která se nachází v seznamu domén (ve sloupci `domains`) jako první. Význam a použití výchozí domény bude vysvětleno později v jedné z následujících kapitol.

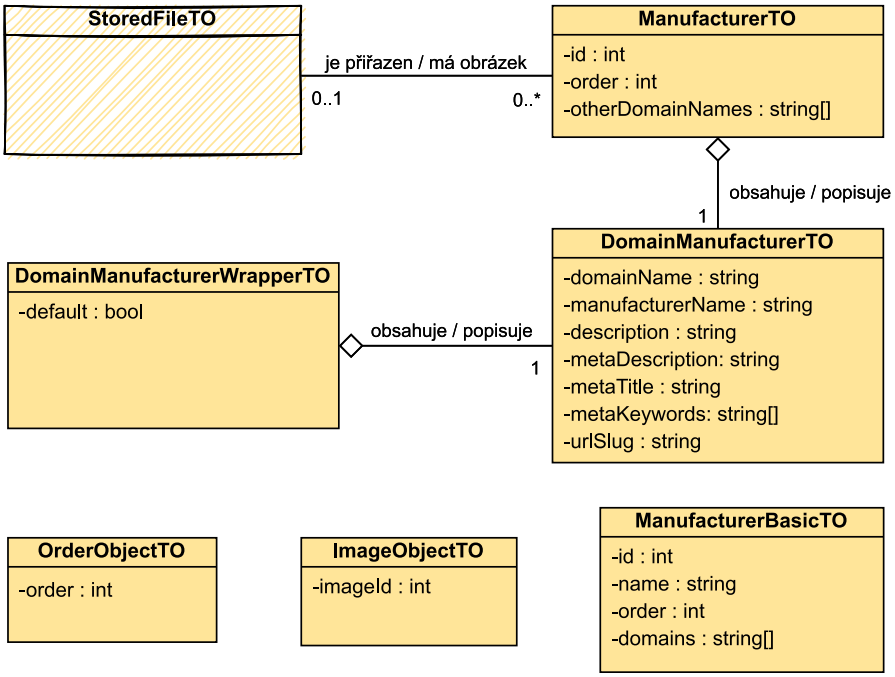
2.2.2.2 Návrh API

Administrace bude za účelem správy výrobců vystavovat API skládající se z několika přístupových bodů. Seznam přístupových bodů je i s popisem uveden v následující tabulce 2.1. Všechny uvedené API endpointy budou mít ještě prefix `/api`, který není v tabulce z důvodu čitelnosti uveden.

2.2.2.3 Doménový model

V této podkapitole nebude popsán klasický doménový model, ale model popisující transportní objekty. Každý transportní objekt bude mít definovanou metodu `buildFromDO` (s výjimkou ob-

Obrázek 2.2 Doménový model transportních objektů pro výrobce



jektů `OrderObjectTO` a `ImageObjectTO`), která bude sloužit pro vytvoření transportního objektu z objektu (či objektů) získaného z databáze. Tyto transportní objekty budou použity při serializaci a deserializaci dat u dříve definovaných API endpointů. Model pro výrobce je znázorněn na obrázku 2.2.

Některé entity v modelu jsou graficky odlišeny (vyšrafovány). Tyto objekty jsou součástí jiných částí systému a nejsou tedy předmětem aktuálně popisovaného návrhu. Jejich popis bude uveden později.

ManufacturerBasicTO

Tento objekt představuje zjednodušenou instanci objektu výrobce. Všechny atributy nabývají hodnot, které jsou definovány pro výchozí doménu.

Tabulka 2.2 Specifikace atributů třídy ManufacturerBasicTO

Název atributu	Definice atributu
id	Identifikátor výrobce
name	Jméno výrobce
order	Pořadové číslo výrobce
domains	Seznam domén, pro které je výrobce definovaný

ManufacturerTO

Představuje plnohodnotného výrobce, který kromě níže popsaných atributů obsahuje i výchozí doménu v podobě vazby na objekt `DomainManufacturerTO`. Dále má objekt výrobce volitelnou vazbu na obrázek, který je zastoupen objektem `StoredFileTO`. Obrázek zatím nemá definované

atributy, neboť jeho definice bude provedena později a zde je uveden pouze pro účely vyjádření vazby.

Tabulka 2.3 Specifikace atributů třídy ManufacturerTO

Název atributu	Definice atributu
id	Identifikátor výrobce
order	Pořadové číslo výrobce
otherDomainNames	Seznam dalších (nevýchozích) domén, pro které je výrobce definovaný

DomainManufacturerTO
Reprezentuje doménově specifické informace pro výrobce, kde hlavním atributem je `domainName`. Výrobce je vždy jen jeden, ale hodnoty v tomto objektu se mohou pro výrobce měnit v závislosti na doméně. Hodnoty, které začínají slovem *meta*, jsou určeny pro generování meta tagů stránky e-shopu[4].

Tabulka 2.4 Specifikace atributů třídy DomainManufacturerTO

Název atributu	Definice atributu
domainName	Název domény, pro kterou jsou platná data v tomto objektu
manufacturerName	Jméno výrobce
description	Popis výrobce
metaDescription	Meta popis výrobce
metaTitle	Meta název výrobce
metaKeywords	Meta klíčová slova
urlSlug	Nebo též URL alias (vysvětlení výrazu v dřívější kapitole 1.3.3)

DomainManufacturerWrapperTO
Tento objekt slouží pro obalení objektu `DomainManufacturerTO` a přidává informaci, zda se jedná o data pro výchozí doménu výrobce. Jelikož se jedná o velice jednoduchý objekt, který byl tímto již popsán, tak nebude uvedena tabulka se specifikací atributů.

StoredFileTO a ImageObjectTO
Tyto dva objekty spolu úzce souvisí. Návrh pro objekt `StoredFileTO` bude proveden později. Avšak je možné alespoň uvést, že každý soubor bude mít svůj identifikátor a tento identifikátor bude používat objekt `ImageObjectTO`. Objekt `ImageObjectTO` je obecný a bude později použit vícekrát. Jeho účelem je předat identifikátor obrázku, který se má pro určitého výrobce nastavit. Tabulka se specifikací atributů bude vynechána.

OrderObjectTO
Tento objekt slouží pro zadání změny pořadí výrobce. Jedná se o obecný objekt, který bude později opětovně použit. Tabulka se specifikací atributů bude opět vynechána.

[4] Více o HTML tagu <meta> například na https://www.w3schools.com/tags/tag_meta.asp.

Mapování DTO objektů na definované API endpointy
Pro lepší přehlednost bude nyní uvedena tabulka 2.5. V tabulce jsou zaznamenány vztahy mezi transportními objekty a jednotlivými API endpointy. Jinak řečeno, jaké hodnoty daný endpoint v optimálním případě vrací (výstup) a na které DTO se mapují vstupní data.

Tabulka 2.5 Použití DTO v jednotlivých API endpointech z tabulky 2.1.

Číslo endpointu	Výstup	Vstup mapován na DTO
1	ManufacturerBasicTO (kolekce)	
2	(ID nového výrobce)	ManufacturerTO
3	ManufacturerTO	
4		
5		ImageObjectTO
6		OrderObjectTO
7	(Jméno domény)	DomainManufacturerTO
8	DomainManufacturerWrapperTO	
9		DomainManufacturerWrapperTO
10		

2.2.3 Správa URL slug

V této kapitole bude představen návrh týkající se již dříve vysvětleného termínu URL slug (analýza, kapitola 1.3.3.3). V rámci požadavků, v kapitole 1.4.3.6, bylo uvedeno několik pravidel týkajících se záznamů URL slug. Tento návrh a následná implementace bude mít především podpůrnou funkci pro další části systému (výrobci, kategorie a další).

V rámci zmíněné podpůrné funkce je potřeba zajistit jednotný přístup při práci s URL slugs, kde bude zajištěna tvorba historie a zároveň budou dodržována všechna pravidla, která jsou určena ve funkčním požadavku.

2.2.3.1 Datový model

Z pohledu datového modelu se pro URL slug nic měnit nebude. URL slugs jsou uloženy v tabulce `oc_url_alias`. V této tabulce stojí za zmínění sloupec `query`, který definuje objekt, ke kterému se záznam vztahuje. Hlavní konstrukce, které se v tomto sloupci objevují, jsou:

- `category_id=?` (pro kategorie),
- `manufacturer_id=?` (pro výrobce),
- `product_id=?` (pro produkty),
- `information_id=?` (pro informace, což je část systému, která není v rámci této práce zahrnuta),

kde místo znaku `?` je vyplněn identifikátor propojeného záznamu.

2.2.3.2 Návrh API

Jak již bylo dříve zmíněno, tato funkcionalita je především podpůrná. To je možné pozorovat i na navrženém API, které v tomto případě obsahuje pouze jeden přístupový bod. Specifikace přístupového bodu i s popisem je uvedena v tabulce 2.6.

Seznam objektů místo jednoho objektu URL slug
Navržený přístupový bod bude vracet seznam objektů, které splňují požadavek na klíčové slovo. Popsaná podoba navrženého API endpointu není původní a byla dodatečně změněna. Původní návrh vracel vždy pouze jeden objekt. Pro pochopení, proč byly tyto změny provedeny, je nutné znát, jak se bude API endpoint používat, což bude níže demonstrováno.

1. V systému existuje záznam klíčového slova *slon* pro výrobce s identifikátorem *10* a doménu *X* (výchozí doména).
2. V systému existuje záznam klíčového slova *slon* pro výrobce s identifikátorem *10* a doménu *Y* (nevýchozí doména).
3. V systému existuje záznam klíčového slova *leopard* pro výrobce s identifikátorem *33* a doménu *Z1* (nevýchozí doména).
4. V systému existuje záznam klíčového slova *leopard* pro výrobce s identifikátorem *11* a doménu *Z2* (nevýchozí doména).
5. Klient se zeptá, zda existuje klíčové slovo *leopard* pro doménu T (nevýchozí doména).
6. Klient se zeptá, zda existuje klíčové slovo *leopard* pro doménu T (výchozí doména pro výrobce s identifikátore *33*).

Situace v prvním a druhém bodě je zcela validní, neboť se jedná o povolenou duplicitu. V současné administraci se tyto případy vyskytují. Daný výrobce má URL slug *slon* jako výchozí hodnotu (*X* je výchozí doménou). Tento výrobce je definován i pro doménu *Y*, která používá stejný URL slug. V systému (databázi) se jedná o zbytečnou duplicitu, která je tolerována.

Pro zodpovězení situace z pátého bodu se bude nyní předpokládat platnost bodů tři a čtyři. O doméně *T* není z pohledu serverové aplikace nic známo, ale z pohledu klientské aplikace se ví, že se jedná o nevýchozí doménu. V tomto případě nezáleží, jestli serverová aplikace vrátí instanci URL slug z bodu tři nebo čtyři.

Pro zodpovězení situace z šestého bodu se bude opět předpokládat platnost bodů tři a čtyři. O doméně *T* není z pohledu serverové aplikace nic známo, ale z pohledu klientské aplikace se ví, že se jedná o výchozí doménu pro zmíněného výrobce. V tomto případě mohou nastat dvě situace. První je, že serverová aplikace vrátí instanci URL slug z bodu čtyři. Jelikož se liší identifikátory výrobců, tak klientská aplikace může poznat, že dané klíčové slovo nelze použít jako výchozí pro všechny domény (doména *Z2* by měla definované jedno klíčové slovo pro dva výrobce). Druhou možností je, že serverová aplikace vrátí instanci URL slug z bodu tři. V tomto případě se jedná o již zmíněnou tolerovanou duplicitu a klientská aplikace nemůže poznat, že existuje ještě další záznam, který použití vylučuje.

Serverová aplikace tedy nemá všechny potřebné informace, aby mohla rozhodnout, kterou instanci URL slug je potřeba na dotaz vrátit. Návrh byl proto upraven a zmíněný přístupový bod bude vracet seznam instancí URL slug.

Povinný query parametr v URL
Přístupový bod neobsahoval v původním návrhu povinný query parametr `default`. Tento parametr byl přidán z optimalizačních důvodů. Pokud je známo, že hledané klíčové slovo nebude použito jako výchozí, tak je možné vyhledávání (shody na klíčové slovo) omezit. Konkrétně se hledá shoda klíčového slova jen pro danou doménu nebo URL slug definovaný pro všechny domény.

Tabulka 2.6 Specifikace API administrace pro výrobce

URI	Metoda	Popis
/url-slugs/{keyword}/domains/{domain}	GET	Pokusí se najít podle klíčového slova a domény odpovídající URL slug. Endpoint má povinný query parametr `default`, který nabývá hodnot `true` nebo `false`.

Naopak pokud bude klíčové slovo použito jako výchozí, tak je nutné zkontrolovat unikátnost se všemi záznamy.

2.2.3.3 Doménový model

V této kapitole bude představen pouze jeden transportní objekt, `UrlSlugTO`. Více objektů se v rámci této funkcionality nevyskytuje, neboť hlavním účelem je poskytnout informaci o existenci URL slug a poskytnout podpůrnou funkci pro jiné části systému (například výrobce).

Tabulka 2.7 Specifikace atributů třídy UrlSlugTO

Název atributu	Definice atributu
id	Identifikátor URL slug
domain	Název domény
keyword	Klíčové slovo
active	Určuje, zda je aktivní (pokud ne, jedná se o historický záznam)
refObjectName	Označení referenčního objektu (získáno z `originalQuery`)
refObjectId	Identifikátor referenčního objektu (získáno z `originalQuery`)
originalQuery	Označení zdroje, který daný URL slug používá (v textu označováno zkráceně jako `query`)

Atribut `refObjectName` může nabývat pouze omezeného množství hodnot, kterými jsou:

- category,
- manufacturer,
- product,
- information,
- other.

2.2.4 Kategorie

V této podkapitole bude představen návrh části systému, který bude pracovat s kategoriemi, parametry, filtry kategorií a bannery. Pro všechna vyjmenovaná témata je potřeba pokrýt dříve definované funkční požadavky.

2.2.4.1 Datový model

Datový model pro kategorie opět musí vycházet z datového modelu, který byl představen v rámci analýzy kategorií (obrázek 1.5). Tento model bude pro nový systém opět rozšířen o sloupec `default_domain` v tabulce `oc_category`.

Současná administrace u kategorie eviduje seznam domén a neeviduje žádnou hodnotu, která by významově odpovídala výchozí doméně. Pro nový systém bude tento pojem zaveden a jedna doména bude vždy výchozí. Podobně jako u výrobců se pro existující záznamy určí jako výchozí ta doména, která se nachází v seznamu domén (ve sloupci `domains`) jako první.

Skupiny domén

V průběhu implementace bylo zjištěno, že u kategorií platí nepsané pravidlo. Existují skupiny domén, které se vždy dané kategorii nastaví (v databázi odpovídá sloupci `domains` v tabulce `oc_category`). Toto pravidlo je podstatné kvůli hierarchii domén, protože provázané kategorie musí mít stejnou množinu domén. Kdyby toto neplatilo, tak by se pro určitou doménu mohlo stát, že kategorie bude mít předka, který není definován pro danou doménu. V takovém případě by daná kategorie byla pro danou doménu (e-shop) v podstatě neviditelná. Současná administrace toto pravidlo nehlídá a je pouze na uživateli, zda jej dodrží.

Zmíněný problém má ještě jeden důsledek. Pokud skupiny domén nejsou dodrženy, tak je mnohem náročnější identifikovat případné cykly v kategoriích (pro jednu doménu a danou množinu kategorií smyčka existovat nemusí a pro jinou doménu ano). Opět platí, že současná administrace cykly nehlídá a v případě vzniku cyklické reference nastanou potíže s načítáním e-shopu.

Poté, co byl tento problém identifikován, bylo dohodnuto, že bude návrh pro novou administraci upraven a přidá se definice skupin domén. Pro skupiny domén bude vytvořena nová tabulka `oc_domain_group`. Zároveň bude přidána tabulka `oc_domain`, která bude obsahovat všechny domény a mapovat je do skupin domén. Nová administrace pak musí kontrolovat při úpravách kategorií, zda jsou v seznamu domén uvedeny všechny domény z dané kategorie. Zároveň díky tomuto kroku bude možné implementovat algoritmus pro detekci cyklů.

2.2.4.2 Návrh API

Jelikož podkapitola Kategorie je rozsáhlejší a zahrnuje kromě obecné logiky kategorií i parametry, filtry a bannery, tak i seznam API přístupových bodů bude delší. Tento seznam je i s popisem uveden v následující tabulce 2.8. Všechny uvedené API endpointy budou mít ještě prefix `/api`, který není v tabulce z důvodu čitelnosti uveden.

V seznamu API endpointů není uveden žádný přístupový bod pro přidání domény dané kategorii nebo pro smazání domény dané kategorie, čímž se návrh týkající se čistě jen kategorií liší od návrhu pro výrobce. Tento rozdíl je zapříčiněn dříve popsaným problémem se skupinou domén.

Tabulka 2.8 Specifikace API administrace pro kategorie, parametry, filtry a bannery

#	URI	Metoda	Popis
1	/categories	GET	Vrátí seznam všech kategorií.
2	/categories	POST	Vytvoří novou kategorii.
3	/categories/{id}	GET	Vrátí informace o kategorii.
4	/categories/{id}	DELETE	Smaže danou kategorii.
5	/categories/{id}/parent-category	PUT	Nastaví dané kategorii zadanou kategorii jako rodičovskou.
6	/categories/{id}/order	PUT	Nastaví dané kategorii zadané pořadí a přepočítá pořadí ostatních kategorií se stejnou rodičovskou kategorií.
7	/categories/{id}/domains/{domain}	GET	Vrátí informace o kategorii pro danou doménu.
8	/categories/{id}/domains/{domain}	PUT	Nastaví dané kategorii pro danou doménu zadaná data.
9	/categories/parameters	GET	Vrátí seznam všech dostupných parametrů.
10	/categories/{id}/parameters	GET	Vrátí seznam všech parametrů pro danou kategorii.
11	/categories/{id}/parameters	PUT	Přijme seznam parametrů pro danou kategorii a podle toho upraví parametry pro danou kategorii (přidá, smaže, aktualizuje).
12	/categories/{id}/filters	GET	Vrátí seznam všech filtrů pro danou kategorii.
13	/categories/{id}/filters	PUT	Přijme seznam filtrů pro danou kategorii a podle toho upraví filtry pro danou kategorii (přidá, smaže, aktualizuje).
14	/categories/{id}/banners/domains	GET	Vrátí seznam domén mající pro danou kategorii definovaný nějaký banner.
15	/categories/{id}/banners/domains	POST	Vytvoří pro danou kategorii předaný seznam bannerů.
16	/categories/{id}/banners/domains/{domain}	GET	Vrátí seznam bannerů pro danou kategorii a doménu.
17	/categories/{id}/banners/domains/{domain}	PUT	Aktualizuje podle předaných dat bannery pro danou kategorii a doménu.
18	/categories/{id}/banners/domains/{domain}	DELETE	Smaže pro danou kategorii a doménu všechny existující bannery.

2.2.4.3 Doménový model

V této podkapitole nebude představen klasický doménový model, ale model popisující transportní objekty. Každý transportní objekt bude mít metodu `buildFromDO` (s výjimkou objektů `OrderObjectTO`, `ImageObjectTO` a `ParentObjectTO`), která bude sloužit pro sestavení transportního objektu z objektu (nebo více objektů) načteného z databáze. Tyto transportní objekty budou opět použity při serializaci a deserializaci dat u dříve definovaných přístupových bodů. Model pro kategorie je na obrázku 2.3.

Některé entity v modelu jsou graficky odlišeny (vyšrafovány). Tyto objekty jsou součástí jiných částí systému a nejsou tedy předmětem aktuálně popisovaného návrhu. Jejich popis bude uveden později.

CategoryBasicTO
Tento objekt představuje zjednodušenou instanci objektu kategorie. Všechny atributy nabývají hodnot, které jsou definovány pro výchozí doménu.

Tabulka 2.9 Specifikace atributů třídy CategoryBasicTO

Název atributu	Definice atributu
id	Identifikátor kategorie
name	Jméno kategorie
order	Pořadové číslo kategorie (v rámci kategorií se stejným rodičem)
parentCategoryId	Identifikátor rodičovské kategorie
domains	Seznam domén, pro které je kategorie definovaná
domainGroupId	Skupina domén, do které je kategorie zařazena

CategoryTO
Představuje kompletní kategorii, která obsahuje níže popsané atributy. Tento objekt obsahuje hodnoty výchozí domény pomocí vazby na instanci objektu `DomainCategoryTO`.

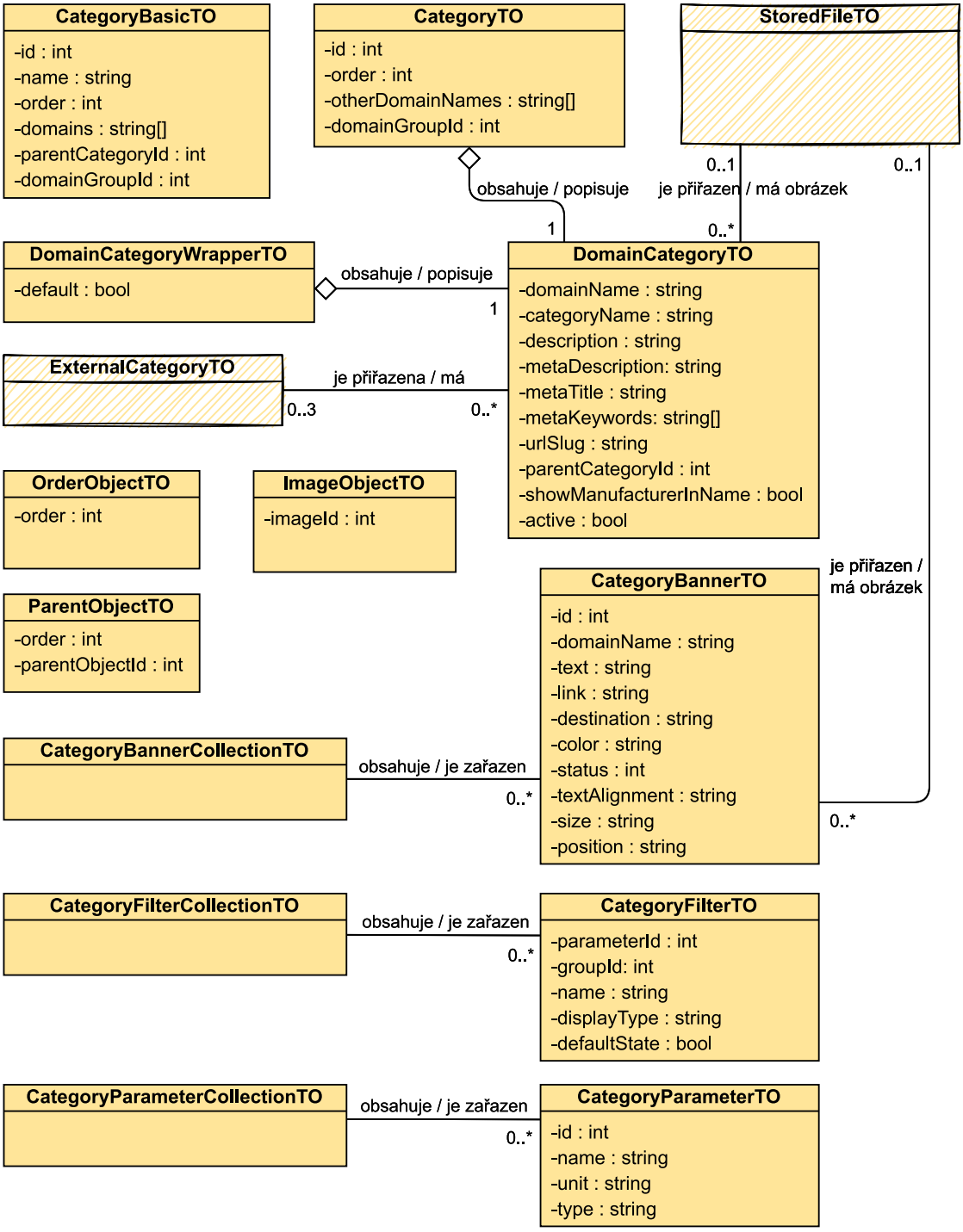
Tabulka 2.10 Specifikace atributů třídy CategoryTO

Název atributu	Definice atributu
id	Identifikátor kategorie
order	Pořadové číslo kategorie
otherDomainNames	Seznam dalších (nevýchozích) domén, pro které je kateg. definovaná
domainGroupId	Skupina domén, do které je kategorie zařazena

DomainCategoryTO
Reprezentuje doménově specifické informace o kategorii. Jméno domény je obsaženo v atributu `domainName`. Podobně jako u výrobce platí, že kategorie je pouze jedna, ale hodnoty jejích atributů se mohou měnit v závislosti na doméně. Hodnoty, které začínají slovem *meta*, jsou určeny pro generování meta tagů stránky e-shopu[5].

Zde je nutné upozornit, že pokud je atribut `parentCategoryId` změněn, tak jeho změna má vždy vliv na všechny domény, pro které je kategorie definovaná. Tento fakt vyplývá ze zjištění z vývoje a následné úpravy návrhu (skupiny domén).

Obrázek 2.3 Doménový model transportních objektů pro kategorie



Dále může být tento objekt propojen až se třemi externími kategoriemi. Maximálně právě se třemi, protože jsou definovány pouze tři možné zdroje - Heureka kategorie, Zbozi.cz kategorie, Google kategorie. Externí kategorie je v modelu definována pomocí vazby na objekt `ExternalCategoryTO`. Externí kategorie zatím nemá definované atributy, neboť její definice bude provedena později a zde je uvedena pouze pro účely vyjádření vazby (proto je v modelu graficky odlišena).

Na rozdíl od výrobců je možné u kategorií definovat obrázek v závislosti na doméně. Tato vazba je znázorněna v modelu a obrázek je zastoupen objektem `StoredFileTO`. Tento objekt nemá definované atributy, protože jeho definice bude uvedena později.

Tabulka 2.11 Specifikace atributů třídy DomainCategoryTO

Název atributu	Definice atributu
domainName	Název domény, pro kterou jsou platná data v tomto objektu
categoryName	Jméno kategorie
description	Popis kategorie
metaDescription	Meta popis kategorie
metaTitle	Meta název kategorie
metaKeywords	Meta klíčová slova pro kategorii
urlSlug	Nebo též URL alias (vysvětlení výrazu v dřívější kapitole 1.3.3)
parentCategoryId	Identifikátor rodičovské kategorie
showManufacturerInName	Zobrazovat výrobce v názvech
active	Zda je kategorie aktivní

DomainCategoryWrapperTO
Tento objekt slouží pro obalení objektu `DomainCategoryTO` a přidává informaci, zda se jedná o data pro výchozí doménu kategorie. Jelikož se jedná o velice jednoduchý objekt, který byl tímto již popsán, tak nebude uvedena tabulka se specifikací atributů.

StoredFileTO a ImageObjectTO
Tyto dva objekty spolu úzce souvisí. Návrh pro objekt `StoredFileTO` bude proveden později. Avšak je možné alespoň uvést, že každý soubor bude mít svůj identifikátor a tento identifikátor bude používat objekt `ImageObjectTO`. Objekt `ImageObjectTO` je obecný a již byl použit při návrhu výrobce. Jeho účelem je předat identifikátor obrázku, který se má nastavit. Tabulka se specifikací atributů bude vynechána.

OrderObjectTO
Tento objekt slouží pro zadání změny pořadí kategorie. Tabulka se specifikací atributů bude opět vynechána.

ParentObjectId
Tento objekt slouží pro zadání změny rodičovské kategorie. Množiny kategorií tvoří stromy kategorií a kategorie v daném stromu jsou vždy řazeny hierarchicky a podle pořadového čísla se řadí pouze kategorie se stejným rodičem. Při změně rodičovské kategorie je proto nutné změnit i

[5] Více o HTML tagu <meta> například na https://www.w3schools.com/tags/tag_meta.asp.

pořadové číslo. Tabulka se specifikací atributů bude vynechána, protože oba atributy byly nyní popsány.

CategoryBannerCollectionTO a CategoryBannerTO

Objekt `CategoryBannerTO` reprezentuje reklamní banner pro určitou doménu a kategorii (kategorie není v objektu přímo uvedena, protože kategorie bude vždy určena API endpointem). Opět se zde objevuje vazba na obrázek, který je reprezentován objektem `StoredFileTO`, jehož definice bude uvedena v jedné z pozdějších kapitol (proto je v modelu graficky odlišen).

Objekt `CategoryBannerCollectionTO` představuje kolekci reklamních bannerů.

Tabulka 2.12 Specifikace atributů třídy CategoryBannerTO

Název atributu	Definice atributu
id	Identifikátor banneru
domainName	Jméno domény
text	Text pro banner
link	Text odkazu
destination	Odkaz, který se otevře při kliknutí na banner
color	Barva banneru
status	Stav banneru
textAlignment	Zarovnání textu na banneru
size	Velikost banneru
position	Umístění banneru

Atribut `color` může nabývat pouze dvou hodnot:

- light - světlý banner,
- dark - tmavý banner.

Atribut `status` může nabývat tří hodnot:

- 0 - znamená vypnuto,
- 1 - znamená zapnuto,
- 2 - znamená test.

Atribut `textAlignment` může nabývat pouze dvou hodnot:

- left - text zarovnán doleva,
- right - text zarovnán doprava.

Atribut `size` může nabývat pouze dvou hodnot:

- smallsh - malý banner,
- *prázdný textový řetězec* - velký banner.

Posledním atributem je `posititon`, který může nabývat dvou hodnot:

- topleft - umístění banneru vlevo,
- topright - umístění banneru vpravo.

CategoryParameterCollectionTO a CategoryParameterTO

Objekt `CategoryParameterTO` reprezentuje parametr. Tento objekt se bude při voláních API endpointů vyskytovat v kolekci `CategoryParameterCollectionTO`. Parametr má atribut `type`, který nabývá pouze omezeného množství hodnot. Hodnoty pro tento atribut byly již dříve určeny mezi požadavky v kapitole 1.4.3.3.

Tabulka 2.13 Specifikace atributů třídy CategoryParameterTO

Název atributu	Definice atributu
id	Identifikátor parametru
name	Jméno parametru
unit	Označení jednotek parametru
type	Typ parametru

CategoryFilterCollectionTO a CategoryFilterTO

Objekt `CategoryFilterTO` představuje filtr v rámci dané skupiny kategorií. Filtr má dále vždy vazbu na parametr a omezení na toto spojení jsou specifikována v požadavcích v kapitole 1.4.3.4. Typ filtru je definován atributem `displayType`, který nabývá pouze omezeného množství hodnot. Tyto hodnoty jsou taktéž uvedeny v požadavcích v již zmíněné kapitole.

Objekt `CategoryFilterCollectionTO` představuje kolekci filtrů.

Tabulka 2.14 Specifikace atributů třídy CategoryFilterTO

Název atributu	Definice atributu
parameterId	Id parametru, na který se filtr váže (též označováno jako `featureId`)
groupId	Identifikátor skupiny
name	Název filtru
displayType	Typ filtru
defaultState	Zda je filtr ve výchozím stavu rozbalený či sbalený

Mapování DTO objektů na definované API endpointy

Pro lepší přehlednost bude nyní uvedena tabulka 2.15. V tabulce jsou zaznamenány vztahy mezi transportními objekty a jednotlivými API endpointy. Jinak řečeno, jaké hodnoty daný endpoint v optimálním případě vrací (výstup) a na které DTO se mapují vstupní data.

Tabulka 2.15 Použití DTO v jednotlivých API endpointech z tabulky 2.8.

Číslo endpointu	Výstup	Vstup mapován na DTO
1	CategoryBasicTO (kolekce)	
2	(ID nové kategorie)	CategoryTO
3	CategoryTO	
4		
5		ParentObjectTO
6		OrderObjectTO
7	DomainManufacturerWrapperTO	
8		DomainManufacturerWrapperTO
9	CategoryParameterCollectionTO	
10	CategoryParameterCollectionTO	
11		CategoryParameterCollectionTO
12	CategoryFilterCollectionTO	
13		CategoryFilterCollectionTO
14	CategoryBannerCollectionTO	
15	(Id vytvořených bannerů)	CategoryFilterCollectionTO
16	CategoryFilterCollectionTO	
17		CategoryFilterCollectionTO
18		

2.2.5 Správce obrázků a souborů

V této podkapitole bude představen návrh pro správu souborů, především obrázků. Požadavek na vytvoření správce souborů byl definován v podkapitole 1.4.3.7. Cílem je poskytnout jednotný přístup pro práci se soubory a poskytnout standardní sadu funkcí pro práci se soubory a složkami.

2.2.5.1 Datový model

Současná administrace definuje všechny soubory a složky pomocí systémové cesty k danému objektu. Tento způsob ztěžuje přejmenování či přesunutí souboru, protože je nutné vědět, kde všude je cesta definována a ručně cestu opravit (systém toto nedělá automaticky). Pro novou administraci bude založena nová tabulka `oc_file_folder`, kde budou registrovány všechny složky a soubory.

Každý soubor a složka bude mít svůj identifikátor, což umožní lepší komunikaci s klientskou aplikací. Zároveň bude možné do budoucna odstranit systémové cesty z ostatních tabulek (vše bude na jednom místě v jedné tabulce). Díky tomuto kroku bude v budoucnu jednoduché přejmenování či přesunutí souboru, protože identifikátor se nezmění a cesta bude upravena pouze na jednom místě.

Aktuálně není možné odstranit sloupce obsahující systémové cesty z ostatních tabulek a nahradit je sloupci s identifikátory. Tento krok by rozbil současnou administraci. Pro novou ad-

ministraci proto bude definována nová tabulka a nově zavedené identifikátory se budou používat jen pro komunikaci skrze API.

Nová tabulka bude obsahovat následující seznam sloupců:

id (int) označuje identifikátor souboru či složky,

parent_id (int) označuje identifikátor složky, do které daný soubor/složka patří,

name (varchar) definuje jméno složky/souboru,

directory (tinyint) určuje, zda se jedná o složku nebo soubor.

2.2.5.2 Návrh API

Administrace bude pro účely správy souborů (jmenovitě především obrázků) poskytovat API rozhraní. Soubory bude možné nahrát, mazat, přejmenovat, přesunout. Bude umožněno pracovat také se složkami a provádět s nimi podobné operace jako se soubory. Seznam přístupových bodů je i s popisem uveden v tabulce 2.16. Všechny uvedené API endpointy budou mít ještě prefix `/api`, který není v tabulce z důvodu čitelnosti uveden.

Jak již bylo uvedeno dříve, každý soubor bude mít svůj identifikátor. Této skutečnosti bude využito v API. Pokud uživatel například u výrobce nastaví určitý obrázek (nahraje nový obrázek), tak se nejprve zavolá API endpoint pro nahrání obrázků, čímž obrázek dostane i svůj identifikátor. Následně při ukládání upraveného výrobce se na serverovou aplikaci pošle pouze identifikátor obrázku (namísto systémové cesty).

V seznamu API endpointů 2.16 je speciální význam kladen především na první přístupový bod, u kterého lze předpokládat, že bude vždy volán jako první. V rámci tohoto přístupového bodu systém provede analýzu úložného prostoru a pokud se v něm nachází složka či soubor, které nejsou uvedeny v databázi, vytvoří se nový záznam. Tento bod je důležitý nejen pro první spuštění aplikace, aby se správně inicializovaly záznamy v databázi, ale také pro pozdější paralelní fungování se současnou administrací (ta totiž novou tabulku pro soubory nebude používat).

Pro všechny přístupové body by mělo platit následující pravidlo. Pokud je záznam v databázi, ale neexistuje na filesystému daná složka (či soubor), záznam se z databáze nemaže, protože by tato operace byla potencionálně nebezpečná. Zároveň však systém bude danou složku (soubor) ignorovat, jako kdyby její záznam v databázi neexistoval. Pokud by byla složka (či soubor) později doplněna, začne se zobrazovat a není nutný žádný zásah do databáze.

2.2.5.3 Doménový model

V této podkapitole opět nebude představen klasický doménový model, ale model popisující transportní objekty. V tomto případě se jedná o velice jednoduchý model, kde nejsou žádné vztahy mezi objekty. Tyto objekty jsou však součástí jiných, již dříve uvedených, částí návrhu. Níže uvedené objekty `StoredFileTO` a `FolderTO` budou mít opět metodu `buildFromDO`. Model je zachycen na obrázku 2.4.

FolderTO

Tento objekt představuje již mnohokrát zmiňovanou složku. Základní charakteristikou složky v tomto případě je, že odkazuje na další složky, které jsou definovány uvnitř ní.

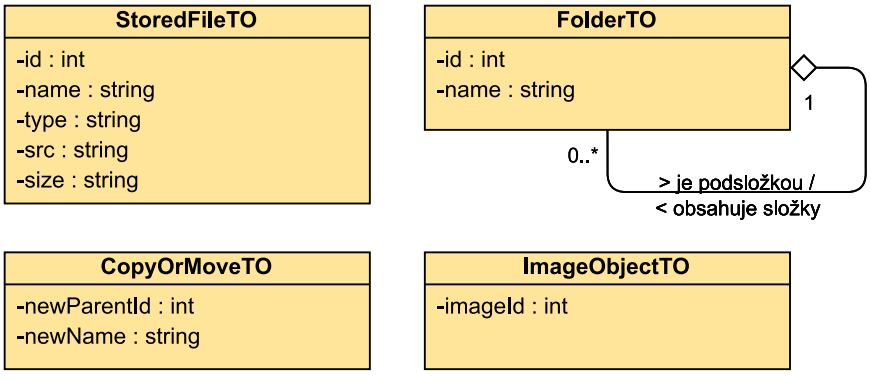
Tabulka 2.17 Specifikace atributů třídy FolderTO

Název atributu	Definice atributu
id	Identifikátor složky
name	Jméno složky

Tabulka 2.16 Specifikace API administrace pro správce obrázků a souborů

#	URI	Metoda	Popis
1	/folders	GET	Vrátí hierarchický seznam všech složek.
2	/folders/{folderId}	POST	Uvnitř dané složky vytvoří novou složku.
3	/folders/{folderId}	DELETE	Smaže danou složku se všemi jejími podsložkami a soubory.
4	/folders/{folderId}/copy	POST	Zkopíruje složku se všemi jejími podložkami a soubory do zadané lokace.
5	/folders/{folderId}/move	POST	Přesune celou složku do nové lokace.
6	/folders/{folderId}/files	GET	Vrátí seznam všech souborů v dané složce.
7	/folders/{folderId}/files	POST	Slouží pro nahrání souborů, které mají být uloženy do zadané složky.
8	/files/{fileId}	GET	Vrátí informace o daném souboru.
9	/files/{fileId}	DELETE	Smaže daný soubor.
10	/files/{fileId}/copy	POST	Zkopíruje daný soubor do zadané lokace.
11	/files/{fileId}/move	POST	Přesune daný soubor do zadané lokace.

Obrázek 2.4 Doménový model transportních objektů pro správce souborů a obrázků



StoredFileTO
Tento objekt představuje soubor, který se vyskytuje na úložišti a v databázi má přidělený identifikátor.

Tabulka 2.18 Specifikace atributů třídy StoredFileTO

Název atributu	Definice atributu
id	Identifikátor souboru
name	Jméno souboru
type	Typ souboru (obrázek, PDF soubor, ostatní)
src	Cesta k souboru
size	Velikost souboru v čitelné podobě pro člověka

Atributy `type`, `src`, `size` nejsou uloženy v databázi. Proto se tyto atributy musí vyplňovat dynamicky na základě uloženého souboru. Cesta k souboru se určí podle nastaveného rodiče (složky) souboru a všech rodičů dané složky. Typ je určen na základě koncovky souboru a platí tato následující pravidla:

- PDF soubor – soubor má koncovka *pdf*,
- obrázek – soubor má jednu z koncovek *jpg*, *jpeg*, *png*, *gif*, *bmp*,
- ostatní.

Posledním atributem je velikost souboru. Ta je získána v bytech a následně je nutné ji přepočítat tak, aby byla pro lidského uživatele jednoduše srozumitelná. Podporované velikosti jsou B (velikost souboru je menší než 1024 B), kB (velikost souboru je menší než 1024 kB) a MB (ostatní).

ImageObjectTO
Tento objekt byl představen již dříve. Jedná se o obecný objekt, jehož účelem je předat identifikátor obrázku. Zde je tento objekt připomenut pouze proto, že je nositelem identifikátoru objektů, které jsou zásadní pro návrh této funkcionality. Tabulka se specifikací atributů bude vynechána.

CopyOrMoveTO
Toto je poslední objekt, který je pro správu souborů navrhnut. Jeho účely jsou pouze transportní a jak název napovídá, bude se používat pro kopírování a přesouvání složky (či souboru). Objekt v sobě nese informace o nové lokaci a novém jménu. Tabulka se specifikací atributů bude vynechána.

Mapování DTO objektů na definované API endpointy
Pro lepší přehlednost bude nyní uvedena tabulka 2.19. V tabulce jsou zaznamenány vztahy mezi transportními objekty a jednotlivými API endpointy. Jinak řečeno, jaké hodnoty daný endpoint v optimálním případě vrací (výstup) a na které DTO se mapují vstupní data.

Tabulka 2.19 Použití DTO v jednotlivých API endpointech z tabulky 2.16.

Číslo endpointu	Výstup	Vstup mapován na DTO
1	FolderTO	
2	(ID nové složky)	FolderTO (bez podsložek)
3		
4	(ID nové složky)	CopyOrMoveTO
5	(ID přesunuté složky)	CopyOrMoveTO
6	seznam StoredFileTO	
7	(ID nového souboru)	(Soubor)
8	StoredFileTO	
9		
10	(ID nového souboru)	CopyOrMoveTO
11	(ID přesunutého souboru)	CopyOrMoveTO

První přístupový bod je mapován pouze na jeden objekt `FolderTO`, protože pouze jedna složka bude kořenová a ostatní složky jsou již v ní jako její podsložky.

2.2.6 Externí kategorie

V této podkapitole bude popsán návrh pro externí kategorie, které byly zmíněny v analýze. Načež také vznikl požadavek na vytvoření funkcionality týkající se externích kategorií (v kapitole 1.4.3.8). Cílem je vytvořit automatizované řešení, které bude automaticky kontrolovat externí kategorie a v případě potřeby provede jejich aktualizaci. Zároveň je potřeba vystavit API endpoint pro získání seznamu externích kategorií, které jsou aktuálně registrovány v systému.

Externí kategorie mají tři možné zdroje a v době psaní této práce pro každou z nich funguje URL adresa, kde je možné kompletní seznam kategorií stáhnout.

- Heureka kategorie – https://www.heureka.cz/direct/xml-export/shops/heureka-sekce.xml
- Zbozi.cz kategorie – https://www.zbozi.cz/static/categories.csv
- Google kategorie – https://www.google.com/basepages/producttype/taxonomy-with-ids.cs-CZ.txt

2.2.6.1 Datový model

Stažené externí kategorie se budou ukládat do nové tabulky `oc_external_category`. Tato tabulka bude mít následující sloupce:

id (int) označuje identifikátor externí kategorie,

external_source_id (int) označuje identifikátor dané kategorie přidělený externím zdrojem,

name (varchar) definuje jméno kategorie,

full path (varchar) definuje celé jméno kategorie (obsahuje názvy nadřazených kategorií),

source (varchar) označuje externí zdroj.

Tabulka 2.20 Specifikace API administrace pro externí kategorie

#	URI	Metoda	Popis
1	/external-sources	PUT	Vyvolá přenačtení externích kategorií z oficiálních zdrojů.
2	/external-sources/{source}	GET	Vrátí seznam všech externích kategorií pro daný zdroj.

2.2.6.2 Návrh API

Nový systém bude poskytovat jednoduché API pro práci s externími kategoriemi. Bude umožněno manuální i automatické spuštění aktualizací externích kategorií a získání seznamu externích kategorií. Seznam přístupových bodů je i s popisem uveden v tabulce 2.20. Všechny uvedené API endpointy budou mít ještě prefix `/api`, který není v tabulce z důvodu čitelnosti uveden.

Samotná aktualizace externích kategorií bude pro každý zdroj probíhat obdobným způsobem.

1. Stažení kategorií od externího zdroje.
2. Zpracování stažených dat a převod kategorií do interních objektů.
3. V nově vzniklé tabulce `oc_external_category` se nalezne identifikátor externí kategorie (identifikátory se nemění na rozdíl od jména kategorie).
   a. Pokud není podle identifikátoru žádný záznam nalezen, tak se jedná o novou kategorii, která se pouze uloží a pokračuje se zpracováním další kategorie.
   b. Pokud je podle identifikátoru nalezen v tabulce záznam, tak externí kategorie v systému již existuje. Externí kategorie musí být ještě dodatečně zpracována (viz další body).
4. Podle starého celého názvu externí kategorie se v tabulkách nahradí záznam za nový celý název externí kategorie.
5. Aktualizuje se záznam o externí kategorii v tabulce `oc_external_category`.

Výše uvedený systém bohužel neodhalí některé situace. Například pokud je externí kategorie zcela smazána, tak zůstane původní název externí kategorie v příslušných záznamech. Dále aktuální databáze již obsahuje externí kategorie, které byly dříve přejmenovány či dokonce smazány. Tyto záznamy také není možné aktualizovat, protože před vznikem tabulky `oc_external_category` neexistovalo propojení mezi starým a novým pojmenováním externí kategorie. Jinak řečeno, není možné propojit staré pojmenování externí kategorie s tím novým, a proto nelze tyto záznamy automatizovaně opravit.

Doporučeným postupem do budoucna by určitě mělo být odstranění jmen externích kategorií ze všech tabulek kromě tabulky `oc_external_category`. Ve všech tabulkách by se místo jména externí kategorie měl používat identifikátor odkazující do tabulky `oc_external_category`. Následně bude jednoduché aktualizovat externí kategorie, protože všechny změny budou pouze v jedné tabulce. Tento krok aktuálně není možné provést, neboť by nebyla možná kompatibilita se současnou administrací (obě administrace by si navzájem škodily).

Podobně jako u správce souborů v kapitole 2.2.5 i zde budou identifikátory externích kategorií použity pro komunikaci s klientskou aplikací. V budoucnosti by se pak celá jména kategorií odstranila z ostatních tabulek a nahradily by je identifikátory. Aktuálně se v tabulkách používají celá jména externích kategorií, což komplikuje jejich aktualizace. Kvůli kompatibilitě se současnou administrací však zatím tuto závislost nelze odstranit a nový systém proto musí identifikátory překládat na cesty a ty ukládat do příslušných tabulek.

2.2.6.3 Doménový model

V této kapitole bude představen pouze jeden transportní objekt a to `ExternalCategoryTO`. Více objektů se v rámci této funkcionality nevyskytuje, neboť hlavním účelem je poskytnout externí kategorie (zajišťuje právě zmíněný objekt). Aby bylo možné externí kategorie takto poskytnout, tak je nutné implementovat funkcionalitu, která stáhne seznam kategorií z dříve zmíněných URL adres. Následně bude nutné data zpracovat a uložit do databáze.

Tabulka 2.21 Specifikace atributů třídy ExternalCategoryTO

Název atributu	Definice atributu
externalId	Identifikátor převzatý od externího zdroje
name	Jméno externí kategorie
fullPath	Celé jméno externí kategorie
source	Označení externího zdroje

Atribut `fullPath` je seznam textových řetězců. Jedná se o pole nadřazených kategorií. Původní návrh počítal s jedním textovým řetězcem, ale Martin Dvořák, týmový kolega podílející se na vývoji klientské aplikace, přišel s návrhem, aby se nadřazené kategorie vracely jako pole. Díky této změně může klientská aplikace zvolit libovolný oddělovač pro nadřazené kategorie.

2.2.7 Objednávky

V této podkapitole bude představen návrh části systému, který bude pracovat s objednávkami. Na základě funkčních požadavků je nutné, aby tato část systému byla navržena pro práci s objednávkami, fakturami a dobropisy.

2.2.7.1 Datový model

Datový model pro objednávky bude opět vycházet z datového modelu současného systému. Databáze současného systému pro objednávky byla představena v dřívější analýze (obrázek 1.6 zachycující datový model). Tento model bude pro nový systém rozšířen o dvě tabulky a jeden sloupec v tabulce `oc_order_history`.

První tabulkou, která bude přidána, je tabulka možných způsobů dopravy zboží (Česká pošta, Balíkovna a mnoho dalších). Jméno tabulky nechť je `oc_shipping_method`. Tato tabulka bude mít pouze identifikátor a název přepravní služby. Druhou tabulkou, která bude přidána, je tabulka dostupných platebních metod. Jméno tabulky nechť je `oc_payment_method`. Tato tabulka bude mít opět identifikátor a název platební metody.

Současná administrace všechny informace o způsobech dopravy a platebních metodách ukládá v tabulkách `oc_setting` a `oc_domain_setup` (spolu s dalšími možnostmi nastavení). Tento způsob zápisu není příliš čitelný při pohledu do databáze, protože je nutné znát klíče/schéma, kterými jsou tyto hodnoty v tabulkách zakódovány. Do budoucna by mohlo být výhodné využít nově vytvořené tabulky a přidat k nim třetí, která by obsahovala klíče obou nových tabulek a doménu. Touto trojicí by byl pro každou doménu jednoznačně definován seznam možných platebních a doručovacích metod. Zároveň by tato tabulka mohla obsahovat i ceny, dostupnost (zda je daná metoda aktivní) a další hodnoty.

V neposlední řadě bude do již existující tabulky `oc_order_history` přidán nový sloupec `created_by_system`. Tato tabulka nese informace o tom, v jakém stavu se objednávka postupně nacházela. Nový sloupec bude informovat o tom, zda byla daná změna v objednávce provedena automaticky systémem, nebo ji zadal obchodník.

Tabulka 2.22 Specifikace API administrace pro objednávky a dobropisy

#	URI	Metoda	Popis
1	/orders	GET	Vrátí seznam všech objednávek (bez dobropisů).
2	/orders/{orderId}	GET	Vrátí informace o dané objednávce (nebo dobropisu).
3	/orders/{orderId}/credit-notes	GET	Vrátí všechny dobropisy pro danou objednávku.
4	/orders/{orderId}/credit-notes	POST	Vytvoří pro danou objednávku nový dobropis se zadanými daty.
5	/orders/{orderId}/addresses	PUT	Nastaví dané objednávce zadané hodnoty týkající se adres (fakturační adresa a doručovací adresa).
6	/orders/{orderId}/order-status	POST	Vytvoří nový stav objednávky.
7	/orders/{orderId}/products	PUT	Nastaví dané objednávce předaný seznam produktů. Pro každý produkt v objednávce nastaví předané detaily o produktu (např. množství).
8	/orders/{orderId}/invoice	GET	Vrátí fakturu pro danou objednávku.
9	/orders/{orderId}/invoice	POST	Vygeneruje pro objednávku fakturu.
10	/orders/{orderId}/invoice	DELETE	Smaže fakturu pro danou objednávku.
11	/orders/{orderId}/shipping-payment-possibilities	GET	Vrátí seznam doručovacích a platebních možností pro danou objednávku.
12	/order-statuses	GET	Vrátí obecný seznam možných stavů objednávky.
13	/payment-methods	GET	Vrátí seznam možných způsobů platby.
14	/shipping-methods	GET	Vrátí seznam možných způsobů dopravy zboží.

2.2.7.2 Návrh API

Administrace bude za účelem správy objednávek vystavovat API skládající se z několika přístupových bodů. Navržené přístupové body jsou i s popisem uvedeny v tabulce 2.22. Opět platí, že všechny uvedené API endpointy budou mít ještě prefix `/api`, který není v tabulce z důvodu čitelnosti uveden.

Seznam uvedených přístupových bodů není zcela kompletní. Chybí některé pomocné přístupové body, které jsou součástí jiných částí systému. Jedná se například o seznam možných sazeb DPH nebo seznam zemí (pro použití v adrese).

2.2.7.3 Doménový model

V této podkapitole budou popsány transportní objekty a vazby mezi nimi. Transportními objekty jsou myšleny objekty, které slouží pro komunikaci vyvíjeného systému s vnějším světem. Klasický doménový model by totiž nepřinesl žádnou novou informaci, neboť data z databáze budou mapována na ekvivalentní třídy a tyto třídy mezi sebou nebudou mít žádné vazby (databáze neobsahuje cizí klíče). Třídy reprezentující data z databáze proto budou transformovány na transportní objekty.

Každý transportní objekt bude mít definovanou metodu `buildFromDO`, která bude sloužit pro vytvoření transportního objektu z objektu (či objektů) získaného z databáze. Tyto transportní objekty, budou použity při serializaci a deserializaci dat u dříve definovaných API endpointů. Model pro objednávky byl z důvodu čitelnosti rozdělen do dvou obrázků. Jedná se o obrázky 2.5 a 2.6. Některé entity v druhém modelu jsou graficky odlišeny (vyšrafovány). Tyto objekty jsou součástí jiných částí systému a nejsou tedy předmětem aktuálně popisovaného návrhu. Objekty `ShippingMethodTO`, `PaymentMathodTO` a `OrderStatusTO` jsou znázorněny jinou barvou, aby bylo zdůrazněno, že tyto tři objekty se vyskytují na obou obrázcích.

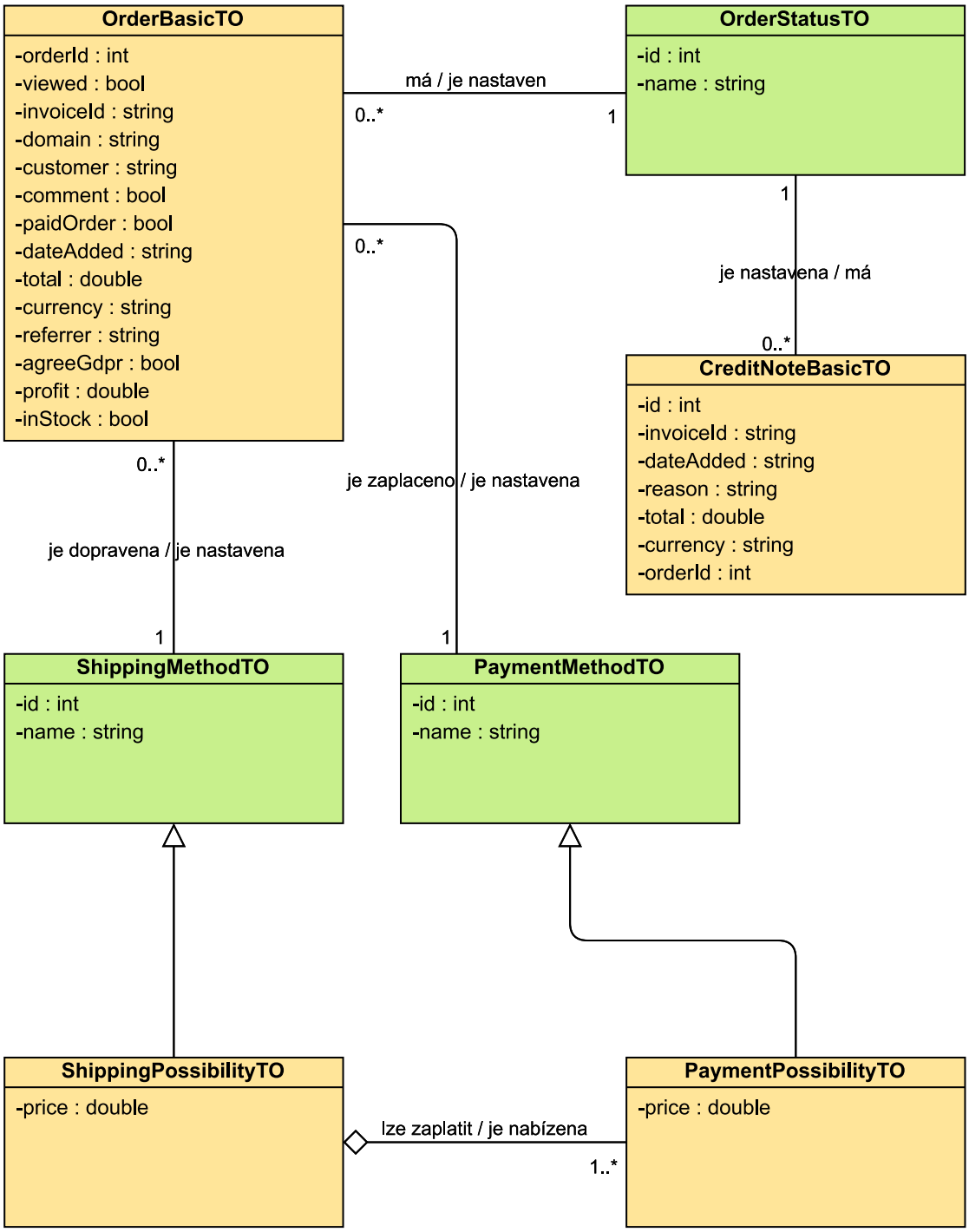
OrderStatusTO
Tento objekt představuje stav, ve kterém se může objednávka (nebo dobropis) nacházet. Příklady možných stavů jsou *K expedici*, *Nevyřízeno*, *Zákazník převzal zboží.*

Tabulka 2.23 Specifikace atributů třídy OrderStatusTO

Název atributu	Definice atributu
id	Identifikátor stavu
name	Název stavu

OrderBasicTO
Tato třída představuje zjednodušenou instanci objektu objednávky. Všechny atributy je možné získat přímo z databázových objektů s výjimkou atributu `profit`. Tento atribut je možné spočítat na základě ceny a pořizovací ceny každého produktu. Dále má tento objekt vazbu na stav objednávky, způsob platby a způsob doručení. Všechny tři zmíněné objekty zde budou také popsány.

Obrázek 2.5 Doménový model transportních objektů pro objednávky 1



Tabulka 2.24 Specifikace atributů třídy OrderBasicTO

Název atributu	Definice atributu
orderId	Identifikátor objednávky
viewed	Zda již byl zobrazen detail objednávky
invoiceId	Identifikátor faktury
domain	Doména, kde byla objednávka uskutečněna
customer	Jméno zákazníka
comment	Zda je u objednávky uveden nějaký komentář
paidOrder	Zda je objednávka již zaplacena
dateAdded	Datum vytvoření objednávky
total	Celková cena za objednávku
currency	Měna, ve které je cena uvedena
referrer	Stránka, ze které uživatel do e-shopu přišel
agreeGdpr	Zda byl udělen souhlas s GDPR
profit	Potencionální zisk na objednávce
inStock	Zda jsou všechny objednané produkty na skladu

CreditNoteBasicTO
Tento objekt reprezentuje zjednodušený pohled na dobropis, který je vždy propojen s nějakou objednávkou. Vazba na objednávku je z důvodu čitelnosti v modelu zaznamenána pouze pomocí identifikátoru v objektu `CreditNoteBasicTO` (v níže uvedené tabulce je vynechán). K jedné objednávce se může vztahovat více různých dobropisů.

Tabulka 2.25 Specifikace atributů třídy CreditNoteBasicTO

Název atributu	Definice atributu
id	Identifikátor dobropisu
invoiceId	Identifikátor faktury
dateAdded	Datum vytvoření
reason	Důvod vytvoření dobropisu
total	Celková hodnota položek v dobropisu
currency	Měna, ve které je cena uvedena

ShippingMethodTO
Tento objekt reprezentuje způsob doručení objednávky od obchodníka k zákazníkovi. Jedná se o velmi jednoduchý objekt, který se skládá pouze z identifikátoru a označení způsobu dopravy.

Tabulka 2.26 Specifikace atributů třídy ShippingMethodTO

Název atributu	Definice atributu
id	Identifikátor způsobu dopravy
name	Název způsobu dopravy

PaymentMethodTO
Tento objekt reprezentuje způsob platby za zboží (uhrazení objednávky). Jedná se o velmi jednoduchý objekt, který se skládá pouze z identifikátoru a označení způsobu platby.

Tabulka 2.27 Specifikace atributů třídy PaymentMethodTO

Název atributu	Definice atributu
id	Identifikátor způsobu platby
name	Název způsobu platby

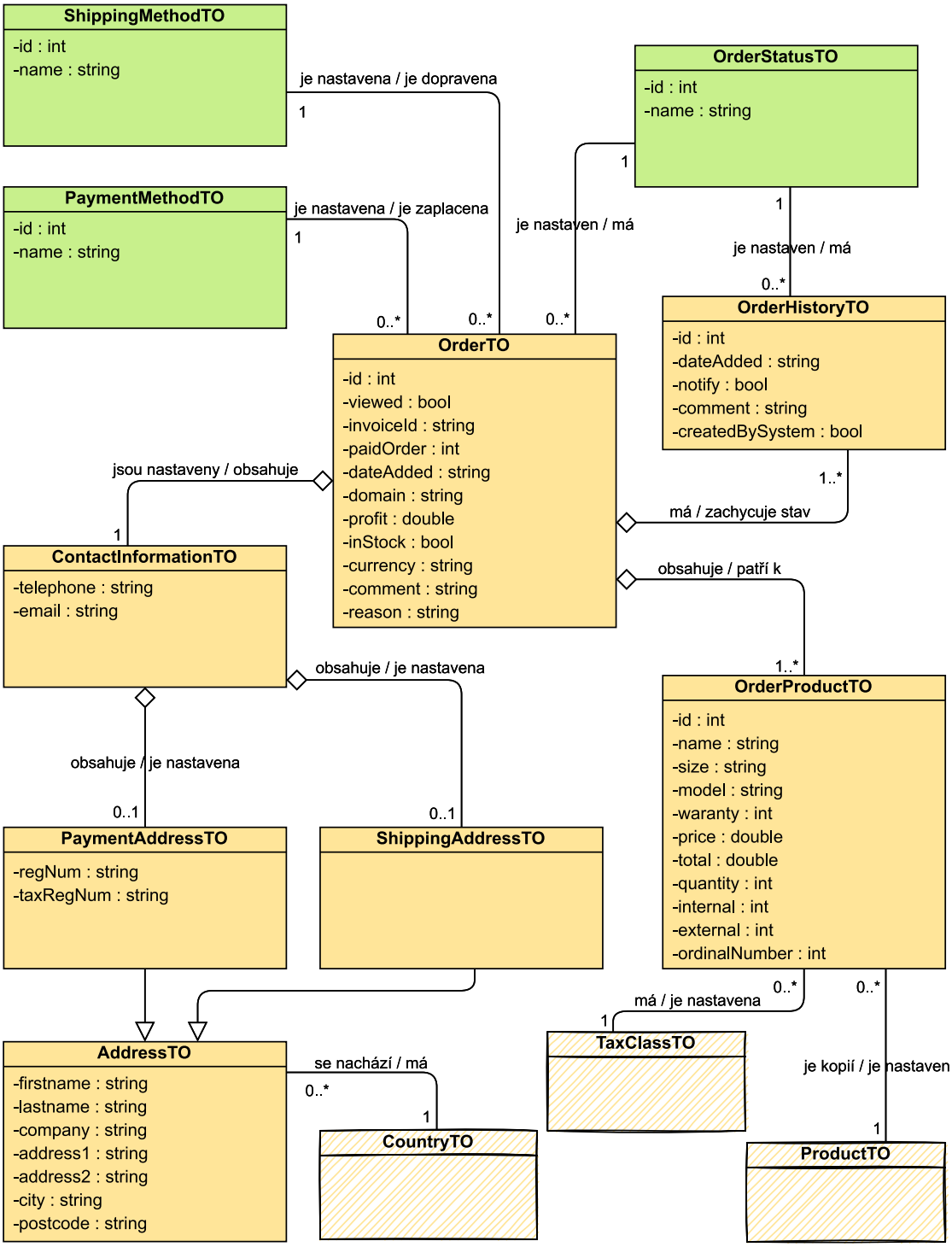
PaymentPossibilityTO
Tento objekt již reprezentuje konkrétní možnost platby na konkrétní doméně a pro konkrétní možnost dopravy (viz dále). Proto je již součástí objektu i cena. Tento objekt je tedy kombinací dříve popsaného objektu `PaymentMethodTO` a informace uvedené v nastavení pro konkrétní doménu.

ShippingPossibilityTO
Tento objekt reprezentuje konkrétní možnost dopravy, která je nabízena na konkrétní doméně. Proto je již součástí objektu i cena. Tento objekt dále odkazuje na jednu či více možností, jak je možné objednávku uhradit (`PaymentPossibilityTO`). Výsledná cena je později v rámci objednávky přidána podobně jako produkt, a proto objednávka nemá přímý vztah na tento objekt, ale jen na obecný způsob dopravy (a obecný způsob platby).

OrderTO
Objekt `OrderTO` je ústředním objektem této části návrhu. Jedná se o objekt představující objednávku a má vazbu na několik dalších objektů. Díky vazbě na `ContactInformationTO` objednávka obsahuje kontaktní informace, včetně fakturační a dodací adresy. Dále každá objednávka má svou historii stavů, kterými prošla (vazba na objekt `OrderHistoryTO`). V neposlední řadě každá objednávka ze své podstaty obsahuje seznam produktů, což je reprezentováno pomocí vazby na objekt `OrderProductTO`. Samozřejmě má objednávka nějaký aktuální stav, nastavený způsob doručení a způsob platby.

Tento objekt může také reprezentovat dobropis, který má úplně stejné atributy a jeden navíc. Jedná se o atribut `reason`. Popis v tabulce bude uveden pouze z pohledu objednávky (pro dobropis je ekvivalentní).

Obrázek 2.6 Doménový model transportních objektů pro objednávky 2



Tabulka 2.28 Specifikace atributů třídy OrderTO

Název atributu	Definice atributu
id	Identifikátor objednávky
viewed	Zda již byl zobrazen detail objednávky
invoiceId	Identifikátor faktury
paidOrder	Zda je objednávka již zaplacena
dateAdded	Datum vytvoření objednávky
domain	Doména, kde byla objednávka uskutečněna
profit	Potencionální zisk na objednávce
inStock	Zda jsou všechny objednané produkty ve skladu
currency	Měna, ve které je cena uvedena
comment	Zda je u objednávky uveden nějaký komentář
reason	Důvod pro vytvoření dobropisu (vyplněno pouze v případě dobropisu)

OrderHistoryTO
Tento objekt reprezentuje určitý stav, ve kterém se objednávka nacházela (nebo aktuálně nachází, pokud se jedná o poslední vytvořený objekt pro objednávku). Může také vyjadřovat stav, ve kterém se nacházel dobropis. V tomto objektu je zahrnuta i nově zavedená hodnota v podobě atributu `createdBySystem`, tedy zda daný stav byl vytvořen automaticky systémem, nebo ho vytvořil obchodník.

Tabulka 2.29 Specifikace atributů třídy OrderHistoryTO

Název atributu	Definice atributu
id	Identifikátor historického stavu
dateAdded	Datum, kdy záznam vznikl
notify	Zda byl zákazník upozorněn na změnu stavu
comment	Komentář k danému stavu
createdBySystem	Zda stav vytvořil (automaticky) systém

OrderProductTO
Tento objekt reprezentuje produkt v objednávce nebo v dobropisu. Jedná se o kopii nabízeného produktu, a proto má na původní produkt referenci. Kopie produktu se v systému musí dělat, neboť u produktů se může měnit cena a další položky. Toto například kvůli fakturám nelze připustit, a proto je nutné znát podobu produktu v době nákupu (vytvoření objednávky). Produkt má také vazbu na sazbu DPH (viz dále).

Tabulka 2.30 Specifikace atributů třídy OrderProductTO

Název atributu	Definice atributu
id	Identifikátor produktu objednávky
name	Název produktu
model	Označení modelu produktu
size	Velikost (používá se například u bot)
waranty	Délka záruky na produkt
price	Cena produktu
quantity	Množství (počet kusů)
total	Celková cena
internal	Počet kusů v (interním) skladu
external	Počet kusů v externích skladu
ordinalNumber	Pořadové číslo produktu v rámci objednávky

AddressTO

Tento objekt představuje společného předka pro dodací adresu a fakturační adresu. Dále zde existuje povinná vazba na zemi, pro kterou je zadaná adresa platná.

Tabulka 2.31 Specifikace atributů třídy AddressTO

Název atributu	Definice atributu
firstname	Křestní jméno zákazníka
lastname	Příjmení zákazníka
company	Název společnosti (pokud je objednávka na společnost)
address1	Adresa (ulice a číslo popisné)
address2	Upřesnění adresy
city	Město
postcode	Poštovní směrovací číslo

PaymentAddressTO

`PaymentAddressTO` je objekt představující fakturační adresu. Je vyplněn pouze pokud se tato adresa liší od dodací adresy nebo v případě nákupu na firmu.

Tabulka 2.32 Specifikace atributů třídy PaymentAddressTO

Název atributu	Definice atributu
regNum	IČO (nebo také IČ, identifikační číslo osoby)
taxRegNum	DIČ (daňové identifikační číslo)

ShippingAddressTO
`ShippingAddressTO` je objekt představující dodací adresu. Tento objekt vychází z objektu adresy a nepřidává žádnou další informaci.

ContactInformationTO
Tento objekt reprezentuje informace o zákazníkovi. Dále má tento objekt vazbu na dodací adresu, která by měla být systémem vždy vyžadována. Jedinou výjimkou, kdy není dodací adresa nutná, je situace, kdy zákazník vyzvedává objednávku osobně na pobočce. Vazba na fakturační adresu je volitelná, neboť při její absenci se použije dodací adresa.

Tabulka 2.33 Specifikace atributů třídy ContactInformationTO

Název atributu	Definice atributu
telephone	Telefonní číslo na zákazníka
email	E-mail na zákazníka

CountryTO, TaxClassTO a ProductTO
Tyto objekty jsou pomocné a pochází z jiných částí systému. Objekt `CountryTO` představuje zemi (používá se pro adresu). Objekt `ProductTO` byl zmíněn již dříve a představuje produkt, který je e-shopem nabízený zákazníkům. Posledním jmenovaným objektem je `TaxClassTO`, který představuje sazbu DPH (například sazbu 21 %).

Mapování DTO objektů na definované API endpointy
Pro lepší přehlednost bude nyní uvedena tabulka 2.34. V tabulce jsou zaznamenány vztahy mezi transportními objekty a jednotlivými API endpointy. Jinak řečeno, jaké hodnoty daný endpoint v optimálním případě vrací (výstup) a na které DTO se mapují vstupní data.

V tabulce je s hvězdičkou uveden objekt `CreditNoteCollection`. Tento objekt nikde výše nebyl popsán a toto pojmenování zde bylo použito pro zkrácení popisu objektu v tabulce. Daný objekt totiž nebyl navržen a jedná se pouze o obal pro dvě kolekce – kolekce dobropisů (`CreditNoteBasicTO`) a kolekce produktů (`OrderProductTO`).

2.2.8 Shrnutí API pro nový systém

Přesný a úplný popis všech API endpointů pro nový systém je uveden v dokumentaci na přiloženém médiu. API endpointy jsou zdokumentovány ve formátu OAS (zkratka ze slov *OpenAPI Specification*). OAS definuje standardizované jazykové rozhraní pro popis RESTful API. Tento popis se snaží být srozumitelný pro lidi i počítače, aby bylo možné pochopit a porozumět požadavkům na službu bez nutnosti přístupu ke zdrojovému kódu. [26]

2.3 Autorizační server

Dalším požadavkem k řešení je autorizační server. Cílem autorizačního serveru je spravovat uživatele, jejich role a především poskytovat službu autentizace a autorizace. V tomto případě bude jako autorizační server použit již hotový produkt Keycloak[6].

Keycloak byl zvolen na základě komunikace s firmou Jagu, s. r. o. Firma zvažuje použití tohoto serveru pro své projekty, a proto bude tento autorizační server použit pro tento projekt.

[6]Odkaz na webové stránky autorizačního serveru Keycloak je https://www.keycloak.org.

Tabulka 2.34 Použití DTO v jednotlivých API endpointech z tabulky 2.22.

Číslo endpointu	Výstup	Vstup mapován na DTO
1	OrderBasicTO (kolekce)	
2	OrderTO	
3	CreditNoteCollection*	
4		OrderTO
5		ContactInformationTO
6		OrderHistoryTO
7		OrderProductTO (kolekce)
8	(soubor, faktura)	
9	(identifikátor faktury)	
10		
11	ShippingPossibilityTO (kolekce)	
12	OrderStatusTO (kolekce)	
13	PaymentMethodTO (kolekce)	
14	ShippingMethodTO (kolekce)	

2.3.1 O autorizačním serveru Keycloak

"Keycloak poskytuje jednotné řešení pro webové aplikace a RESTful webové služby. Cílem Keycloak produktu je, aby zavedení ochrany bylo snadné, a proto bylo pro vývojáře snadné ochránit vyvíjené aplikace a služby, které provozuje jejich organizace. Bezpečnostní funkce, které musí vývojáři obvykle napsat sami, jsou poskytovány k jednoduchému a okamžitému použití. Produkt lze snadno přizpůsobit požadavkům vaší organizace."[7] [27, překlad autor]

Keycloak podporuje mnoho funkcí, například:

- podpora autorizačních frameworků OAuth 2.0 a OpenID Connect,
- administrátorská konzole pro centralizovanou správu uživatelů, rolí, mapování rolí, konfigurací,
- dvoufázové ověření (neboli 2FA). [28]

2.3.2 Základní principy komunikace

1. Uživatel otevře klientskou aplikaci, kde není aktuálně přihlášený.
2. Uživatel je typicky vyzván k přihlášení a po kliknutí (potvrzení) je přesměrován na Keycloak, kde provede přihlášení.

[7] Keycloak is a single sign on solution for web apps and RESTful web services. The goal of Keycloak is to make security simple so that it is easy for application developers to secure the apps and services they have deployed in their organization. Security features that developers normally have to write for themselves are provided out of the box and are easily tailorable to the individual requirements of your organization.

3. Po úspěšném přihlášení je uživatel přesměrován zpět na klientskou aplikaci a klientská aplikace získá ID token, refresh token a access token.
4. Klientská aplikace posílá požadavky na server s vyplněnou hlavičkou `Authorization`, ve formátu `Authorization: Bearer <access token>`.
5. Server provede validaci tokenu. Možnosti k této problematice budou popsány níže.
6. Pokud server zjistí, že token není validní, tak vrátí HTTP status code 401 (Unauthorized).
7. Pokud server úspěšně ověří platnost tokenu, je zpracován požadavek a podle role jsou případně vrácena data.

Bearer autentizace, nebo též autentizace pomocí tokenu, je autentizační schéma, které pracuje s bezpečnostními tokeny nazývanými bearer tokeny. Tento token je šifrovaný text, který byl vygenerován serverem. Obvyklým formátem je JWT (zkratka z anglického označení *JSON Web Token*). A jak anglický název napovídá, jedná se o formát inspirovaný formátem JSON. Ve vyvíjené aplikaci se budou používat právě popsané bearer tokeny ve formátu JWT. [29]

Existují dva základní přístupy, jak validovat bearer token - introspekce a ověření podpisu, které budou níže popsány.

Validace pomocí introspekce, anglicky *introspection*, funguje tak, že server zavolá endpoint autorizačního serveru a vyžádá si informace o tokenu. Výhodou tohoto přístupu je, že je možné poznat tzv. revoke (termín označuje zrušení tokenu, tedy odhlášení). Informace o tokenu, které přijdou v odpovědi, budou ve formátu JWT, který byl představen v předchozím odstavci. [30]

Druhou možností je validace pomocí JWKs. Znak *s*, který je na konci zkratky, označuje, že se jedná o množinu JWK. Zkratka JWK označuje datovou strukturu ve formátu JSON, která označuje kryptografický klíč. Tento formát se používá pro sdílení veřejných klíčů mezi různými službami. Pomocí JWKs je tedy možné ověřit platnost podepsaného tokenu. Bohužel tato metoda nepozná tzv. revoke (termín označuje zrušení tokenu, tedy odhlášení). [31]

Pro vyvíjenou aplikaci bude použita validace pomocí JWKs. V bearer tokenu jsou obsaženy informace o uživateli, včetně rolí. Pro novou administraci bude předpokládána jediná role, která bude označena jako `ROLE_USER`. Nová administrace musí definovat seznam rolí, který bude zatím čítat pouze jedinou (již zmíněnou) roli. Hlavním předpokladem je v budoucnosti možnost snadného rozšíření tohoto seznamu.

2.4 Zvolené technologie

V této podkapitole budou shrnuty a popsány zvolené technologie. Volba těchto technologií se řídila požadavky na nový systém s přihlédnutím k technologiím, které firma Jagu, s. r. o., už používá. V této podkapitole již nebude věnována pozornost autorizačnímu serveru Keycloak, kterému byla věnována velká pozornost v předchozí kapitole.

2.4.1 PHP

PHP je široce užívaným a univerzálním skriptovacím jazykem. Tento open-source jazyk (neboli jazyk s otevřeným zdrojovým kódem) je vhodný především pro tvorbu webových aplikací. PHP kód je vykonáván na serveru, což je podstatný rozdíl například od jazyku JavaScript, který pracuje na straně klienta. V případě jazyka PHP tedy klient obdrží výsledek běhu PHP kódu, nikoliv PHP kód, který by měl být vykonán. Jelikož je PHP multiplatformní, tak může běžet na serveru s téměř libovolným operačním systémem – MacOSX, Linux, Windows. [32][33]

PHP jazyk vznikl v polovině 90. let 20. století a rozvíjí se až dodnes. Jazyk již tedy existuje delší dobu, nezanikl a prošel si dlouhou cestu. Aktuální a nejnovější je verze PHP 8.1, která bude použita pro nově vyvíjený systém. [34]

2.4.2 Symfony

"Symfony je seskupení PHP komponent, Web Application framework, filozofie a komunity."[8] [35, překlad autor]

Symfony je framework založený na jazyku PHP. Framework označuje množinu nástrojů a metodiku pro vývoj aplikace. Symfony je open-source framework, tedy framework s otevřeným zdrojovým kódem, pro vývoj webových aplikací a jeho velkou předností je velmi kvalitní dokumentace a široká komunita vývojářů z celého světa. [35]

Symfony je založené na MVC architektuře, která již byla popsána v jedné z předcházejících kapitol. Tato architektura je velmi rozšířená a přináší dobrý základ do nově vznikající aplikace. [23]

Tento framework je stále rozšiřován a udržován společností SensioLabs s pomocí více než tří tisíc dobrovolných přispěvatelů. [36]

2.4.3 MySQL

MySQL je relační databázový systém vyvinutý společností Oracle, který je založený na dotazovacím jazyku SQL. Jeho první verze byla vydána 23. května 1995. [37]

Hlavní výhodou tohoto databázového systému je jeho široká podpora zahrnující operační systémy založené na Unixu, jako například mnoho distribucí Linuxu či macOS, a Windows. [37]

Jedná se o open-source řešení, které je využíváno řadou velkých firem a má širokou komunitní základnu. Díky velké komunitní podpoře existuje mnoho knihoven pro práci s MySQL databázemi pro téměř každý programovací jazyk. [37]

2.4.4 Doctrine ORM

Doctrine je knihovna určená pro jazyk PHP (verze PHP 7.1 nebo vyšší) pro objektově relační mapování. Tato knihovna tedy zprostředkovává přístup k úložišti, kam se ukládají PHP objekty. [38]

Doctrine používá Data mapper pattern (pattern neboli návrhový vzor), který odděluje doménový model od databáze a práci s ní. Obsahuje objekt pro práci s databází (Doctrine tyto objekty nazývá *Repository*), který například podle zadaného dotazu vrátí instanci objektu představujícího záznam v databázi. Tento vrácený objekt je objekt patřící do doménového modelu a díky tomuto oddělení je možné se zaměřit na objektově orientovanou business logiku a úložiště řešit samostatně. [38]

2.4.5 nginx

Nginx je HTTP a reverzní proxy server, který vyvinul a vydal v roce 2004 Igor Sysoev. V současné době vývoj a údržbu serveru zajišťuje společnost Nginx, Inc. Nginx poskytuje také mail proxy server a umožňuje řídit jakýkoliv provoz přicházející přes TCP a UDP protokoly. [39]

Nginx server je otestován pro širokou škálu operačních systémů a platforem, například Solaris, macOS, Windows 10. Kompletní seznam je k nalezení v oficiální dokumentaci pro server nginx. [39]

Dle společnosti Netcraft obsluhoval server nginx v květnu roku 2022 21,67 % nejvytíženějších stránek na internetu. [39]

[8] Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony

2.4.6 Docker

Docker je otevřená platforma (anglicky *open platform*) pro vývoj, doručení a běh programu. Docker umožňuje oddělit aplikaci od infrastruktury, a proto je doručení aplikace rychlejší. S platformou Docker je možné spravovat infrastrukturu úplně stejně jako konfigurovat provozovanou aplikaci. [40]

Docker definuje izolovaná prostředí, která se nazývají kontejnery (anglicky *containers*). Kontejner představuje balíček obsahující knihovny a další zdroje nutné pro běh aplikace. Na daném prostředí je pak možné spustit mnoho kontejnerů současně. Každý kontejner obsahuje vše, co je nutné pro běh aplikace, a díky tomu nezáleží, co je aktuálně nainstalované v současném systému. Proto je jednoduché sdílet kontejnery a být si jistý, že všichni budou mít stejné prostředí, které bude fungovat vždy stejně. [40]

Kapitola 3
Realizace

Tato kapitola se bude věnovat implementaci navrženého systému. Jelikož výstupem této části práce je především implementace a vytvoření navrženého systému, budou v textu popsány pouze vybrané části (zajímavé implementační detaily). Nejprve bude krátce popsán Docker a dvě připravená docker prostředí. Následovat bude krátký popis nastavení CI a několik zajímavějších implementačních detailů z vývoje nového systému pro administraci.

3.1 Docker

Prvním krokem je příprava prostředí pomocí technologie nazývané Docker, aby všichni vývojáři měli stejné běhové prostředí. Výhodou je také snadné spuštění prostředí a vyvíjeného systému. Tato podkapitola velmi úzce souvisí s následující kapitolou věnující se nastavení projektu a CI.

Jak již bylo zmíněno dříve, prostředí se bude skládat ze tří komponent – databáze, webového serveru a modulu s PHP. Nejdříve se pro každou komponentu definuje soubor zvaný *Dockerfile*. Uvnitř tohoto souboru se definuje prostředí, které je nezbytné pro běh dané komponenty. Mnohdy lze využít některý z již vytvořených oficiálních obrazů (anglický a používanější výraz je *image*). [41]

Pokud jsou již kontejnery (*Dockerfile*) pro komponenty připravené, dalším krokem je vytvoření souboru, který se nazývá *docker-compose*. Tento soubor sdružuje komponenty k sobě a umožňuje jejich hromadné spuštění v izolovaném prostředí. Každá komponenta v tomto souboru je označována jako služba (anglicky *service*) a definuje se u ní například:

- její jméno,
- na kterých definovaných službách je závislá,
- cesta k souboru *Dockerfile*,
- na kterém portu služba běží,
- takzvaný *volume* (svazek či úložiště), který umožňuje namapovat složky a soubory do adresářové struktury vytvořené uvnitř daného kontejneru. [41]

3.1.1 Implementace

Byly vytvořeny dva *docker-compose* soubory. První je pojmenován `docker-compose.yaml` a je proto výchozí variantou. Tato varianta docker prostředí se připojuje na vzdálenou databázi a

tudíž komponenta, která se stará o databázi, nedefinuje databázi přímo, ale definuje pouze připojení k ní. V připravené šabloně tohoto souboru nejsou vyplněny všechny informace, protože pro připojení ke vzdálené databázi jsou nutné přístupové údaje. Každý uživatel má samozřejmě vlastní.

Druhý soubor je pojmenován `docker-compose-local.yaml` a jedná se o variantu, která inicializuje lokální databázi. Tento docker je při startu pomalejší, neboť je nutné inicializovat lokální databázi, ale umožňuje pracovat se systémem i bez přístupu k internetu.

3.2 Nastavení projektu (CI)

V této kapitole bude popsáno CI pro nový systém administrace. Co je to CI neboli Continuous Integration? Tento pojem vyjadřuje automatickou integraci změn kódu od více uživatelů (vývojářů). CI je také jeden z klíčových bodů DevOps. DevOps je spojení, které vzniklo ze slov *development* a *IT operations* a neoznačuje pouze jednu metodologii. Jedná se o způsob myšlení, filosofii vývoje softwaru, množinu takzvaných *best practises* a zahrnuje i mnoho používaných nástrojů a metodologií. DevOps obvykle zahrnuje agilní vývoj, automatizaci, continuous integration, continuous delivery, continuous deployment, monitorování, sledování a průběžné vyhodnocování (na základě zpětných vazeb, anglicky *feedback*). [42][43]

Hlavním cílem CI je tedy umožnit snadnou spolupráci mnoha vývojářů nad společným kódem. Umožňuje časté slučováním změn v kódu (anglický běžně užívaný termín *merge*) do centrálního úložiště, kde probíhá automatické sestavení (anglicky *build*) aplikace a testování. Tyto automatizované postupy a nástroje umožňují pravidelnou kontrolu kódu aplikace. [42]

3.2.1 Docker

Prvním podstatným krokem přípravy vývojového prostředí je použití virtualizační technologie Docker, která již byla popsána v předchozí podkapitole. Díky této technologii budou mít všichni vývojáři jednotné vývojové prostředí a jeho inicializace bude velmi jednoduchá.

Dalším krokem a jádrem procesu CI je verzovací systém, kterým je v rámci této práce GitLab. GitLab je centralizovaný verzovací systém a pro tuto práci byl poskytnut firmou Jagu, s. r. o. GitLab poskytuje nástroj GitLab CI, který bude popsán v následující kapitole.

3.2.2 GitLab CI

GitLab CI je jedním z mnoha nástrojů poskytovaných nástrojem GitLab. Tento nástroj umožňuje definovat seznam úloh, které mají být automaticky vykonány po nahrání jakékoliv změny do repositáře projektu. [44]

Prvním krokem, který je nutné provést, je vytvořit konfigurační soubor *.gitlab-ci.yml* v kořenovém adresáři repositáře. Tento konfigurační soubor využívá syntaxi jazyka YAML. V souboru se zapisuje seznam kroků, které mají být vykonány. Dále se v něm definuje prostředí, ve kterém se budou jednotlivé kroky vykonávat (velice podobné s dříve popsanou technologií Docker).

Dále už stačí pouze nahrát do repositáře nějaké změny a nástroj GitLab CI se o vše sám postará. GitLab byl v rámci této práce poskytnut společností Jagu, s. r. o. Na něm jsou umístěny všechny zdrojové kódy, které byly v rámci této práce vytvořeny.

3.2.2.1 Konfigurační soubor pro GitLab CI

V rámci konfiguračního souboru se definují fáze (anglicky *stages*). Pro vyvíjený systém byly definovány dvě fáze:

- první pro statickou analýzu kódu pomocí knihovny PHPStan (tento nástroj bude popsán později),

- druhá fáze pro spuštění PHPUnit testů (rovněž bude tato knihovna popsána později).

Pro statickou analýzu kódu není potřeba běžící aplikace, a proto byla tato fáze definována jako první. Jelikož autor této práce neměl s nástrojem GitLab CI žádné předešlé zkušenosti, bylo nastavení automatického spouštění PHPUnit testů náročnější. Pro tuto fázi bylo nutné nadefinovat MySQL databázi, inicializovat ji (vytvořit všechny potřebné tabulky) a spustit migrační skripty. Upravený a zkrácený konfigurační soubor je uveden v příloze E.

3.3 Systém pro administraci

Nyní je již pro vývoj připravený docker, je nastavené CI ve vytvořeném repositáři na GitLabu a tudíž může začít vývoj nového systému pro administraci naplno. V této podkapitole budou popsána pouze vybraná témata. Témata, která jsou buď pro nový systém velmi významná, nebo něčím zajímavá. Na úvod bude uveden stručný výpis hlavních knihoven, které budou v projektu použity.

3.3.1 Prostředí a hlavní knihovny

V této kapitole budou krátce představeny některé použité knihovny. Mnoho použitých knihoven je poskytováno samotným frameworkem Symfony.

symfony/filesystem[1] označuje knihovnu poskytující základní operace nad souborovým systémem (anglicky *file system*).

doctrine/orm[2] označuje již zmíněnou knihovnu Doctrine ORM (kapitola 2.4.4). Tato knihovna poskytuje rozhraní pro práci s databází.

friendsofsymfony/rest-bundle[3] je knihovna poskytující mnoho nástrojů pro vývoj RESTful API. Provádí například serializaci objektů.

symfony/security-bundle[4] označuje knihovnu poskytující integraci Security komponenty. Tato knihovna bude využita později při implementaci autentizace a autorizace.

symfony/form[5] je knihovna pro formuláře. Symfony formuláře jsou efektivní způsob, jak validovat vstupní data.

zenstruck/schedule-bundle[6] je knihovna, která poskytuje funkcionalitu plánování úkolů (anglicky *task* nebo také *job*). Každý úkol je možné naplánovat na specifický interval, ve kterém se bude spouštět.

Jak již bylo uvedeno v návrhu, všechny API endpointy budou mít nastavený prefix `/api`. Toto nastavení musí být provedeno v rámci konfigurace *FOSRestBundle*. Stejné pravidlo musí být taktéž v konfiguračním souboru `config/routes/annotations.yaml`, a to z důvodu směrování, anglicky *routing* (to totiž zajišťuje standardně Symfony).

Podstatná je také konfigurace pro autentizaci a autorizaci, která bude uvedena v jedné z pozdějších kapitol.

[1] Knihovna je dostupná z oficiálního package repositáře https://packagist.org/packages/symfony/filesystem.
[2] Knihovna je dostupná z oficiálního package repositáře https://packagist.org/packages/doctrine/orm.
[3] Knihovna je dostupná z oficiálního package repositáře https://packagist.org/packages/friendsofsymfony/rest-bundle.
[4] Knihovna je dostupná z oficiálního package repositáře https://packagist.org/packages/symfony/security-bundle.
[5] Knihovna je dostupná z oficiálního package repositáře https://packagist.org/packages/symfony/form.
[6] Knihovna je dostupná z oficiálního package repositáře https://packagist.org/packages/zenstruck/schedule-bundle.

3.3.2 Domény

V rámci návrhu bylo rozhodnuto o zavedení takzvané výchozí domény. Hlavní motivací bylo mít v seznamů domén vždy jednu hlavní doménu a od té odvozovat texty pro ostatní domény. Tato výchozí doména byla zavedena pro výrobce a kategorie.

Jako velmi náročné, z pohledu business logiky, se nakonec ukázaly dva API endpointy. Konkrétně jeden pro výrobce a jeden pro kategorie:

- `/manufacturers/{id}/domains/{domain}` (PUT),
- `/categories/{id}/domains/{domain}` (PUT).

U těchto dvou endpointů se přijatá data mapují na objekt `DomainManufacturerWrapperTO` (pro výrobce) nebo `DomainCategoryWrapperTO` (pro kategorie). V obou případech objekty obsahují atribut `default`, který bude později důležitým rozhodovacím kritériem. V případě obou přístupových bodů může nastat několik situací, které je nutné vyřešit. Konkrétně mohou nastat tři situace:

- úprava výchozí domény,
- úprava nevýchozí domény,
- změna výchozí domény.

Nejprve bude znovu připomenut výraz override, který byl již zmíněn dříve. Jedná se o záznam v tabulce `oc_domain_override`, který pro danou doménu definuje jinou hodnotu, než je uvedená v hlavní tabulce. Jinak řečeno, přetěžuje hodnotu uvedenou v hlavní tabulce. Pojmem hlavní tabulka jsou souhrnně označeny tabulky `oc_manufacturer`, `oc_manufacturer_description`, `oc_category` a `oc_category_description`.

V ideálním případě by override záznamy asi neexistovaly, protože mnoho věcí se díky nim komplikuje. Například kvůli nim existuje tabulka `oc_category_cache`, která obsahuje předpočítané hodnoty pro kategorie v závislosti na doméně (tato tabulka se při každé změně musí aktualizovat).

3.3.2.1 Ilustrativní situace

V původním návrhu se předpokládalo, že výchozí doména bude mít vždy všechny hodnoty v hlavních tabulkách. Tento předpoklad byl však mylný a samozřejmě existuje i možnost override hodnoty pro výchozí doménu. Kvůli zajištění stejného chování současné a nové administrace bylo tedy nutné připustit možnost, že výchozí doména může mít override. Tím se však mnoho věcí značně komplikuje, a proto bude představen příklad, na kterém bude později ilustrováno několik problémů a následných řešení.

Pro jednoduchost bude použit výrobce s identifikátorem *1*. Tento výrobce má pouze jméno a je definován pro tři domény. Nechť se jedná o domény: *alfa.cz*, *beta.cz*, *gama.cz*. První jmenovaná doména nechť je výchozí doménou. V tabulce `oc_manufacturer` bude uloženo jako jméno výrobce *Filip*. Pro doménu *alfa.cz* bude existovat override s hodnotou *FilipA*. Pro doménu *beta.cz* bude existovat override s hodnotou *FilipB*. Pro doménu *gama.cz* se bude používat hodnota uložená v tabulce výrobce. Pro lepší čitelnost je tato situace zanesena také v tabulce 3.1.

3.3.2.2 Úprava výchozí domény

Úprava výchozí domény je první zmíněnou situací, která nastane, pokud je URI přístupového bodu použita pro výchozí doménu nějakého výrobce či kategorie. Ve zvoleném ilustrativním případě by se jednalo o `/manufacturers/1/domains/alfa.cz`. V tomto případě musí být hodnota

Tabulka 3.1 Ilustrativní situace pro popis práce s doménami (* označuje výchozí doménu)

Název domény	Výchozí hodnota	Specifická hodnota
alfa.cz *	Filip	FilipA
beta.cz	Filip	FilipB
gama.cz	Filip	*(nespecifikováno)*

atributu `default` vždy `true`. Pokud je hodnota tohoto atributu `false`, jedná se o nevalidní data. Pokud pro výchozí doménu neexistuje override, je možné změnit hodnotu přímo v hlavní tabulce. Pokud pro výchozí doménu existuje override, musí být aktualizován override záznam a příslušné záznamy v hlavním tabulkách zůstanou nezměněny.

Celou situaci lze demonstrovat na připraveném příkladu, kde uživatel vidí, že hodnoty jsou *FilipA* (pro první a zároveň výchozí doménu), *FilipB* (pro druhou doménu) a *Filip* (pro poslední doménu). Pokud má být změněna hodnota výchozí domény, která je z vnějšího pohledu *FilipA*, tak je nutné změnit pouze hodnotu override hodnoty, protože jinak se změní hodnota i pro doménu *gama.cz*.

3.3.2.3 Úprava nevýchozí domény

Další možnou situací je úprava nevýchozí domény. Tato situace nastane, pokud je URI přístupového bodu použita pro nevýchozí doménu výrobce či kategorie. Ve zvoleném ilustrativním příkladě by se jednalo například o `/manufacturers/1/domains/gama.cz`. Dále by hodnota atributu `default` byla nastavena na hodnotu `false`.

V tomto případě se přijatá data porovnávají s výchozími daty v hlavních tabulkách. Pokud se hodnoty shodují, tak se override záznam nevytváří (a pokud existuje, tak se smaže). Pokud se hodnoty liší, tak se vytvoří (nebo aktualizuje) override záznam.

Popsané chování je možné opět demonstrovat na ilustrativním příkladě. Pro doménu *gama.cz* by se override záznam s hodnotou *Filip* nevytvořil, neboť tato hodnota je výchozí (díky tomu, že je nastavena v hlavní tabulce). Naopak pro hodnotu *FilipA* nebo *FilipC* by bylo vytvoření override záznamu nezbytné.

3.3.2.4 Změna výchozí domény

Poslední možnou situací je změna výchozí domény. Tato situace nastane, pokud je URI přístupového bodu zavolána pro nevýchozí doménu výrobce či kategorie. Ve zvoleném ilustrativním příkladě by se jednalo například o `/manufacturers/1/domains/beta.cz`. Dále by hodnota atributu `default` byla nastavena na hodnotu `true`.

V tomto případě je vhodné celý proces rozdělit na dva kroky. Nejprve se provede úprava původní výchozí domény. Úprava v tomto případě znamená, že se všechny override hodnoty pro výchozí doménu přepíšou do hlavních tabulek. Následně se upraví související záznamy tak, aby z vnějšího pohledu nebyl vidět žádný rozdíl. Pro ilustrativní příklad by to znamenalo přepsání override hodnoty *FilipA* do sloupce *Výchozí hodnota* (a smazání tohoto overridu), čímž by se tato hodnota stala výchozí hodnotou pro všechny domény. Proto se následně musí vytvořil nový override záznam pro doménu *gama.cz* s hodnotou *Filip*. Ilustrativní příklad by po těchto úpravách vypadala tak, jak je znázorněno v tabulce 3.2. Jak je z tabulky zřejmé, při pohledu z vnějšího světa se nic nezměnilo, avšak nyní neexistují override záznamy pro výchozí doménu.

Druhým krokem je samotná změna výchozí domény. Při této operaci se nejdříve smažou všechny override záznamy pro novou výchozí doménu, neboť tyto záznamy nejsou potřeba (nová data pro tuto doménu přišla společně s požadavkem na změnu výchozí domény). Následně jsou

Tabulka 3.2 Ilustrativní situace pro popis práce s doménami 2 (* označuje výchozí doménu)

Název domény	Výchozí hodnota	Specifická hodnota
alfa.cz *	FilipA	*(nespecifikováno)*
beta.cz	FilipA	FilipB
gama.cz	FilipA	Filip

Tabulka 3.3 Ilustrativní situace pro popis práce s doménami 3 (* označuje výchozí doménu)

Název domény	Výchozí hodnota	Specifická hodnota
alfa.cz	FilipZ	FilipA
beta.cz *	FilipZ	*(nespecifikováno)*
gama.cz	FilipZ	Filip

nová data zapsána do hlavním tabulek. Pro záznamy, kde se liší nová data a data původní výchozí domény, se vytvoří override záznam pro původní výchozí doménu.

Opět pro lepší představu bude tento krok prakticky ukázán na příkladu (výchozí stav nechť zachycuje tabulka 3.2). Novou výchozí doménou bude *beta.cz*, jak již bylo zmíněno na začátku této podkapitoly. Společně s požadavkem na změnu bylo nové jméno výrobce pro tuto doménu nastaveno na *FilipZ*. Nejprve se tedy smaže override záznam pro doménu *beta.cz*, protože již není aktuální (nové jméno je *FilipZ*). Jelikož se liší hodnota pro současnou výchozí doménu (*FilipA*) a novou výchozí doménu (*FilipZ*), tak se vytvoří override záznam pro doménu *alfa.cz* s hodnotou *FilipA*. Následně se změní výchozí doména a výchozí hodnota pro jméno výrobce bude *FilipZ*. Konečná podoba je zaznamenána v tabulce 3.3.

3.3.3 Externí kategorie

V rámci této podkapitoly nebude popsána implementace API endpointů pro externí kategorie, protože se jednalo o standardní implementaci a v návrhu již bylo shrnuto vše podstatné. V této podkapitole budou popsány některé detaily týkající se implementace automatizované aktualizace externích kategorií.

Nejprve bylo nutné vytvořit spustitelný příkaz, který Symfony označuje jako *Console Commands* [45]. Pro přehlednost je vhodné příkazy vyčlenit do samostatného adresáře, např. `Command`, podobně jako Controllery jsou v adresáři `Controller`. Vytvořený příkaz musí rozšiřovat Symfony třídu `Command`. V příkazu lze například pomocí anotace nastavit jméno a popis. Kompletní popis s příklady a ukázkami lze nalézt v oficiální dokumentaci Symfony [45].

Po předchozím kroku je možné nový příkaz kdykoliv manuálně zavolat z konzole. Pro automatické spouštění tohoto příkazu byla využita již zmíněná knihovna *zenstruck/schedule-bundle*. Pro automatické spuštění příkazu je jednou z variant implementace rozhraní (anglický a běžně používaný výraz je *interface*) `SelfSchedulingCommand` a implementovat metodu `schedule`, která danou úlohu naplánuje [46].

Vytvořený příkaz je uveden v příloze C. Automatické spuštění bylo nastaveno na pondělí v jednu hodinu ráno. Takto vytvořený příkaz však nefungoval a v následující podkapitole bude vysvětleno proč.

3.3.3.1 Vytvořený a naplánovaný příkaz nefunguje

Příkaz je možno manuálně spustit, ale nefunguje automatické spuštění. Dle [47] a dokumentace v tomto repositáři by mělo vše fungovat, neboť byly následovány všechny instrukce. Tedy, téměř všechny instrukce (viz dále).

Nějakou dobu se nedařilo nalézt problém a řešení. Onen problém, který zde nastal, je naznačen již v samotném úvodu popisu této knihovny, který zní:

"Schedule Cron jobs (commands/callbacks/bash scripts) within your Symfony application." [47]

V překladu to znamená, že knihovna pomáhá naplánovat Cron úlohy v Symfony aplikaci. Co však znamená termín *Cron úloha*?

Cron je nástroj pro plánování úloh v operačních systémech Unix a jim podobných. Díky tomuto nástroji je možné naplánovat spuštění jakéhokoliv skriptu na specifický čas, datum nebo interval. Je to démon (anglicky *daemon*), který běží na pozadí operačního systému. Úlohám, které jsou již naplánovány, se říká *Cron jobs*. [48]

Pro vyřešení tohoto problému tedy stačí přidat do Docker konfigurace pro PHP kontejner knihovnu a nástroj Cron. Jakmile je tento nástroj součástí prostředí, ve kterém běží vyvíjená aplikace, je už pouze nutné spustit příkaz, který je uveden i v dokumentaci a vypadá následovně.

```
* * * * * cd /path-to-your-project \
&& php bin/console schedule:run >> /dev/null 2>&1
```

Tento příkaz byl při prvotní implementaci opomenut. Příkaz spustí na pozadí úlohu, která každou minutu zkontroluje, zda se nemá spustit nějaký CronJob. Aby se tato instrukce nemusela vždy ručně zadávat po spuštění prostředí, tak i tento příkaz bude součástí konfigurace pro Docker kontejner PHP.

3.3.4 Autentizace a autorizace

V rámci této podkapitoly bude popsána autentizace a autorizace pro nový systém administrace. Základem pro funkční zabezpečení je knihovna *symfony/security-bundle*, která byla zmíněna a stručně popsána již v dřívější podkapitole. Tato knihovna je samozřejmě podrobně zdokumentována v oficiální dokumentaci frameworku Symfony a přináší do aplikace nový konfigurační soubor `config/packages/security.yaml` [49].

Upravený konfigurační soubor je pro ukázku uveden v příloze D. V ukázce je vidět nastavení pro `access_control`, tedy řízení přístupu (autorizace). V této sekci se definuje, které role jsou nezbytné pro přístup k daným API endpointům. Tento seznam je hierarchický, a proto se aplikuje první pravidlo, které bude vyhovovat API endpointu, který uživatel zkouší zavolat. Dle návrhu je zde zatím uvedena pouze jediná role a to `ROLE_USER`.

Z důvodu praktické ukázky je v sekci `role_hierarchy` definována role `ROLE_ADMIN`, která dědí všechna práva, která má role `ROLE_USER`. V ukázce konfigurace je také v komentáři uveden příklad s rolí `ROLE_SUPER_ADMIN`.

Z ukázky je také zřejmé, že administrace implementuje vlastní třídu pro autentizaci, která je pojmenována jako `KeycloakAuthenticator`. Tato třída musí dle pravidel použité Symfony knihovny rozšiřovat abstraktní třídu `AbstractAuthenticator`. Proto musí nová třída pro autentizaci implementovat následující čtyři metody.

support je metoda, která je volaná při každém požadavku (*requestu*) a rozhoduje, zda by měla být daná třída použita pro autentizaci.

authenticate je metoda, která provádí autentizaci. V případě `KeycloakAuthenticator` pouze ověří, že request obsahuje hlavičku `Authorization`, ve které musí být nějaký bearer token.

onAuthenticationSuccess je metoda volaná v případě úspěšné autentizace (obvykle bývá prázdná, protože v tomto případě není další akce nutná).

onAuthenticationFailure je metoda vyjadřující selhání při autentizaci. V této metodě již obvykle bývá nějaký kód a v případě `KeycloakAuthenticator` je vrácena odpověď ve formátu JSON (popis chyby) s HTTP statusem 401.

V nové administraci je kromě třídy pro autentizaci (`KeycloakAuthenticator`) definována třída představující uživatele, `KeycloakUser`. Uživatel má definované uživatelské jméno, identifikátor a seznam rolí. Instance uživatele vzniká ve třídě `KeycloakUserProvider`, která je také nastavena v konfiguračním souboru.

Funkce třídy `KeycloakUserProvider` je vcelku zřejmá z názvu (*provider*, tedy poskytovatel). Tato třída v rámci procesu úspěšné autentizace poskytne instanci uživatele, který je reprezentován již zmíněnou třídou `KeycloakUser`. Třída `KeycloakUserProvider` implementuje rozhraní `UserProviderInterface`, které je opět definováno frameworkem Symfony. Díky tomu je opět nutná implementace několika metod, z nichž nejdůležitější je metoda `loadUserByIdentifier`. Tato metoda je volána na konci dříve uvedené metody `authenticate`.

Uvnitř metody `loadUserByIdentifier` se provádí validace tokenu. Pro validaci tokenu je však nezbytná znalost veřejného klíče autorizačního serveru (celý postup je uveden níže).

1. Zavolá se příslušný API endpoint autorizačního serveru pro získání veřejného klíče.
2. Získaný veřejný klíč se uloží do paměti (cache) na dobu dvou hodin (tedy načítání veřejného klíče se dle aktuální implementace provádí maximálně jednou za dvě hodiny).
3. Veřejný klíč se použije k ověření platnosti tokenu a mohou nastat dvě možnosti.
   a. V případě neplatného tokenu (či jakéhokoliv jiného selhání při validaci tokenu) se vyhodí výjimka, která se dostane zpátky do třídy `KeycloakAuthenticator`, kde se zavolá již zmíněná metoda `onAuthenticationFailure`.
   b. Token je validní. V tomto případě jsou z tokenu získány informace o uživateli (uživatelské jméno, identifikátor, seznam rolí).
4. Informace získané z tokenu jsou použity k vytvoření instance uživatele. Seznam rolí přijatých v tokenu je v tomto procesu namapován na seznam rolí definovaných v systému administrace (viz `ROLE_USER`).

Na základě vytvořené instance uživatel a seznamu jeho rolí je následně dle konfigurace vyhodnoceno, zda má uživatel oprávnění k přístupu k danému API endpointu (autorizace).

3.3.5 Objednávky

Pro objednávky byl autorem této práce vypracován pouze návrh a na implementaci se podílel spíše z pozice zadavatele než programátora. Implementace návrhu byla svěřena několika studentům bakalářského studia na ČVUT FIT v rámci předmětu BI–SP1. Tito studenti v rámci zmíněného předmětu pracovali pod vedením Ing. Jiřího Hunky a Ing. Jana Matouška na novém systému pro administraci. Nepodíleli se však na vývoji částí systému jako jsou výrobci, kategorie a jiné části, které byly v této práci popsány. Jejich zaměření bylo spíše na okrajové části systému. Skupina studentů byla rozdělena na dva týmy. První tým byl zaměřený na frontend a druhý tým na backend. Jelikož dříve popsaný návrh je pro backend, tak této implementace se účastnil backend tým. Jednalo se o studenty:

- Vojtěch Beneš,
- David Mareš.

Tabulka 3.4 Implementace API endpointů pro objednávky

#	URI	Metoda	Stav
1	/orders	GET	Neobsahuje všechny požadované informace (chybí například `profit` nebo `paymentMethod`).
2	/orders/{orderId}	GET	Neobsahuje všechny požadované informace (chybí například `profit` nebo `inStock`).
3	/orders/{orderId}/credit-notes	GET	V implementaci není vrácen objekt obsahující kolekci dobropisů a kolekci produktů (obsažených v dobropisech). Místo toho je kolekce produktů součástí objektu pro dobropis.
5	/orders/{orderId}/addresses	PUT	✓
12	/order-statuses	GET	✓
13	/payment-methods	GET	✓
14	/shipping-methods	GET	✓

Za účelem zahájení spolupráce, představení návrhu a diskuzi nad vypracovaným návrhem se uskutečnily celkem tři schůzky, kde byli přítomni jmenovaní studenti, Ing. Jiří Hunka, Ing. Jan Matoušek a Bc. Tomáš Hojek (autor této práce). Díky těmto schůzkám bylo z návrhu odstraněno několik nedostatků, které by byly jistě objeveny později při implementaci. Zmínění studenti připravili migrace pro úpravu datového modelu a částečně implementovali několik API endpointů. V rámci implementace objednávek tedy nebylo implementováno vše, co bylo ve funkčních požadavcích specifikováno a co bylo popsáno ve vypracovaném návrhu.

Přehled implementovaných endpointů je zobrazen v tabulce 3.4. Jednotlivé API endpointy obsahují pořadová čísla, která jim byla přidělena v dřívější tabulce 2.22 uvedené v návrhu objednávek. Dále tabulka pro přehlednost obsahuje URI endpointů a HTTP metody. V posledním sloupci je popsán stav implementace (znak ✓ označuje přístupové body, které jsou kompletní a měly by poskytovat všechny potřebné informace).

Kapitola 4
Testování

Tato kapitola bude věnována testování vytvořeného prototypu aplikace. Co znamená testování aplikace? Jedná se o proces, který může provádět člověk nebo skript, nástroj či jakýkoliv testovací framework. Ve všech zmíněných možnostech je účelem nalezení chyb a jejich následná oprava. Díky tomu je možné dodávat robustní software s minimem chyb. [50]

Rozlišují se dva základní typy testování, které nyní budou popsány.

Manuální testování je používáno především v raných fázích vývoje, kdy jsou testovány právě vyvíjené funkce. Jedná se o velmi specifické testování, které je obvykle velmi úzce zaměřené. Tento druh testování může být prováděn i ve větším měřítku, ale tato možnost již vyžaduje testovací tým (především pokud by se mělo jednat o periodickou a pravidelnou činnost). [50]

Automatizované testování je používáno pro otestování vyvinuté aplikace (či funkce). Testuje se použitelnost (například u frontend aplikace), funkcionalita a může se testovat i výkon. Cílem by mělo být efektivně otestovat velkou část aplikace a odhalit co nejvíce nedostatků a chyb. [50]

Dle [51] existuje několik doporučení pro správné a kvalitní testování:

- průběžné testování vyvíjené aplikace,
- konzultovat zadání – v případě této práce byly vždy zadání i návrh konzultovány s kolegou Aloisem Koubou a také s druhým týmem, který vyvíjel klientskou aplikaci, tedy Martinem Dvořákem a Janem Babákem,
- používat manuální i automatizované testování,
- možnost použití metodologie zvané Programování řízené testy (anglicky *Test driven development*) – v tomto případě jsou nejprve napsány testy, a teprve poté se implementuje daná funkce (tato metodologie nebyla v rámci implementace této práce používána).

4.1 Manuální testování

Během vývoje byla vyvíjená aplikace testována manuálně, a to jednak autorem této práce, ale i kolegou Aloisem Koubou. Dále byla aplikace testována kolegy Martinem Dvořákem a Janem Babákem, kteří používali vytvořená rozhraní pro klientskou aplikaci. Díky této spolupráci a průběžnému testování byla objevena řada chyb, které mohly být velmi rychle opraveny.

Při testování serverové aplikace na základě volání z klientské aplikace objevil kolega Martin Dvořák nedostatek týkající se Cross-origin resource sharing. Tento nedostatek je uveden zde, neboť se jednalo o problém nalezený při integraci.

Cross-Origin Resource Sharing (zkráceně CORS) je mechanismus založený na HTTP hlavičkách. Tento mechanismus byl zaveden kvůli bezpečnosti a umožňuje serveru rozpoznat jakékoliv jiné origin, než které jsou povoleny. Origin je pojem, který je definován protokolem, doménovým jménem (anglicky *hostname*) a portem dané URL adresy. [52][53]

Zdroj [53] uvádí jako příklad stejných origin například následující dvojici:

- `http://example.com/app1/index.html`
- `http://example.com/app2/index.html`

Adresy mají stejný protokol (`http`), doménové jméno (`example.com`) a port, který není uveden (výchozí port je 80).

Pokud by adresy v příkladu nebyly ze stejné origin, tak se aplikují pravidla CORS dle nastavení serveru.

V první testované implementaci nebyla CORS politika nastavena, neboť autor této práce se s CORS ve svém profesním životě ještě nesetkal. Jako řešení byla přidána knihovna *nelmio/NelmioCorsBundle* [54] s příslušnou konfigurací.

4.2 Automatizované testování

Systém je pokryt několika druhy testů. Základním prvkem testování je statická analýza kódu. Další druhy testování se zaměřují na správné chování aplikace, které jsou již časově náročnější, ale o to cennější.

4.2.1 Statická analýza kódu

Statická analýza kódu označuje techniky a nástroje, které dokáží odhalit možné chyby bez spuštění dané aplikace. Jedná se tedy o techniky hledání chyb ve zdrojovém kódu, a proto je možné tuto formu označit jako white box testování (testování se znalostí zdrojového kódu). [55]

Dle [55] se pomocí statické analýzy často detekují:

- výkonnostní problémy,
- bezpečnostní chyby,
- nedodržení standardů či norem,
- použití již zastaralých programovacích technik (může zahrnovat například knihovní funkce, které jsou již označeny jako zastaralé a neměly by se proto používat).

Pro statickou analýzu kódu byl zvolen nástroj PHPStan [56]. Jedná se o nástroj, který provede analýzu kódu, aniž by byl kód spuštěn. Jedná se tedy o klasickou statickou analýzu kódu, kdy jsou kontrolována pravidla vycházející se syntaxe jazyka PHP. PHPStan dle [57] kontroluje mnoho různých situací. V následujícím výčtu jich pro ukázku bude několik uvedeno.

- Provádí kontrolu existence tříd použitých v kódu, například konstruktech `instanceof` či `catch`.
- Provádí kontrolu existence volaných metod a funkcí, včetně kontroly jejich viditelnosti. Pokud metodu není možné volat mimo třídu, kde byla definována, tak při zavolání z venčí PHPStan upozorní na chybu.

- Kontroluje, zda metoda vrací stejný typ, který deklaruje, že vrací.
- Kontroluje existenci proměnných v závislosti na podmínkách, cyklech apod.

Nastavení PHPStan se provádí v konfiguračním souboru `phpstan.neon`. V tomto souboru se definují různé parametry, například:

- cesty k souborům, které se mají kontrolovat,
- úroveň kontroly (PHPStan umožňuje nastavit úroveň kontroly, více lze najít v příslušné dokumentaci [58]),
- chyby a soubory, které mají být ignorovány.

Pro tuto práci autor nastavil statickou analýzu pro všechny zdrojové kódy a testy. Úroveň kontroly byla nastavena na maximum.

4.2.2 Jednotkové testy

Jednotkové testy mají zkontrolovat funkčnost vybrané části kódu nezávisle na okolním prostředí. Obvykle je takto zkontrolována jedna třída či jedna funkce (v rámci jednoho testu). Tento typ testů umožňuje rychlou kontrolu malých dílčích funkcí systému.

Pro tyto byla zvolena knihovna PHPUnit [59], která je doporučena i frameworkem Symfony v oficiální dokumentaci [60]. Dle [60] se rozlišují tři základní typy testů:

Unit tests (jednotkové testy) testují očekávané chování velmi malého úseku zdrojového kódu (například třídy).

Integration tests (integrační testy) ověřují funkčnost větších celků. V rámci těchto testů se kontroluje především business logika, která obvykle zahrnuje komunikaci několika tříd a komunikaci s nižšími vrstvami aplikace (databází).

Application tests (aplikační či systémové testy) kontrolují chování celé aplikace. To znamená, že se vytvoří HTTP request a kontroluje se přijatá odpověď. Tento typ testů nebyl v rámci této práce řešený a je zde pouze zmíněn jako doporučení pro budoucí použití v aplikaci.

V rámci vyvíjené aplikace byly napsány Unit testy pro ověření funkčnosti transportních objektů (TO). V rámci těchto testů se testuje například metoda `buildFromDO`, která byla popsána již v návrhu. Díky tomu existuje kontrola, že konverze dat z databáze do transportního objektu probíhá korektně. Dále byly průběžně přidávány integrační testy pro kontrolu vyvíjených funkcionalit. Jelikož jsou tyto testy celkem dlouhé a mají vždy dlouhou úvodní pasáž, kde je nutné inicializovat data pro testování, tak zde nebudou uvedeny.

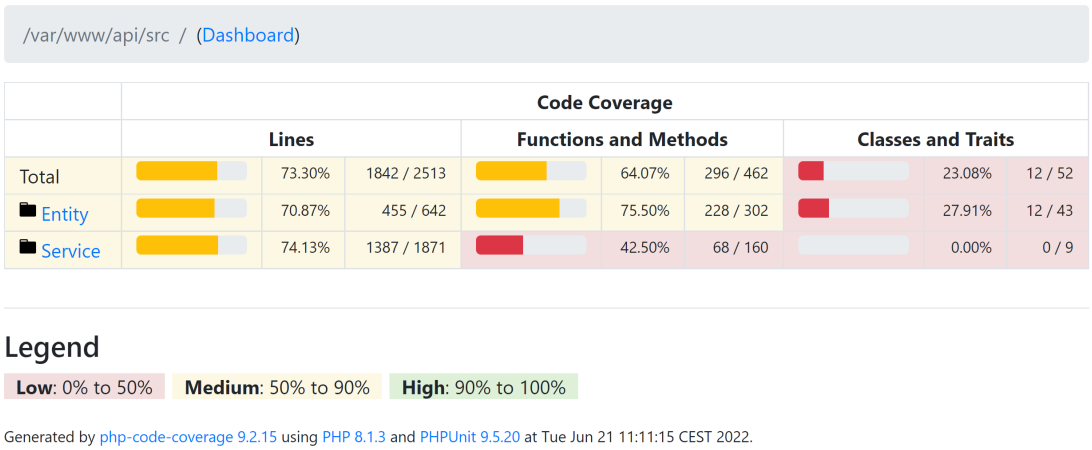
Ve světě vývoje softwaru však existuje pojem *code coverage*. Tento pojem označuje míru, která se používá k popisu pokrytí zdrojového kódu testy. Jinak řečeno, tento pojem říká, do jaké míry je funkcionalita systému testována pomocí výše jmenovaných testů. Samozřejmě zde platí jednoduchá přímá úměra, čím větší je pokrytí testy, tím je program více otestován a je menší šance, že bude obsahovat chybu. [61]

Dle [61] existuje několik metrik pro určení míry pokrytí zdrojového kódu. V tomto textu bude uvedeno pouze několik z nich, neboť tyto metriky budou později použity pro prezentaci míry pokrytí testy v rámci této práce.

Line Coverage je velice jednoduchá metrika, která měří, zda každá řádka zdrojového kódu byla pokryta nějakým testem.

Function and Method Coverage je metrika, která sleduje, zda každá funkce (či metoda) byla zavolána nějakým testem.

Obrázek 4.1 Code coverage – všeobecný přehled



Class and Trait Coverage je metrika, která sleduje, zda každá metoda nějaké třídy je testována nějakým testem.

Pro měření v rámci této práce byla použita knihovna *sebastianbergmann/phpunit*, která pro tuto funkcionalitu využívá knihovnu *sebastianbergmann/php-code-coverage*. Pro metriku *Function and Method Coverage* se započítávají pouze ty funkce a metody, které mají pomocí testů pokryty všechny řádky zdrojového kódu. Pro metriku *Class and Trait Coverage* platí podobný princip. Třída je označena jako pokrytá testy, pokud jsou všechny její funkce a metody označeny jako pokryté. Tato poslední metrika je tedy velmi přísná, neboť vyžaduje pokrytí každého řádku zdrojového kódu v rámci třídy.

Výsledkem měření code coverage je takzvaný *report* (zpráva). V rámci této práce byl vygenerován report pro navržené a implementované části systému (s výjimkou objednávek, které nebyly dokončeny a pro které jednotkové testy neexistují). Celý vygenerovaný report je možno nalézt na přiloženém médiu ve formátu HTML. Pro ukázku zde budou uvedeny dva obrázky z tohoto reportu. Prvním je všeobecný přehled, kde je možno vidět rozdělení na Unit testy (nazváno jako Entity) a Integration testy (nazváno jako Service), obrázek 4.1. Druhým obrázkem je přehled pokrytí zdrojového kódu pro Service třídy, obrázek 4.2.

Obrázek 4.2 Code coverage pro Service třídy

/var/www/api/src / Service / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		74.13%	1387 / 1871		42.50%	68 / 160		0.00%	0 / 9
CategoryBannerService.php		84.87%	101 / 119		37.50%	3 / 8		0.00%	0 / 1
CategoryService.php		76.26%	408 / 535		37.93%	11 / 29		0.00%	0 / 1
DomainService.php		50.88%	58 / 114		42.86%	9 / 21		0.00%	0 / 1
ExternalSourceService.php		68.99%	89 / 129		33.33%	5 / 15		0.00%	0 / 1
FeatureService.php		84.92%	152 / 179		50.00%	6 / 12		0.00%	0 / 1
FileManagerService.php		84.64%	303 / 358		57.14%	20 / 35		0.00%	0 / 1
ManufacturerService.php		49.64%	136 / 274		17.39%	4 / 23		0.00%	0 / 1
OrderService.php		96.43%	27 / 28		66.67%	2 / 3		0.00%	0 / 1
UrlSlugService.php		83.70%	113 / 135		57.14%	8 / 14		0.00%	0 / 1

Legend

Low: 0% to 50% **Medium**: 50% to 90% **High**: 90% to 100%

Generated by php-code-coverage 9.2.15 using PHP 8.1.3 and PHPUnit 9.5.20 at Tue Jun 21 11:11:15 CEST 2022.

Kapitola 5
Závěr

Cílem této práce bylo navrhnout a vyvinout základ nového systému pro administraci e-shopů, který by zohledňoval hlavní funkcionality stávajícího systému postaveného na platformě OpenCart. Hlavním úkolem bylo vyvinou systém, který nebude závislý na zmíněné platformě nebo jakékoliv jiné platformě podobného zaměření.

Tento cíl byl splněn. Nejdříve bylo analyzováno současné řešení, kterým je aplikace pro administraci e-shopů od společnosti Jagu, s. r. o. Tato analýza probíhala za významné pomoci Ing. Jana Matouška, který současnou administraci velmi dobře zná. Díky tomu bylo snadnější a rychlejší proniknout do tajů současné administrace. Na základě této analýzy vznikl seznam funkčních a nefunkčních požadavků. Tento seznam byl dále doplněn o požadavky vycházející ze zadání této práce.

Na základě požadavků byl vypracován návrh pro nový systém administrace. Návrh se kromě samotného systému pro administraci zabýval také autorizačním serverem a současnou databází, nad kterou měl být vystavěn nový systém. Autorizační server nebyl v rámci práce vyvíjen, nýbrž bylo použito již existující řešení v podobě autorizačního serveru Keycloak. Všechny návrhy byly průběžně konzultovány s kolegy, kteří se na vývoji nového systému taktéž podíleli. Klientskou část aplikace (frontend) navrhovali a vyvíjeli Martin Dvořák a Jan Babák. Serverovou část aplikace (backend) navrhoval a vyvíjel autor této práce ve spolupráci s Aloisem Koubou. Návrhy byly rovněž konzultovány s Ing. Jiřím Hunkou a Ing. Janem Matouškem.

Provedené návrhy byly následně implementovány. Jako první byl realizován návrh týkající se výrobců, který implementoval autor této práce ve spolupráci s kolegou Aloisem Koubou. Následovala implementace kategorií, na které se rovněž podíleli autor této práce a Alois Kouba, který u kategorií dle návrhu implementoval parametry, bannery a filtry. Následovala implementace podpůrné funkcionality týkající se URL slug a implementace správce souborů a externích kategorií, přičemž zpracovat tato témata bylo již úkolem autora této práce.

Jako poslední přišla na řadu implementace návrhu objednávek, která neproběhla v plném rozsahu a byla implementována jen část navržené funkcionality. Tato poslední část práce byla netradiční, neboť implementace byla svěřena dvěma studentům bakalářského studia v rámci předmětu BI–SP1. Autor této práce si tak v malém měřítku mohl vyzkoušet pozici vedoucího týmu, která představovala seznámení s vypracovaným návrhem a následné sledování samotné implementace (code review).

Posledním tématem této práce bylo testování, které bylo prováděno průběžně spolu s implementací. Nalezené chyby byly průběžně opravovány a občas byly opravovány i nedostatky v návrhu, které se objevily při integraci s klientskou aplikací. Některé z chyb a úprav byly pro svou významnost popsány v dřívějších kapitolách této práce.

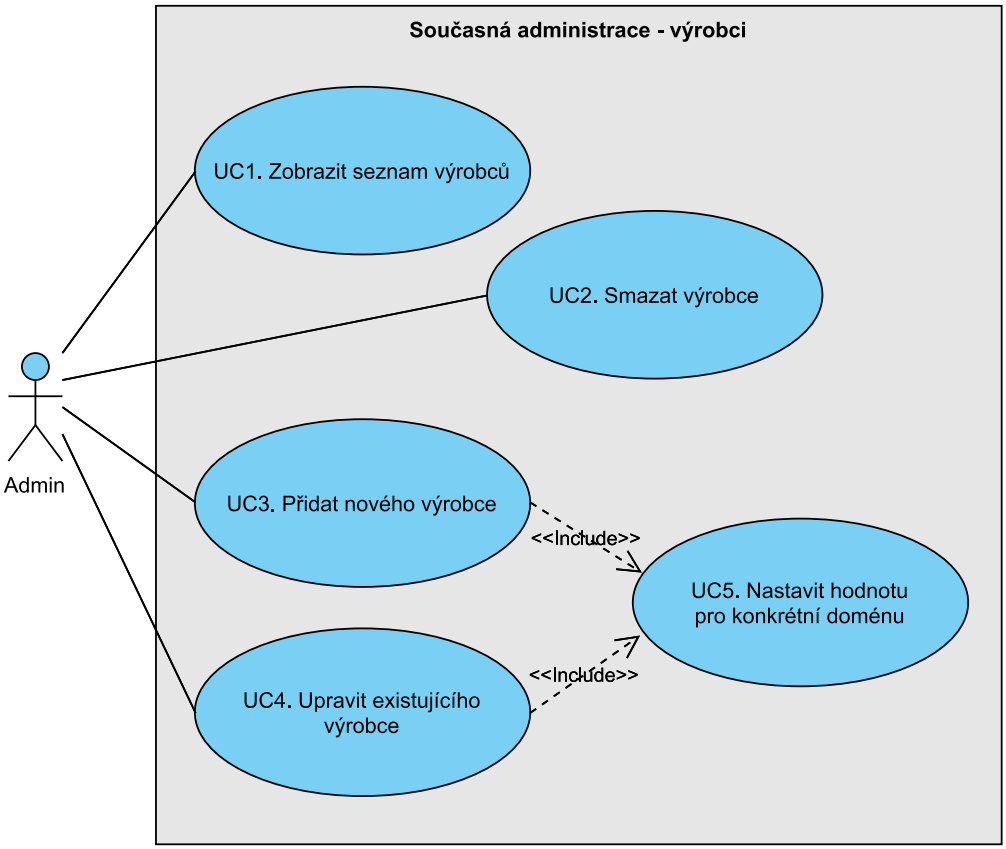
Úkolem do budoucna je jistě dokončení implementace návrhu týkajícího se objednávek. Systém rovněž nebyl nasazen, což by bylo možno provést pomocí nástroje Deployer, který se používá

pro nasazování PHP aplikací. Vyvinutý systém pracuje s výrobci, kategoriemi, produkty a částečně objednávkami. Jedná se tedy o základ, který bude jistě dále rozšiřitelný tak, aby pokrýval všechny funkčnosti, které nabízí současná administrace pro e-shopy. Vzdálenější budoucností je úprava databáze, která není navržena ideálně. Tato úprava povede i ke značným úpravám v nově vyvinutém systému, avšak díky navrženému a implementovanému rozdělení entit uvnitř systému (na entity z databáze a entity používané pro komunikaci s vnějším světem) bude jistě tento krok snazší.

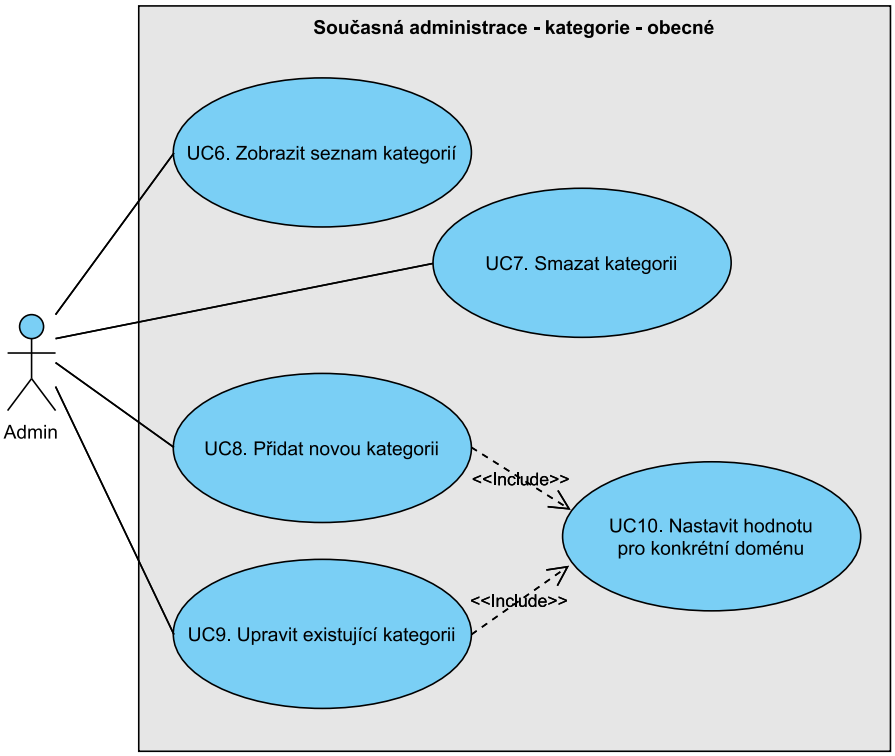
Příloha A

Diagramy případů užití

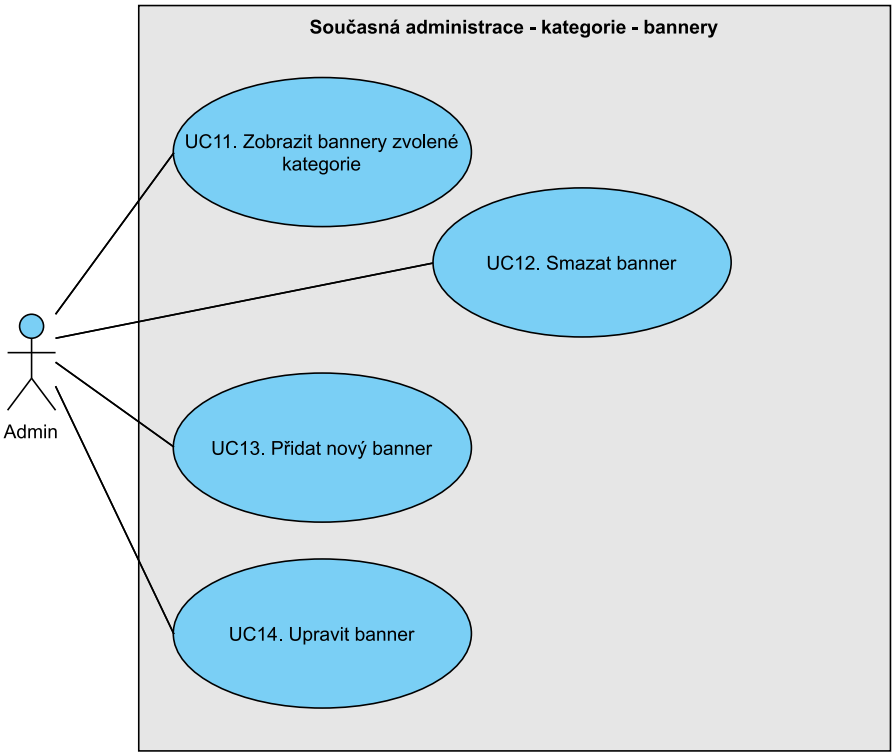
Obrázek A.1 Diagram případů užití pro výrobce.



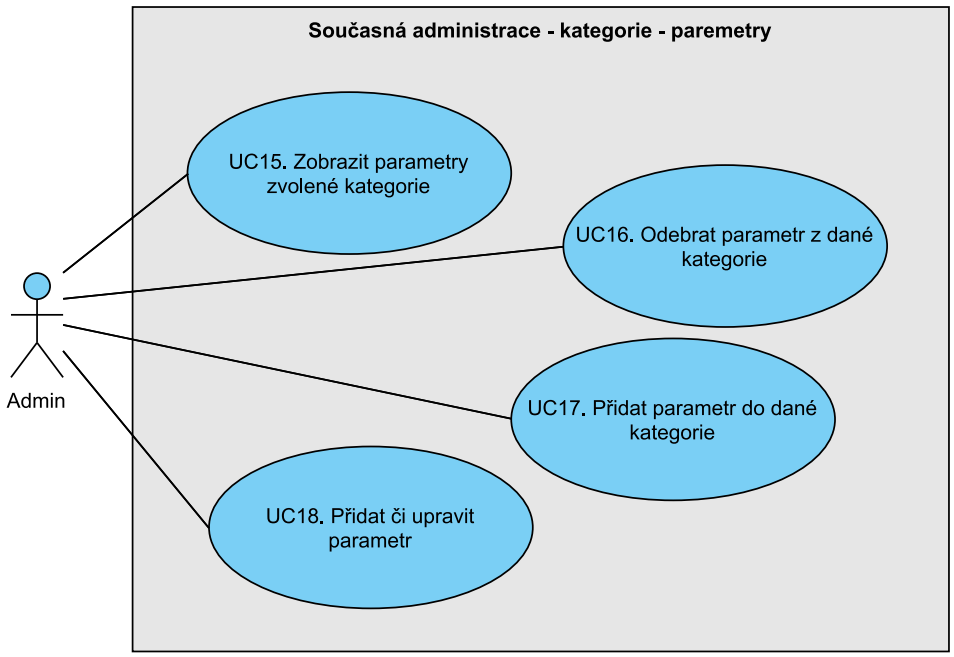
Obrázek A.2 Diagram případů užití pro kategorie (obecné)



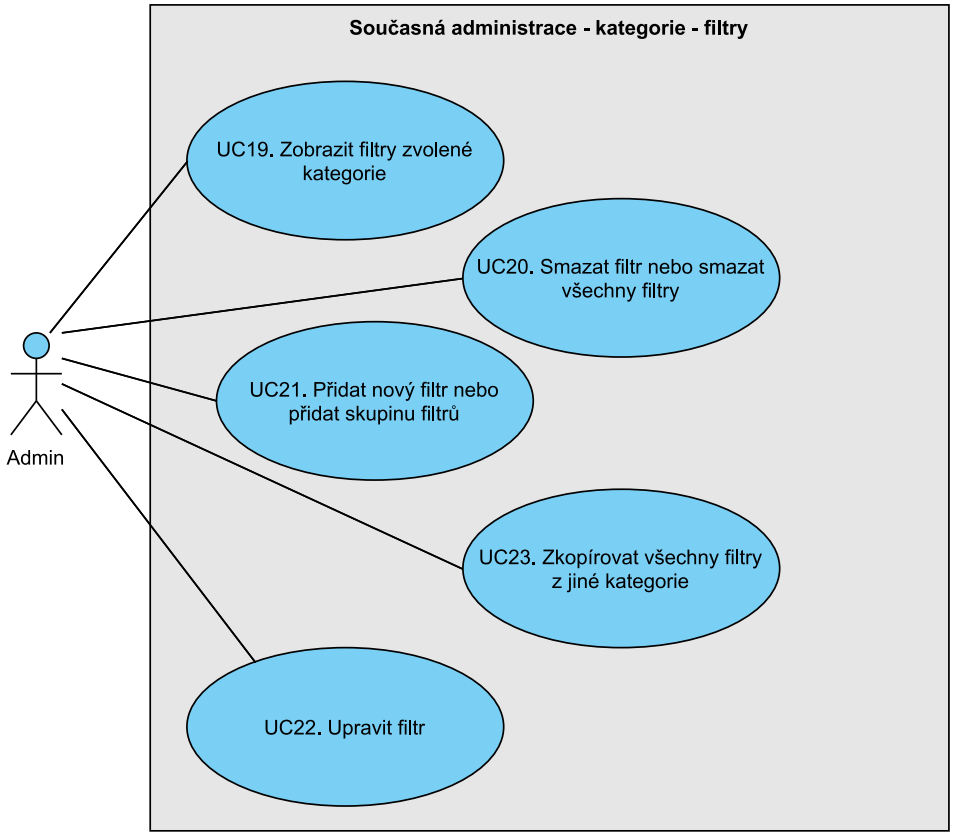
Obrázek A.3 Diagram případů užití pro kategorie (bannery)



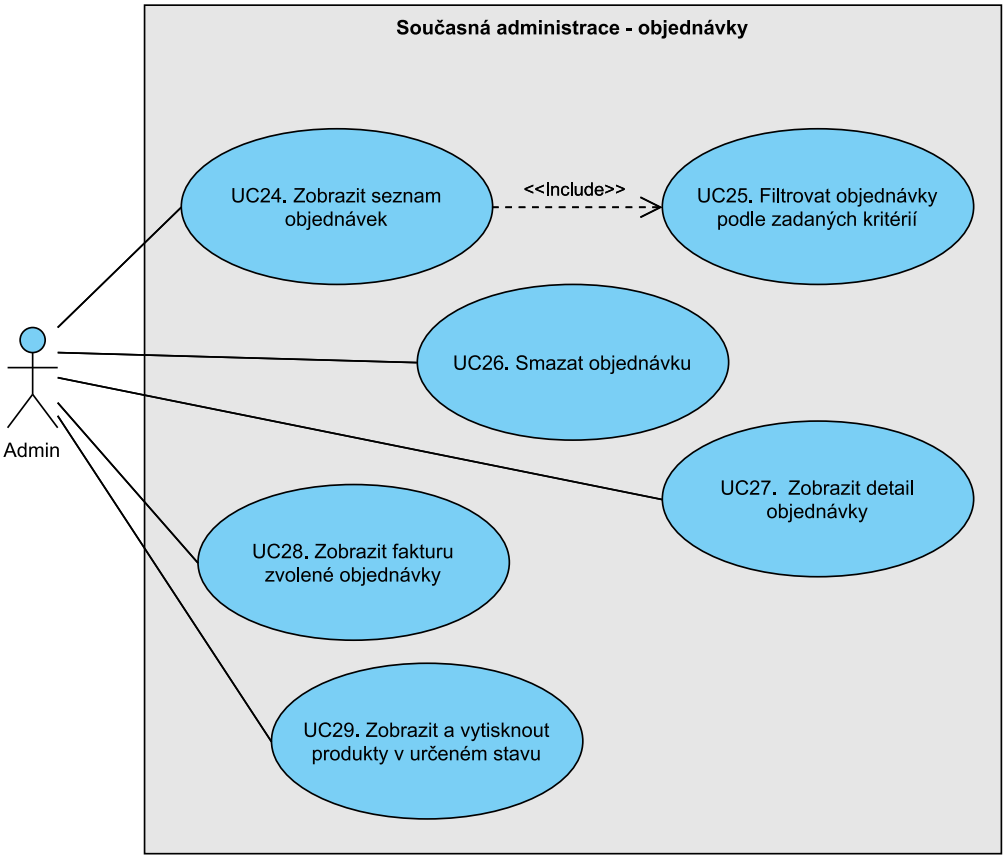
Obrázek A.4 Diagram případů užití pro kategorie (parametry)



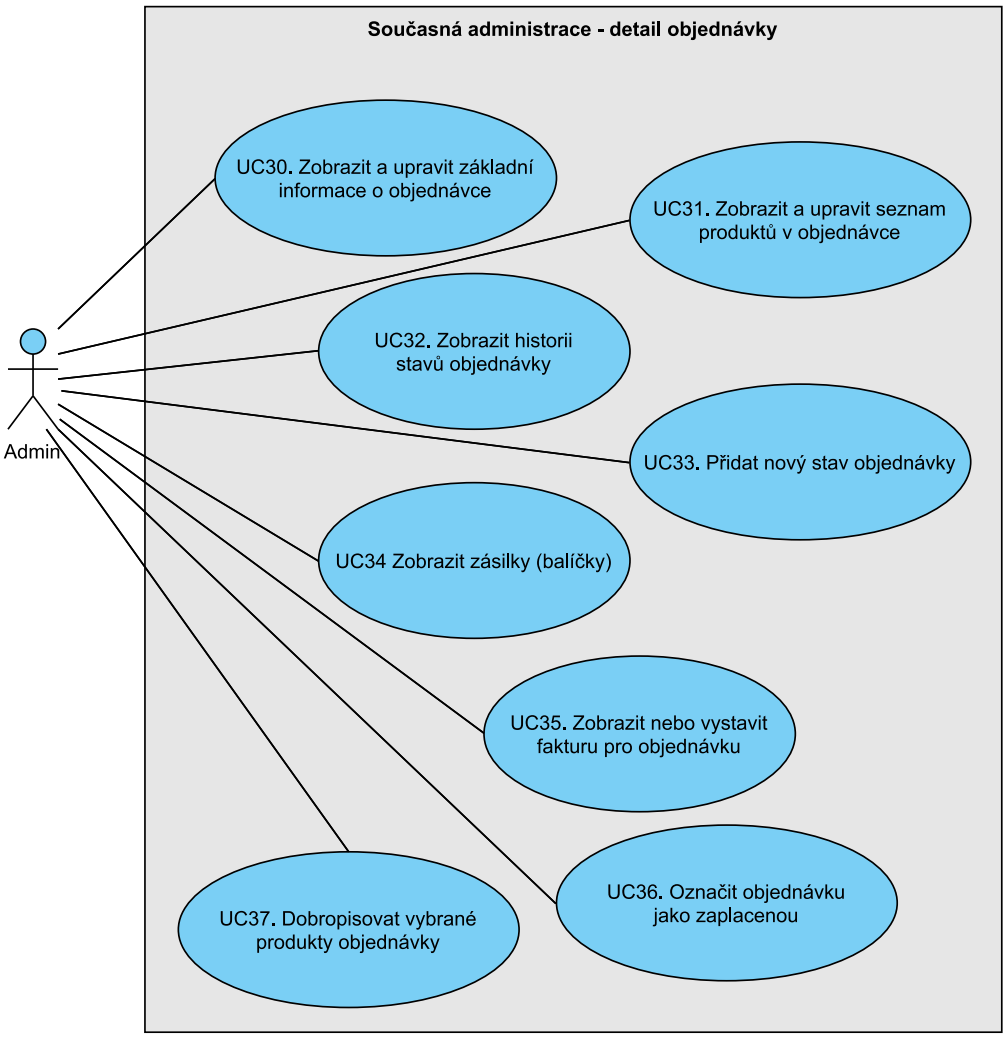
Obrázek A.5 Diagram případů užití pro kategorie (filtry)



Obrázek A.6 Diagram případů užití pro objednávky.



Obrázek A.7 Diagram případů užití pro detail objednávky.



Příloha B

Docker

```
version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Výpis kódu 1 Příklad souboru *docker-compose.yaml* z oficiální dokumentace pro Docker [41].

Příloha C

Aktualizace externích kategorií

V této příloze je uveden výpis kódu pro příkaz, který spustí aktualizaci externích kategorií. Tento výpis kódu byl z důvodu čitelnosti upraven, změny jsou převážně formátovací povahy. Popis a vysvětlení tohoto kódu je k nalezení v příslušné kapitole (kapitole řešící realizaci navrženého systému).

```php
<?php declare(strict_types=1);

namespace App\Command;

use App\Service\ExternalSourceService;
use Symfony\Component\Console\Attribute\AsCommand;
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Throwable;
use Zenstruck\ScheduleBundle\Schedule\SelfSchedulingCommand;
use Zenstruck\ScheduleBundle\Schedule\Task\CommandTask;

#[AsCommand(
    name: 'app:update-external-categories',
    description: 'The command to update external categories.',
    hidden: false
)]
class UpdateExternalCategoriesCommand extends Command
    implements SelfSchedulingCommand
{
    private ExternalSourceService $externalSourceService;

    public function __construct(ExternalSourceService $externalSourceService)
    {
        $this->externalSourceService = $externalSourceService;

        parent::__construct();
    }

    protected function execute(InputInterface $input,
                               OutputInterface $output): int
    {
        $this->externalSourceService->updateExternalCategories();

        $output->writeln('Updating of external categories is already done.');
        return Command::SUCCESS;
    }

    public function schedule(CommandTask $task): void
    {
        // update will be every monday at 1:00
        $task->mondays()->at("1:00");
    }
}
```

Výpis kódu 2 Implementace příkazu pro automatickou aktualizaci externích kategorií

Příloha D

Konfigurační soubor pro autorizaci

```
security:
    enable_authenticator_manager: true
    providers:
        keycloak_provider:
            id: App\Security\KeycloakUserProvider
    firewalls:
        main:
            stateless: true
            custom_authenticators:
                - App\Security\KeycloakAuthenticator

    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
        # example with admin role with higher priority
        # - { path: ^/api/manufacturer, roles: ROLE_ADMIN }
        - { path: ^/api, roles: ROLE_USER }

    role_hierarchy:
        # following rule means that admin has also rights of the user
        ROLE_ADMIN: ROLE_USER

        # example: Users with following ROLE_SUPER_ADMIN, will automatically
        #          have ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH and
        #          ROLE_USER (inherited from ROLE_ADMIN).
        # ROLE_SUPER_ADMIN: [ ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH ]
```

Výpis kódu 3 Ukázka konfiguračního souboru pro autentizaci a autorizaci

Příloha E

Konfigurační skripty pro GitLab CI

```
#!/usr/bin/env bash

echo "Migrations are running now!"
php bin/console doctrine:migrations:migrate --env=test --no-interaction

echo "Migrations are done. Running tests."
php vendor/phpunit/phpunit/phpunit --testdox --configuration phpunit.xml.dist
```

Výpis kódu 4 Ukázka skriptu php_test_run.sh volaného z .gitlab-ci.yml

```
default:
  image: php:8.1.3-fpm

variables:
  # definition of variables for database
  # ...

before_script:
  # installation of some necessary libraries (eg. composer)
  # ...
  - echo "Initialization done"

stages:
  - test

cache:
  key: $CI_COMMIT_REF_SLUG
  paths:
    - .tmp/
    - vendor/

tests:
  stage: test
  services:
    - name: mysql:8.0
      alias: localdb
  script:
    # installation of mysql client and initialization of DB
    # ...
    - echo "Database is created."
    - sh php_test_run.sh

static_analysis:
  stage: test
  script:
    - vendor/bin/phpstan analyse -c phpstan.neon --memory-limit 512M
      --no-progress --error-format gitlab | tee phpstan.json
  artifacts:
    when: always
    reports:
      codequality: phpstan.json
```

Výpis kódu 5 Ukázka konfiguračního souboru .gitlab-ci.yml

Bibliografie

1. OLIC, Aleksandar. Waterfall Project Management Methodology. In: *Project Management* [online]. ActiveCollab, 2017 [cit. 2022-05-28]. Dostupné z: https://activecollab.com/blog/project-management/waterfall-project-management-methodology.
2. LAOYAN, Sarah. Everything you need to know about waterfall project management. In: *Asana* [online]. Asana, 2021 [cit. 2022-05-28]. Dostupné z: https://asana.com/resources/waterfall-project-management-methodology.
3. *SDLC - Iterative Model* [online]. Tutorials Point, 2022 [cit. 2022-05-29]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm.
4. *SDLC Iterative Model* [online]. TutorialsCampus, 2021 [cit. 2022-05-29]. Dostupné z: https://www.tutorialscampus.com/sdlc/img/iterative-model.png.
5. *SDLC Iterative Model* [online]. TutorialsCampus, 2021 [cit. 2022-05-29]. Dostupné z: https://www.tutorialscampus.com/sdlc/iterative-model.htm.
6. *SDLC - Agile Model* [online]. Tutorials Point, 2022 [cit. 2022-05-30]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm.
7. *SDLC Agile Model* [online]. TutorialsCampus, 2021 [cit. 2022-05-30]. Dostupné z: https://www.tutorialscampus.com/sdlc/agile-model.htm.
8. *SDLC Agile Model* [online]. TutorialsCampus, 2021 [cit. 2022-05-30]. Dostupné z: https://www.tutorialscampus.com/sdlc/img/agile-model-with-multiple-iterations.png.
9. LITHMEE. What is the Difference Between Centralized and Distributed Version Control. In: *Pediaa* [online]. Pediaa, 2019 [cit. 2022-05-30]. Dostupné z: https://pediaa.com/what-is-the-difference-between-centralized-and-distributed-version-control/#Centralized%5C%20Version%5C%20Control.
10. *Merge-based vs Lock-based Version Control* [online]. Free Software Foundation, Inc., 2022 [cit. 2022-06-01]. Dostupné z: https://www.gnu.org/software/emacs/manual/html_node/emacs/VCS-Merging.html.
11. *Introduction* [online]. OpenCart, 2016 [cit. 2022-05-20]. Dostupné z: http://docs.opencart.com/en-gb/introduction/.
12. *System / Server requirements* [online]. OpenCart, 2016 [cit. 2022-05-22]. Dostupné z: http://docs.opencart.com/en-gb/requirements/.
13. *Developing Modules* [online]. OpenCart, 2016 [cit. 2022-05-24]. Dostupné z: http://docs.opencart.com/en-gb/developer/module/.
14. *MVC-L* [online]. OpenCart, 2016 [cit. 2022-05-24]. Dostupné z: http://docs.opencart.com/image/mvcl.png. Obrázek z dokumentace OpenCart.

15. *Jagu s.r.o., IČO: 28426126 - Obchodní rejstřík* [online]. peníze.cz, 2022 [cit. 2022-05-22]. Dostupné z: https://rejstrik.penize.cz/28426126-jagu-s-r-o.

16. NOVÁČEK, Tomáš. *Migrace customizovaných e-shopu OpenCart 1.1*. Praha, 2018. Diplomová práce. ČVUT v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.

17. SANDERS, Harry. *What Is a Slug? URL Slugs and Why They Matter for SEO* [online]. Semrush Blog, 2021 [cit. 2022-05-25]. Dostupné z: https://www.semrush.com/blog/what-is-a-url-slug/.

18. RYDVAL, Slávek. *Případy užití (Use Cases)* [online]. Slávek Rydval a OMG, 2011 [cit. 2022-06-13]. Dostupné z: http://ocup.ocup.cz/2010/07/pripady-uziti-use-cases.html.

19. *Functional and Nonfunctional Requirements: Specification and Types* [online]. Altexsoft, 2021 [cit. 2022-06-02]. Dostupné z: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/.

20. DYSON, Jonathan. *Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements* [online]. Linkedin, 2019 [cit. 2022-06-16]. Dostupné z: https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson.

21. *What is Symfony* [online]. Symfony, 2022 [cit. 2022-06-03]. Dostupné z: https://symfony.com/what-is-symfony.

22. *Symfony versus Flat PHP* [online]. Symfony, 2022 [cit. 2022-06-05]. Dostupné z: https://symfony.com/doc/current/introduction/from_flat_php_to_symfony.html#symfony-versus-flat-php.

23. *Chapter 2 - Exploring Symfony's Code* [online]. Symfony, 2022 [cit. 2022-06-05]. Dostupné z: https://symfony.com/legacy/doc/gentle-introduction/1_4/en/02-Exploring-Symfony-s-Code.

24. RAICHL, Šimon. Lekce 7 - Formát JSON. In: *JavaScript - Online kurzy programování a největší český e-learning* [online]. ITnetwork, 2022 [cit. 2022-06-08]. Dostupné z: https://www.itnetwork.cz/javascript/oop/objekty-json-a-vylepseni-diare-v-javascriptu.

25. *Databases and the Doctrine ORM* [online]. Symfony, 2022 [cit. 2022-06-05]. Dostupné z: https://symfony.com/doc/current/doctrine.html.

26. *OpenAPI Specification* [online]. Swagger, 2022 [cit. 2022-06-16]. Dostupné z: https://swagger.io/specification/.

27. *Server Administration Guide: Keycloak features and concepts* [online]. Keycloak, 2022 [cit. 2022-06-10]. Dostupné z: https://www.keycloak.org/docs/latest/server_admin/index.html.

28. *Server Administration Guide: Features* [online]. Keycloak, 2022 [cit. 2022-06-10]. Dostupné z: https://www.keycloak.org/docs/latest/server_admin/index.html.

29. *Bearer Authentication* [online]. SmartBear, 2022 [cit. 2022-06-10]. Dostupné z: https://swagger.io/docs/specification/authentication/bearer-authentication/.

30. PARECKI, Aaron. Token Introspection Endpoint. In: *OAuth 2.0 Simplified* [online]. Okta, 2022 [cit. 2022-06-10]. Dostupné z: https://www.oauth.com/oauth2-servers/token-introspection-endpoint/.

31. ABEL, Marcos. Validate JWT tokens using JWKS in Java. In: *Trabe* [online]. Medium, 2020 [cit. 2022-06-10]. Dostupné z: https://medium.com/trabe/validate-jwt-tokens-using-jwks-in-java-214f7014b5cf.

32. *PHP: What is PHP?* [Online]. php.net, 2022 [cit. 2022-06-11]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php.

33. *PHP: What do I need?* [Online]. php.net, 2022 [cit. 2022-06-11]. Dostupné z: https://www.php.net/manual/en/tutorial.requirements.php.
34. *PHP: History of PHP* [online]. php.net, 2022 [cit. 2022-06-11]. Dostupné z: https://www.php.net/manual/en/history.php.php.
35. *Symfony at a Glance* [online]. Symfony, 2022 [cit. 2022-06-11]. Dostupné z: https://symfony.com/at-a-glance.
36. *Symfony Contributors* [online]. Symfony, 2022 [cit. 2022-06-11]. Dostupné z: https://symfony.com/contributors.
37. *What is MySQL? Everything You Need to Know* [online]. Talend [cit. 2022-06-11]. Dostupné z: https://www.talend.com/resources/what-is-mysql/.
38. *Getting Started with Doctrine* [online]. Doctrine [cit. 2022-06-11]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/stable/tutorials/getting-started.html.
39. *nginx* [online]. Nginx, Inc. [cit. 2022-06-11]. Dostupné z: http://nginx.org/en/.
40. *Docker overview* [online]. Docker Inc., 2021 [cit. 2022-06-11]. Dostupné z: https://docs.docker.com/get-started/overview/.
41. *Overview of Docker Compose* [online]. Docker Inc., 2021 [cit. 2022-06-14]. Dostupné z: https://docs.docker.com/compose/.
42. REHKOPF, Max. What is continuous integration? In: *Software Development* [online]. Atlassian, 2022 [cit. 2022-06-16]. Dostupné z: https://www.atlassian.com/continuous-delivery/continuous-integration.
43. HALL, Tom. DevOps Best Practices. In: *Software Development* [online]. Atlassian, 2022 [cit. 2022-06-16]. Dostupné z: https://www.atlassian.com/devops/what-is-devops/devops-best-practices.
44. *CI/CD concepts* [online]. GitLab, 2022 [cit. 2022-06-16]. Dostupné z: https://docs.gitlab.com/15.0/ee/ci/introduction/.
45. *Console Commands* [online]. Symfony, 2022 [cit. 2022-06-15]. Dostupné z: https://symfony.com/doc/current/console.html.
46. *Defining the Schedule: Self-Scheduling Commands* [online]. zenstruck/schedule-bundle, 2022 [cit. 2022-06-15]. Dostupné z: https://github.com/zenstruck/schedule-bundle/blob/HEAD/doc/define-schedule.md#self-scheduling-commands.
47. *zenstruck/schedule-bundle: Schedule Cron jobs (commands/callbacks/bash scripts) within your Symfony application.* [Online]. Kevin Bond, 2022 [cit. 2022-06-15]. Dostupné z: https://github.com/zenstruck/schedule-bundle#quick-start.
48. *CronJob* [online]. Seobility [cit. 2022-06-16]. Dostupné z: https://www.seobility.net/en/wiki/CronJob.
49. *Security* [online]. Symfony, 2022 [cit. 2022-06-16]. Dostupné z: https://symfony.com/doc/current/security.html.
50. UNADKAT, Jash. *Beginner's Guide To Software Application Testing* [online]. BrowserStack, 2021 [cit. 2022-06-14]. Dostupné z: https://www.browserstack.com/guide/learn-software-application-testing.
51. KADLECOVÁ, Michaela. *7 TIPŮ JAK NA TESTOVANÍ WEBŮ A MOBILNÍCH APLIKACÍ* [online]. Rascasone, 2021 [cit. 2022-06-14]. Dostupné z: https://www.rascasone.com/cs/blog/7-tipu-jak-na-testovani-webu-internetovych-stranek-a-mobilnich-aplikaci.
52. *Cross-Origin Resource Sharing (CORS)* [online]. MDN, 2022 [cit. 2022-06-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

53. *MDN Web Docs Glossary: Definitions of Web-related terms: Origin* [online]. MDN, 2021 [cit. 2022-06-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Origin.
54. *nelmio/NelmioCorsBundle: Adds CORS (Cross-Origin Resource Sharing) headers support in your Symfony application* [online]. Nelmio, 2022 [cit. 2022-06-14]. Dostupné z: https://github.com/nelmio/NelmioCorsBundle.
55. RICHARDSON, Alan. *What is Static Analysis?* [Online]. SECURE CODE WARRIOR, 2021 [cit. 2022-06-14]. Dostupné z: https://www.securecodewarrior.com/blog/what-is-static-analysis.
56. *phpstan/phpstan: PHP Static Analysis Tool - discover bugs in your code without running it!* [Online]. Ondřej Mirtes, 2022 [cit. 2022-06-14]. Dostupné z: https://github.com/phpstan/phpstan.
57. MIRTES, Ondřej. *PHPStan: Find Bugs In Your Code Without Writing Tests!* [Online]. PHPStan, 2016 [cit. 2022-06-14]. Dostupné z: https://phpstan.org/blog/find-bugs-in-your-code-without-writing-tests.
58. MIRTES, Ondřej. *User Guide: Rule Levels* [online]. PHPStan, 2022 [cit. 2022-06-14]. Dostupné z: https://phpstan.org/user-guide/rule-levels.
59. *sebastianbergmann/phpunit: The PHP Unit Testing framework.* [Online]. Sebastian Bergmann, 2022 [cit. 2022-06-14]. Dostupné z: https://github.com/sebastianbergmann/phpunit.
60. *Testing* [online]. Symfony, 2022 [cit. 2022-06-20]. Dostupné z: https://symfony.com/doc/current/testing.html.
61. *PHPUnit Manual: 9. Code Coverage Analysis* [online]. Sebastian Bergmann, 2021 [cit. 2022-06-21]. Dostupné z: https://phpunit.readthedocs.io/en/9.5/code-coverage-analysis.html.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `text` ..... text práce
  - `DP_Hojek_Tomáš_2022.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojová forma práce ve formátu LaTeX
- `assets` ..... přílohy
  - `code_coverage` ..... výstup z code coverage ve formátu HTML
  - `database_and_diagram` ..... schémata databáze a diagramy případů užití
  - `design` ..... diagramy a modely týkající se návrhu systému
  - `docs` ..... dokumentace API v OpenAPI