FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Backend e-shopu – doprovodné procesy
Student:	David Mareš
Vedoucí:	Ing. Jan Matoušek
Studijní program:	Informatika
Obor / specializace:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Ve spolupráci se studentským týmem realizujte část backendu e-shopu. Zaměřte se na automatizaci pravidelných podpůrných procesů (přepočty cen, kontroly dat atd.) a vytvořte univerzální nástroj pro generování podkladů pro účetní software a úřady. Vytvářený backend musí nabízet vhodné rozhraní pro nově vznikající frontend. Kladťe důraz na práci s týmem.

Postupujte v těchto krocích:
1. Analyzujte současné řešení e-shopu spravovaného společností Jagu s.r.o. a předešlé práce věnující se vývoji daného projektu.
2. Analyzujte pravidelné procesy související s provozem e-shopu; zaměřte se na ty, které nesouvisejí přímo s vyřizováním objednávek. Analyzujte exporty podkladů pro účetní software a Český statistický úřad.
3. Navrhněte nové komponenty systému pro podporu těchto procesů, návrh konzultujte s ostatními členy týmu i se zadavatelem.
4. Implementujte výsledný návrh.
5. Návrh řádně zdokumentujte a vhodnými způsoby otestujte.
6. Zhodnoťte výsledné řešení, navrhněte úpravy do budoucna.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 8. února 2023 v Praze.

Bakalářská práce

BACKEND E-SHOPU – DOPROVODNÉ PROCESY

David Mareš

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Matoušek
10. května 2023

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych především vyjádřit vděčnost svému vedoucímu, Ing. Janu Matouškovi, za jeho cenné rady a konzultace po celou dobu tvorby této práce, stejně jako za jeho podporu a volnost při výběru tématu. Dále bych chtěl poděkovat své rodině a přátelům za veškerou podporu, kterou mi poskytovali nejen během psaní této práce, ale i v průběhu celého studia.
Rád bych také poděkoval Danielovi Espitiovi za jeho vynikající práci s korekturou textu a Nikitovi Golmgrenovi za příjemnou spolupráci.

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.

Abstrakt

Práce se zaobírá rozšířením backendové části systému pro správu e-shopu. V rámci práce proběhla důkladná analýza aktuálně používaného řešení, které bylo potřeba nahradit z důvodu použití zastaralých technologií a vysoké provázanosti, což způsobovalo obtížnou údržbu i rozšiřitelnost. Práce se skládá z několika částí, z nichž každá přináší něco nového a užitečného pro správu e-shopu.

Jedním z klíčových prvků této práce je nástroj pro export dat, který využívá Doctrine ORM a vlastní parser pro skládání databázových dotazů, a následnou tvorbu souboru ve vybraném datovém formátu. Tento nástroj byl vytvořen s ohledem na vysokou konfigurovatelnost, ale také bezpečnost.

Dalším významným prvkem práce je řešení pro automatizaci procesů. Vytvořené řešení využívá Symfony Command společně s balíčkem The ScheduleBundle pro plánování a správu jednotlivých činností.

Na závěr se práce věnuje možným rozšířením vytvořeného řešení, především vystavením uživatelského rozhraní pro tvorbu konfigurací do nástroje pro export dat, který v rámci této práce vznikl.

Klíčová slova administrace e-shopu, backend, exportování, automatizace procesů, PHP, Symfony, Doctrine ORM, DQL, Symfony Command, The ScheduleBundle

Abstract

The work focuses on expanding the backend part of a system for managing an e-shop. As part of the work, a thorough analysis of the currently used solution was conducted, which needed to be replaced due to the use of outdated technologies and high complexity, making maintenance and scaling difficult. The work consists of several parts, each of which brings something new and useful for managing the e-shop.

One of the key elements of this work is a tool for exporting data, which uses Doctrine ORM and a custom parser for composing database queries and creating files in selected data format. This tool was created with a focus on high configurability and security.

Another significant element of the work is the solution for automating processes. The created solution uses Symfony Command together with The ScheduleBundle package for scheduling and managing individual tasks.

Finally, the work deals with possible extensions of the created solution, especially the development of a user interface for creating configurations used by the data export tool, which was created as part of this work.

Keywords e-shop administration, backend, exporting, process automation, PHP, Symfony, Doctrine ORM, DQL, Symfony Command, The ScheduleBundle

Seznam zkratek

API	Application Programming Interface
CSV	Comma-separated values
ČSÚ	Český Statistický Úřad
DQL	Data Query Language
EAN	European Article Number
JSON	JavaScript Object Notation
MVC	Model View Controller
ORM	Object Relational Mapper
PHP	PHP: Hypertext Preprocessor
REST	Representational State Transfer
SKU	Stock Keeping Unit
SQL	Structured Query Language
XML	Extensible Markup Language

Úvod

Tato práce se zabývá automatizací procesů, které jsou nezbytné pro provoz e-shopu. Cílem je vytvořit komponentu, která umožní snadnou a efektivní správu e-shopu s minimálními manuálními zásahy.

Kromě automatizace pravidelných činností se tato práce zaměřuje i na synchronizaci zásob, stavu objednávek, produktů a zásilek se skladovým systémem. Tento nástroj je určen pro provozovatele e-shopu, kteří chtějí optimalizovat svůj pracovní proces a minimalizovat náklady na personál.

Mezi pravidelné činnosti, které tato práce zpracovává, patří: aktualizace kurzu měn, kontrola funkčnosti yotutube odkazů, hlídací pes pro produkty, rozesílání žádostí o recenzi, překlad produktů, přepočet cache cen.

V této práci se nebudu zabývat procesy, které přímo souvisejí s vyřizováním objednávek a načítáním produktů od dodavatelů, tyto procesy jsou nad rámec této práce a jsou již obvykle řešeny pomocí specializovaných nástrojů.

V této práci bude také představen nástroj pro generické exporty, který umožní rychlý a jednoduchý export dat z e-shopu do různých formátů. Tento nástroj může být užitečný pro všechny, kdo potřebují pravidelně exportovat data z e-shopu a zpracovávat je v jiných aplikacích.

Důležitost tématu spočívá v tom, že automatizace procesů a vytvoření nástrojů pro export dat může výrazně usnadnit a zefektivnit práci provozovatelů e-shopů. To může vést ke snížení nákladů na personál, zvýšení produktivity a zlepšení zákaznického zážitku.

Motivací volby tématu je neustále rostoucí obliba e-commerce a potřeba efektivního řízení e-shopů. V této práci budou využity moderní technologie a programovací jazyky, které umožní vytvořit robustní a uživatelsky přívětivé řešení.

Tato práce bude užitečná pro provozovatele e-shopů, kteří chtějí optimalizovat svůj pracovní proces a minimalizovat náklady na personál. Také může být inspirací pro další vývojáře, kteří se chtějí věnovat automatizaci procesů a vývoji nástrojů pro správu e-shopů.

Mé řešení vzniká v rámci projektu, jehož účelem je nahradit současné řešení spravované společností *Jagu s.r.o.* za nové, které bude lépe udržitelné, rozšiřitelné a odstraní technologické limitace současného řešení.

Projektem se zabývá mimo zaměstnanců *Jagu s.r.o.* i mnoho studentů *FIT ČVUT v Praze* v rámci předmětu *Softwarový týmový projekt* nebo ve svých bakalářských či diplomových pracích.

Tato práce přímo navazuje na kód aplikace vytvořený v bakalářské práci Aloise Kouby a diplomové práci Tomáše Hojka.

Současně na tomto projektu vzniká bakalářská práce kolegy Nikity Golmgrena, který se zaobírá košíkem e-shopu.

Kapitola 1
Cíle práce

Cílem této bakalářské práce je vytvořit dvě nové komponenty, které budou řešit podpůrné procesy na backendu e-shopu. První komponenta se zaměří na automatizaci pravidelných činností, které jsou nezbytné pro správný chod e-shopu. Druhá komponenta bude sloužit jako exportovací nástroj umožňující konfigurovatelnost uživatelem a schopnost exportu dat z databáze v různých datových formátech.

Důležité je soustředit se na spolupráci se zaměstnanci *Jagu s.r.o.*, výše zmíněným studentským týmem, který v rámci předmětu *Softwarový týmový projekt* současně implementuje komponentu pro odesílání emailů, a v neposlední řadě s kolegou Nikitou Golmgrenem, který v rámci své bakalářské práce implementuje komponentu košíku e-shopu, aby vzniklé řešení bylo kompatibilní a jednotné.

Pro úspěšné splnění cíle je nutné nejprve provést analýzu současného řešení e-shopu spravovaného společností *Jagu s.r.o.* a předešlých prací věnovaných vývoji tohoto projektu.

Dalším dílčím cílem je provedení analýzy doprovodných pravidelných procesů souvisejících s provozem e-shopu, včetně synchronizačních procesů se skladem e-shopu a dalších systémů a podpůrných činností nezbytných pro provoz e-shopu.

Třetím dílčím cílem je analýza exportů podkladů pro účetní software a Český statistický úřad, ale také možností implementace univerzálního nástroje pro export dat do těchto systémů.

Čtvrtým dílčím cílem je návrh nových komponent systému pro podporu výše uvedených procesů a konzultace návrhu s ostatními členy týmu a zadavatelem.

Pátým dílčím cílem je samotná implementace navržených komponent a následné testování výsledného řešení.

Posledním dílčím cílem je důkladná dokumentace návrhu a výsledného řešení a zhodnocení jeho účinnosti s návrhem případných úprav pro budoucí využití.

Kapitola 2
Analýza

Analýza je velmi důležitou součástí vývoje software, která nám pomůže pochopit problém, který řešíme, zjistit jeho náročnost a odhalit komplikované části. Bez analýzy je velká šance, že program buď nevznikne vůbec, nebo vznikne suboptimální řešení.

V této části se nejdříve zaměřím na analýzu stavu projektu a současného řešení od *Jagu s.r.o.* Dále se budu zabývat analýzou vývoje a spolupráce v týmu, jelikož tato práce vzniká společně s Nikitou Golmgrenem, jehož bakalářská práce se věnuje košíku vznikajícího e-shopu, a také s týmem studentů z předmětu BI-SP1. Nakonec se v této části podívám na problematiku dvou komponent, které nově vznikají v rámci této práce.

2.1 Stav projektu

Aktuální administrace používaná několika zákazníky *Jagu s.r.o.* je implementována pomocí Open-Cart, open source eCommerce platformy, která slouží jako základ pro vývoj e-shopů. Řešení je stále funkční, ale z důvodu použitých technologií a velké provázanosti je toto řešení dnes náročné udržovat a především rozšiřovat.

Z tohoto důvodu vznikl tento projekt, který má za úkol nahradit aktuální, dnes však již zastaralé, řešení administrace e-shopů. V rámci projektu vzniká nový frontend i backend. Na projektu se kromě zaměstnanců *Jagu s.r.o.* podílejí studenti *FIT ČVUT v Praze* v rámci předmětu *Softwarový týmový projekt* a také ve svých bakalářských či diplomových pracích.

Projekt je nyní v rozpracovaném stavu a vyvíjený software není zatím schopen být využit v produkčním prostředí kvůli některým chybějícím funkcím a nedostatečnému otestování software jako celku v provozu či při napojení na reálnou databázi.

2.1.1 Aktuální řešení

Aktuální řešení od společnosti *Jagu s.r.o* umožňuje jednoduše spravovat e-shopy pomocí administračního systému. Je vyjímečné především zaměřením se na možnost mít jednu administraci pro správu více e-shopů, kde mohou být produkty sdíleny mezi různými e-shopy, ale také instancovány pro jednotlivé e-shopy s různými obrázky, popisy či cenami.

Toto řešení bylo založeno na open source platformě Opencart. OpenCart je open-source e-commerce platforma napsaná v programovacím jazyce PHP. Je navržena tak, aby poskytovala snadné a přehledné řešení pro online prodejce a podnikatele, kteří chtějí vytvořit svůj vlastní internetový obchod.

OpenCart poskytuje celou řadu funkcí, včetně možnosti správy katalogu, správy objednávek, správy skladů, možností plateb a dopravy, správy zákazníků a mnoho dalšího. Platforma je navržena s ohledem na snadnou instalaci a používání a je podporována rozsáhlou komunitou vývojářů, kteří pravidelně přispívají novými funkcemi a plug-iny. [1]

OpenCart využívá MySQL databázi pro ukládání dat. V době vzniku současného řešení ještě MySQL nepodporovalo cizí klíče, z tohoto důvodu se v databázi časem objevili nekonzistentní data, což je jedním z důvodů, proč vznikl tento projekt.

Dalším důvodem je velmi obtížná rozšiřitelnost řešení z důvodu stáří použitých technologií, ale také technického dluhu, který postupně v tomto řešení vznikl.

Společnost *Jagu s.r.o.* se zabývá především:

- Zakázkovým vývojem softwarových řešení na míru
- Grafickými službami
- Tvorbou webových stránek či e-shopů
- Konzultacemi, podporou a provozem webových aplikací [2]

Mezi e-shopy používající existující produkt *Jagu s.r.o.* patří:

stylka.cz E-shop zaměřující se na prodej nejen elektroniky pro domácnost, zdraví a sport.

muj-russellhobbs.cz E-shop, který je certifikovaným prodejcem značky Russel Hobbs.

muj-remington.cz E-shop, který je certifikovaným prodejcem značky Remington.

kudrnka.cz E-shop zaměřující se na prodej domácí elektroniky, potřeb pro chovatele, módy a produktů pro zdraví a kosmetiku.

sgear.cz E-shop prodávající produkty v oblasti kempování, zdravotnické vybavení a potřeby pro přežití.

2.1.2 Nové řešení

Jak jsem již zmínil v úvodu, vznikl projekt, který má nahradit současnou administraci e-shopu novým moderním řešením. Na projektu vznikly bakalářské i diplomové práce na backend i frontend e-shopu a jeho administrace. Vzniku nového řešení se ve svých pracích věnovali Bc. Alois Kouba, Bc. Martin Dvořák, Bc. Jan Babák a Ing. Tomáš Hojek. Dále se tomuto projektu věnovali i současně věnují další studenti v rámci předmětu *Softwarový týmový projekt 1* i *2*.

Ve zmíněných pracích byly zvoleny programovací jazyky, frameworky a další technologie pro běh aplikace. Také byl vytvořen funkční základ backendu i frontendu nové aplikace. Nyní je potřeba rozšířit tento základ o další požadované funkce, aby bylo možné v budoucnu systém nasadit.

Následující podkapitoly se věnují struktuře projektu a použitým technologiím na backendu. Při psaní jsem čerpal především z bakalářské práce Aloise Kouby *„Backend administrace e-shopu“* [3] a diplomové práce Tomáše Hojka *„Backend a CI administrace e-shopu“* [4], kteří se ve svých pracích detailně zabývali vytvořením backendu aplikace a volbou architektury a technologií.

2.1.2.1 Struktura projektu

Jako architektonický styl nové aplikace byl zvolen populární styl MVC. Tato architektura je jedním z hlavních přínosů nové aplikace. *„Po konverzaci v týmu a konzultaci se zadavatelem jsme dospěli k závěru, že nejvhodnější bude napsat novou administraci jako třívrstvou. Konkrétně se jedná o architektonický vzor MVC, neboli Model View Controller.“*. [3]

MVC *(Model-View-Controller)* je architektonický vzor pro vývoj softwaru. Je rozdělen na tři komponenty: Model, View a Controller.

Model reprezentuje data a logiku aplikace. Model vystavuje rozhraní pro manipulaci s daty a poskytuje tím aplikaci sémantickou a významovou konzistenci.

View reprezentuje prezentační vrstvu aplikace a poskytuje uživatelské rozhraní pro interakci s uživatelem. View může být stránka webového prohlížeče nebo okno desktopové aplikace.

Controller zpracovává uživatelské vstupy a řídí tok aplikace. Controller přijímá uživatelské vstupy od View, zpracovává je a aktualizuje Model nebo View.

MVC je užitečné pro oddělení dat, logiky a prezentační vrstvy aplikace, což umožňuje efektivní vývoj, údržbu a testování aplikace. [5]

Model byl následně rozšířen o další vrstvu a to DTO *(Data Transfer Objects)* objekty. *„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.“* [4] Ty poskytují další vrstvu mezi komponenty Model a Controller.

Projekt je rozdělen na frontend a backend. Frontend reprezentuje komponentu View, zatímco backend Controller i Model. V projektu vznikají nové řešení pro obě části. Obě vyvíjené části spolu komunikují pomocí REST API.

REST byl zvolen na základě nefunkční požadavku *„Aplikace má API s vhodnou architekturou“. „Ten (míněno architektonický styl REST) jsme zvolili po diskuzi v týmu a po validaci zadavatele.“*. [3]

REST *(Representational State Transfer)* je architektonický styl pro navrhování webových služeb, který klade důraz na jednoduchost, škálovatelnost, přenositelnost a rozšířitelnost. REST API je rozhraní, které splňuje omezení stylu REST, využívající standardní HTTP metody *(GET, POST, PUT, DELETE, atd.)* k přenesení dat v různých formátech. Nejčastěji používaným formátem je JSON, ale lze data přenášet i pomocí formátů XML, HTML a dalších. [6]

Tato práce se bude zabývat především backendovou částí e-shopu. V následující kapitole udávám výpis technologií, které byly vybrány pro vývoj backendu. Diagram se strukturou projektu lze nalézt obrázku 2.1.

2.1.2.2 Použité technologie

PHP Kód projektu je psaný v PHP. PHP je skriptovací jazyk pro vývoj dynamických webových stránek a aplikací, který umožňuje interakci s databázemi, manipulaci s daty, ale také tvorbu uživatelského rozhraní. Jazyk byl vytvořen Rasmusem Lerdorfem v roce 1994 a postupně se stal populárním mezi webovými vývojáři díky své jednoduchosti a efektivnosti. [7, 8]

Symfony Použitým frameworkem je Symfony. Symfony je open-source framework pro vývoj webových aplikací v jazyce PHP. Bylo vytvořeno v roce 2005, dnes je využíváno mnoha vývojáři po celém světě. Symfony poskytuje řadu nástrojů a komponent, které usnadňují vývoj webových aplikací a zvyšují jejich efektivitu.

Framework usnadňuje vývoj webových aplikací tím, že nabízí mnoho hotových komponent, které je možné využít při tvorbě aplikace, jako například autentizace, zabezpečení, formuláře, ORM (Object-Relational Mapping) a mnoho dalších. Symfony je navržen tak, aby byl modulární, flexibilní a snadno rozšiřitelný. Framework je vyvíjen a udržován komunitou, která poskytuje širokou podporu a dokumentaci pro vývojáře. [9, 10]

„Famework 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“. [4]

MySQL Lokální i vzdálená testovací databáze jsou v MySQL stejně jako reálná databáze současné administrace. MySQL je open-source relační databázový systém, který je široce používán pro správu webových aplikací a dalších softwarových projektů. Je to relační databáze, což znamená, že data jsou organizována do tabulek, které jsou propojeny pomocí

klíčových polí. MySQL podporuje standardní dotazovací jazyk SQL a poskytuje vysokou úroveň zabezpečení a výkonu pro správu dat.

MySQL nabízí mnoho funkcí, jako jsou transakce, replikace a podpora pro ukládání a vyhledávání prostorových dat. Je také snadno přizpůsobitelný a může být použit s mnoha programovacími jazyky, jako jsou PHP, Python a Java. Díky své vysoké dostupnosti a spolehlivosti se MySQL stal jedním z nejpopulárnějších relačních databázových systémů v průmyslu. [11]

Doctrine ORM Doctrine ORM je knihovna pro jazyk PHP, která slouží jako *Object-Relational Mapping* (ORM) nástroj pro pohodlný přístup k databázi. ORM umožňuje převést databázové tabulky na objekty a umožňuje jednoduché manipulace s nimi pomocí jazyka PHP. Doctrine ORM poskytuje jednoduchý způsob práce s databází a zajišťuje bezpečnost aplikace, jelikož je chráněna proti SQL injection útokům. Knihovna také nabízí podporu pro vytváření relačních vztahů mezi objekty. Doctrine ORM se často používá v rámci frameworků, jako je například Symfony. [12]

nginx Nginx je open-source webový server a reverzní proxy server, který je často používán jako náhrada za tradiční Apache webový server. Nginx je navržen tak, aby byl rychlý, efektivní a škálovatelný. Je často používán pro vysoké provozovatelské zatížení.

Reverzní proxy server umožňuje přesměrování požadavků z klienta na jiný server, což může pomoci snížit zátěž na primárním serveru. Nginx také umožňuje nastavení rozsáhlých pravidel řízení přístupu a řízení toku dat.

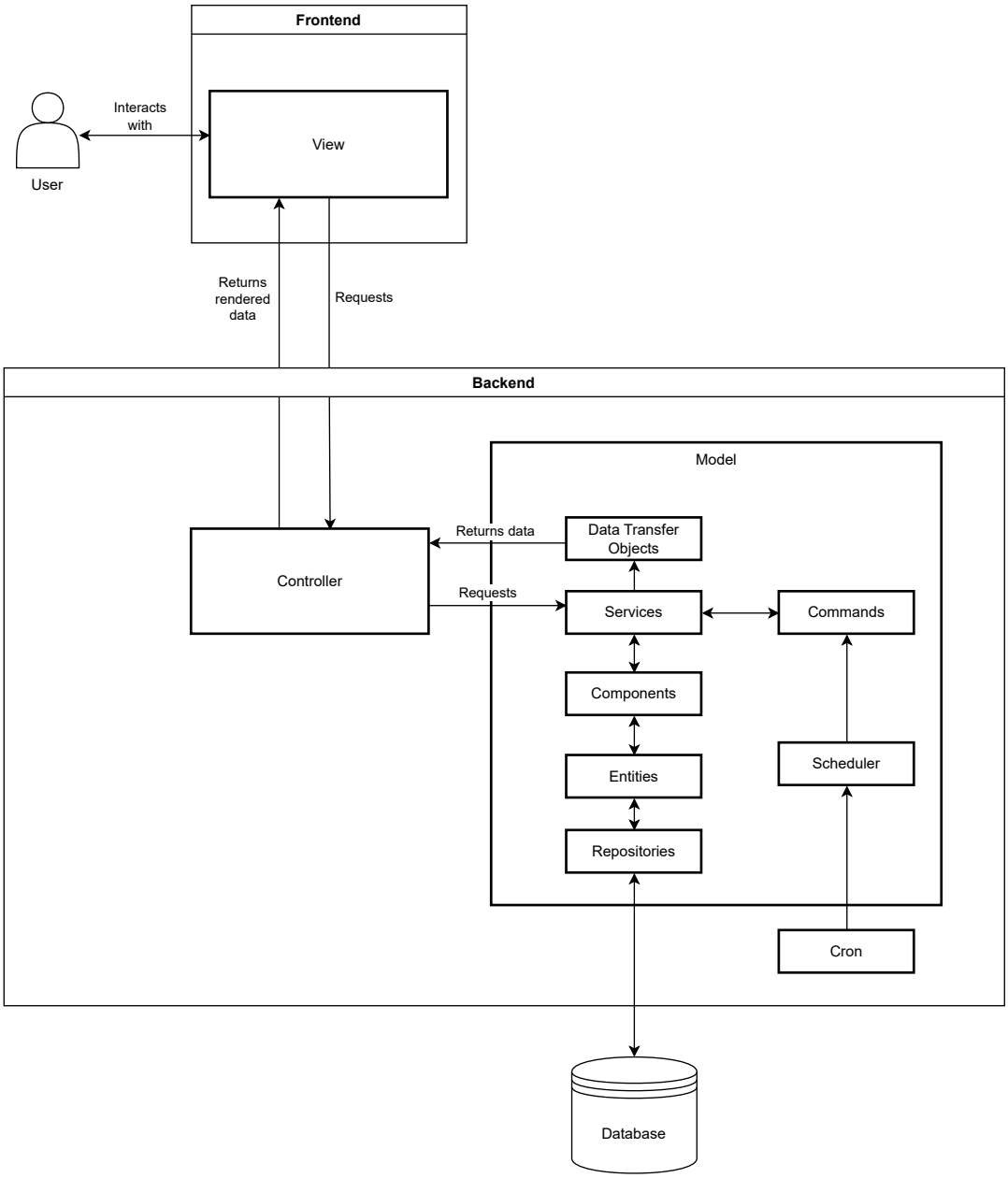
Kromě webového serveru a reverzního proxy serveru může Nginx také slouží jako mail proxy server a podporuje TCP a UDP protokoly. [13]

Docker Docker je open-source projekt umožňující vytváření, běh a správu virtualizovaných kontejnerů pro aplikace. Každý kontejner obsahuje všechny závislosti potřebné k běhu aplikace, včetně operačního systému, což umožňuje snadné přenositelnosti mezi různými prostředími. Docker se používá pro izolaci aplikací a zajišťuje, aby aplikace běžely stejně v různých prostředích.

Docker umožňuje také snadné doručování aplikací, protože každý kontejner je izolovaný a může být snadno přenesen mezi různými prostředími bez nutnosti řešit problémy s kompatibilitou závislostí a konfigurace. [14]

Celkově tedy Docker umožňuje rychlejší a snadnější vývoj, doručování a běh aplikací v izolovaných kontejnerech.

Keycloak Open Source správce identit a přístupu. Je založen na standartních protokolech a nabízí podporu OpenID Connect, OAuth 2.0 a SAML. Administrátor spravuje všechny aspekty Keycloak serveru, může povolit či zakázat jisté služby. Administrátor může spravovat uživatele, jejich práva a *sessions*. Uživatelé si mohou aktualizovat profil, měnit hesla či si nastavit dvoufaktorovou autentifikaci. Umožňuje autentifikovat uživatele a nabízí i autorizační služby pokud mají uživatelé různá práva. Využívá se pro správu uživatelů a jejich autorizaci. [15]



Obrázek 2.1 Struktura projektu

2.2 Vývoj v týmu

Vývojem v týmu je potřeba se zabývat, jelikož i tato práce vzniká v kooperaci se studentským týmem. Jak jsem zmínil v úvodu, současně s touto prací vzniká bakalářská práce Nikity Golmgrena zaměřující se na košík e-shopu, ale také probíhá spolupráce se studentským týmem, který v rámci předmětu *Softwarový týmový projekt* rozšiřuje další části frontendu i backendu aplikace.

V rámci této práce jsem měl také možnost podílet se tvorbě rozhraní komponenty pro odesílání emailů v roli zadavatele. Během této spolupráce jsem vytvořil požadavky na vnikající komponentu, které jsem následně s týmem konzultoval, stejně tak jako návrh rozhraní. Tato komponenta bude v budoucnu využita řešením pro automatizaci činností, které vzniklo v rámci této práce, k odesílání emailů.

V nadcházejících sekcích se budu zabývat výhodami vývoje v týmu, problémy a jejich řešeními.

2.2.1 Výhody vývoje v týmu

Vývoj v týmu má mnoho výhod, zde ze zaměřím na ty, které považuji za nejdůležitější:

Rychlejší vývoj Vývoj v týmu je obecně rychlejší než individuální vývoj. Různí členové týmu mohou pracovat na různých částech projektu současně, což vede k rychlejší tvorbě výsledného produktu.

Větší možný rozsah Vývoj v týmu umožňuje práci na rozsáhlých projektech, které by jednotlivec nebyl schopen zvládnout sám v omezených časových podmínkách. Také lze využít různých dovedností a specializací členů týmu pro pokrytí částí projektu, kde jiní nemají takové dovednosti.

Předávání zkušeností Členové týmu mezi sebou mohou sdílet své zkušenosti a zvyšovat tím dovednosti týmu jako celku, což napomáhá rychlejšímu řešení problémů, které v projektu mohou nastat. Předáváním zkušeností se členové týmu neustále vzdělávají, to se pak často projeví na budoucích projektech.

Lepší kvalita kódu Narozdíl od individuálního vývoje, při vývoji v týmu jsou často jasně definované programovací standarty a konvence, které se členové snaží dodržovat. Také probíhají kontroly kvality kódu, jako je například *code review*. Obecně tak vzniká v týmových projektech kvalitnější kód.

[16, 17]

2.2.2 Problémy při vývoji v týmu

Nejčastějším problémem při vývoji bývají problémy s komunikací. Nedostatečná koordinace mezi členy týmu může způsobit výstupy, které nesplňují požadavky. Pokud nejsou členové týmu schopni jasně a srozumitelně komunikovat, může dojít k nedorozumění a výsledkem mohou být nesprávná rozhodnutí.

Dalším častým problémem jsou nejasné cíle. Ty mohou být interpretovány každým členem týmu trochu jinak a při nedostatečné komunikaci může být výsledkem produkt, který nesplňuje požadavky.

V praxi se také často setkáváme s problémy s dokumentací. Přesná, kompletní a aktuální dokumentace je důležitou součástí úspěšného vývoje. Pokud dokumentace není kvalitní, může dojít k chybám a zpomalení vývoje. Nesprávné nebo zastaralé informace vedou k nedorozuměním a špatnému chápání požadavků.

V neposlední řadě, problémem v týmovém vývoji také bývá zpoždění jedné části projektu, což často způsobí zpomalení celého vývoje. Toto zpoždění může vést k frustraci, neboť nelze dodržet stanovené termíny a další části projektu nemohou pokračovat včas. [18, 19]

V tomto projektu se setkáváme především s problémem, že mnoho studentů, kteří se na něm podílí, se účastní svého prvního většího projektu v oblasti softwarového vývoje a zároveň nemají dostatečné zkušenosti s danými technologiemi.

2.2.3 Mitigace problému při vývoji v týmu

V tomto projektu byly použity různé nástroje, aby se předešlo co nejvíce problémů spojených s vývojem v týmu.

2.2.3.1 Nedostatečná komunikace a koordinace

Pro řešení tohoto problému mohou být využity různé komunikační nástroje, jako například Slack nebo Microsoft Teams.

Důležité je také zajistit, aby komunikace byla dostačná. Toho můžeme dosáhnout pravidelným setkávání týmu.

Komunikace by se neměla omezovat pouze na textové zprávy, ale měly by být využity fyzické schůzky nebo alespoň online hovory. I přesto, že textové zprávy bývají nejrychlejší formou předání informací, nebývají zpravidla tou nejlepší. [20]

V průběhu projektu byly použity jako hlavní komunikační prostředky fyzické schůzky a platforma Slack. Přes platformu Slack probíhala textová komunikace a hovory. To pomohlo zlepšit koordinaci v rámci týmu a zabránit problémům spojeným s nedostatečnou komunikací.

„Slack je platforma pro podporu produktivity, která každému umožňuje automatizaci bez nutnosti psaní kódu a využití umělé inteligence. Díky ní se vyhledávání a sdílení znalostí stává snadným a týmy zůstávají propojeny a angažované. Po celém světě je Slack platformou, které společnosti důvěřují a lidé ji rádi používají.“ [21]

Využití verzovacího systému je důležitou formou koordinace v týmovém vývoji. Verzovací systém umožňuje správu a sledování změn v kódu, často poskytuje i další nástroje pro komunikaci a kooperaci. V tomto konkrétním projektu je pro verzování kódu použit nástroj GitLab.

GitLab je *DevSecOps* platforma pro Git repozitáře, která poskytuje integrované funkce jako například kontinuální integraci, sledování problémů, podporu týmu a dokumentaci wiki. [22]

„DevSecOps — zkráceně pro vývoj, bezpečnost a operace — automatizuje integraci zabezpečení v každé fázi životního cyklu softwaru, od počátečního návrhu přes integraci, testování, nasazení a dodání softwaru.“ [23]

2.2.3.2 Nejasné cíle

Problém nejasných cílů je potřeba řešit důkladným popisem požadavků a specifikací. [19] Důležité je také zajistit, aby všichni členové týmu měli přístup ke stejným informacím, ale také aby věděli, kde informace hledat. Vhodným nástrojem pro takových sdílení informací může být třeba Redmine nebo Confluence.

V tomto projektu byla pro definici cílů využita především aplikace Redmine, zároveň byly jasně definované cíle v rámci vznikajících bakalářských prací.

„Redmine je flexibilní webová aplikace pro projektový management. Napsaná pomocí frameworku Ruby on Rails, je multiplatformní a podporuje různé databáze.“ [24]

2.2.3.3 Problémy s dokumentací

Problémy s dokumentací lze řešit definováním jasných pravidel pro dokumentaci a pravidelným aktualizováním dokumentace. Důležité je také zajistit, aby dokumentace byla přístupná pro všechny členy týmu, konkrétním příkladem je přístup vývojářů frontendové části ke specifikacím *endpointů* na backendu. Pro dokumentaci kódu lze využít nástroje jako PHPDoc nebo Javadoc nebo různé wiki, které umožňují snadno sdílet dokumentaci s celým týmem.

V tomto projektu existují různé formy dokumentace. V *README* souboru na GitLabu jsou k dispozici instalační návody a informace o častých problémech. Na Redmine wiki jsou k nalezení specifikace, návody a řešené problémy. Dále jsou zdokumentovány *endpointy* backendu pomocí OpenAPI verze 3.0.1 v GitLabu. Pro vytváření této dokumentace je použit nástroj Swagger.

„Swagger je výkonný, přesto snadno použitelný, soubor nástrojů pro vývojáře API týmů a jednotlivců, umožňující vývoj v průběhu celého životního cyklu API, od návrhu a dokumentace až po testování a nasazení.“ [25]

2.2.3.4 Zpoždění projektu

Jednou z možností, jak řešit problém se zpoždění jedné části projektu je například přidělením většího počtu lidí na danou část projektu. To však v ohledu k tomuto projektu není příliš realistické. Je také možné rozdělit úkoly na menší části a vytvořit plán projektu s menšími jasně definovanými termíny. [18] Důležité je také zajistit pravidelné sledování průběhu projektu a řešit případných problémů.

Využití nástrojů pro správu a plánování projektů, jako například Redmine nebo Jira, může v těchto situacích značně pomoci. I pro tyto účely je v tomto využíván zmíněný Redmine.

Dalším způsobem, kterým lze řešit zpoždění jiných částí projektu, je *mock testing*. *Mock testing* je testovací technika, při které se vytvářejí falešné objekty nebo funkce, které nahrazují reálné objekty nebo funkce v testovaném kódu. Umožňuje tak izolovat jednotlivé části kódu. [26] Toho můžeme využít například pro testování částí softwaru, které závisí na jiných částech, které ještě nejsou k dispozici.

2.2.3.5 Neznalost technologií

Tento problém lze řešit mnoha způsoby. Jedním z nejlepších řešení jsou organizovaná školení a workshopy. [27] V rámci tohoto projektu však toto řešení není dobře aplikovatelné, protože projekt vzniká především v rámci různých předmětů na *FIT ČVUT v Praze* a formou bakalářských a diplomových prací.

Nicméně členové týmu mohou využít přístup k dokumentaci v Redmine wiki a na platformě GitLab a dalším zdrojům, které jim pomohou získat nezbytné znalosti pro zapojení se v rámci projektu.

Dalším řešením je zavedení pravidelných schůzek a specializovaných kanálů na platformě Slack, kde mohou členové týmu konzultovat problémy, na které narazí a podívat se, jak je řešili v minulosti. Díky tomu si členové týmu navzájem pomáhat a sdílet své zkušenosti.

2.3 Automatizace pravidelných činností

Tato část této práce se zaměřuje na automatizaci doprovodných procesů, které se odehrávají na backendu e-shopu. Mezi tyto pravidelné činnosti patří nejen synchronizace s napojenými systémy, ale také aktualizace *cache*, akutalizace kurzů měn, překlad produktů do jiných jazyků a další. V následující části se zaměřím na analýzu těchto procesů.

Současné řešení automatizace zmíněných procesů využívá softwarový plánovač úkolů, který spouští předpřipravené příkazy. Tyto příkazy pak dále spouštějí scripty či PHP kód pro vykonání úkolu. Níže se budu zabívat konkrétními úkoly a jakým způsobem je současný backend vykonává.

Byl mi umožněn přístup k záloze kódu backendové části jednoho z nasazených projektů *Jagu s.r.o.*, abych mohl analyzovat jak v současném řešení fungují automatizované činnosti a exporty do účetnictví.

Díky tomu jsem měl šanci zjistit, jaké pravidelné činnosti probíhají v aktuální administraci a co je očekávané od nové administrace.

Také jsem měl možnost si díky této příležitosti zjistit formáty pro exporty do účetních sytémů, které nenabízeli veřejně dostupnou dokumentaci formátu pro importování do těchto systémů.

2.3.1 Synchronizační činnosti se skladem

Backend e-shopu je napojený na skladový systém, s kterým komunikuje pomocí API. Se skladovým systémem se vyměňují informace o zásobách, objednávkách, produktech a zásilkách.

Je potřeba pravidelně synchronizovat počty zásob produktů, informace o produktech a informace o objednávkách. Skladový systém sleduje stavy zásilek, které je třeba také pravidelně zjišťovat backendem e-shopu pro automatické změny stavů objednávek a následné odesílání emailů zákaníkům.

Samotné sledování zásilek mohou zákazníci sledovat u přepravce, je však potřeba je informovat o převzetí zásilky dopravcem, zrušení objednávky v případě opakovaného nepřevzetí či poděkovat za převzetí zásilky.

Tato funkcionalita byla nedávno přenesena na skladový systém, nově už nebude e-shop přímo komunikovat s API dopravce, ale se skladovým systémem pro zjistění stavu zásilek.

Při volání API skladového systému se musí autorizovat *Bearer tokenem*.

Ten získá pomocí *POST* requestu na url systému pro generování tokenů, v kterém se autentifikuje pomocí přihlašovacích údajů. S touto autorizací pak může využívat API skladového systému.

2.3.1.1 Synchonizace zásob

Jedna z informací, které je potřeba synchronizovat se skladovým systémem, jsou zásoby produktů. Je zapotřebí pravidelně kontrolovat, kolik kusů máme na skladě, a ukládat tyto informace pro e-shop a jeho administraci. Tato synchronizace se provádí každých 5 minut.

Pomocí *GET* requestu na příslušnou adresu API skladového systému se získá odpověď, která se skládá z pole, které je naplněno instancemi produktů s počtem kusů na skladě `quanity` a počtem blokovaných kusů `blocked`.

Ukázku odpovědi skladového systému lze nalézt v příloze: 13

Následně je třeba nahradit počet volných kusů na skladě v administraci e-shopu za počet kusů na interním skladě.

Toho dosáhneme tak, že pro každou instanci produktu vrácenou ze skladového systému nahradíme v tabulce `oc_product` atribut `internal_quantity` za rozdíl `quantity` a `blocked`.

Totéž se provede pro různé možnosti produktu v tabulce `oc_product_option_value`.

Identifikátor produktu a možnosti produktu se aktuálně hledají v databázi uvnitř tabulky `oc_product_octopus` pomocí navráceného identifikátor instance produktu ze skladového systému.

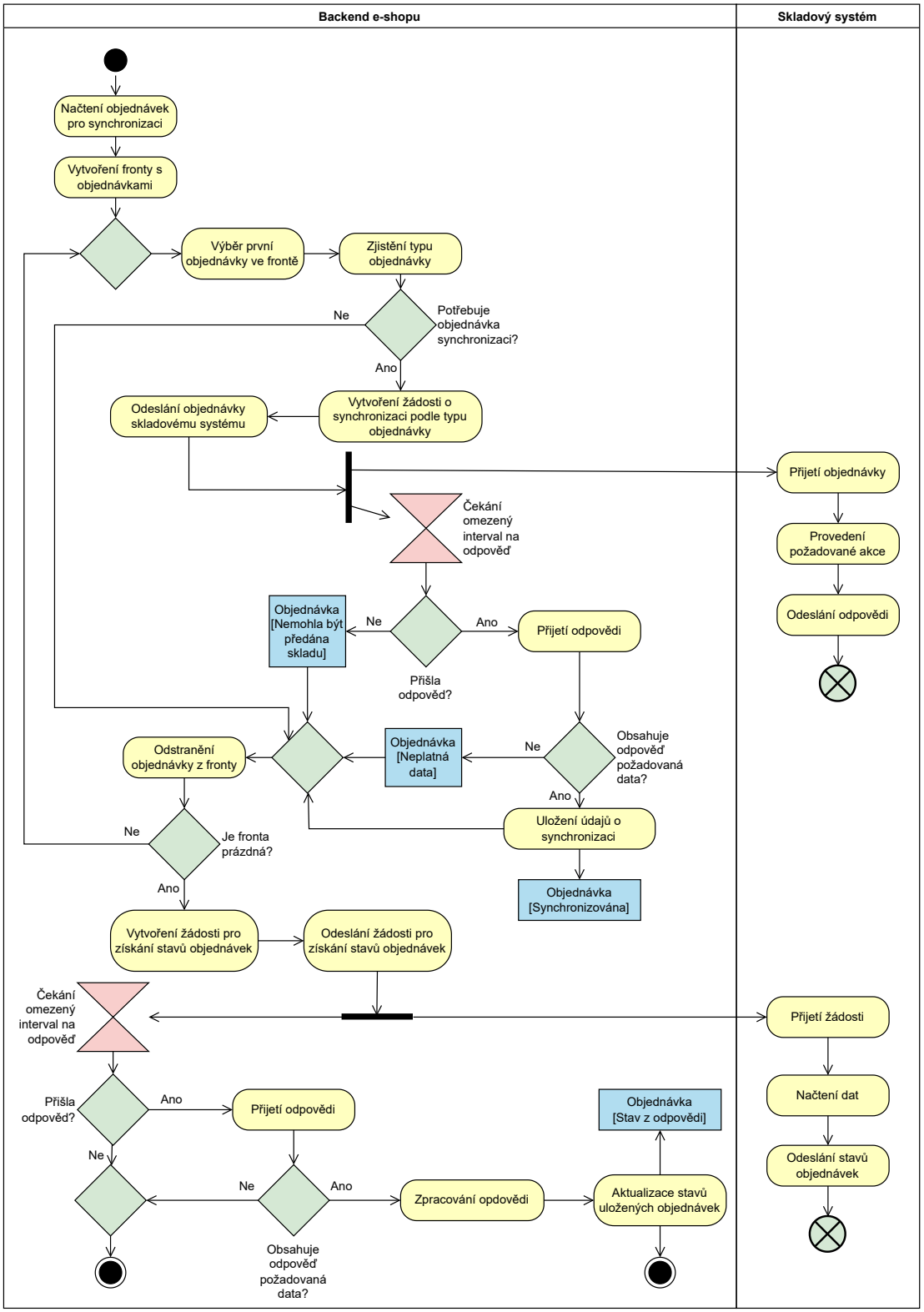
Jelikož se ukázalo, že původní *endpoint* je zastaralý *(deprecated)*, bylo zapotřebí zjistit, který *endpoint* má být nově využit, a ověřit, že nabízí veškeré potřebné informace. Díky poskytnutému přístupu k dokumentaci API skladového systému od *Jagu s.r.o.* jsem nalezl *endpoint*, který nahrazuje původní, a porovnal, že se odpovědi shodují.

2.3.1.2 Synchronizace stavu objednávek

Synchronizace stavu objednávek je nejčastější synchronizační činností mezi e-shopem a skladem. Tato synchronizace se provádí každou minutu oběma směry, tedy jak z e-shopu do skladového systému, tak i ze skladového systému do e-shopu. Do skladového systému se posílají nové objednávky, ale také zrušení nebo úpravy některé z objednávek. Naopak ze skladu přicházejí informace o naskladnění produktů u objednávky nebo o stavech doručených produktů zákazníkovi.

Při synchronizaci objednávek je nutné rozlišovat několik typů objednávek, které jsou uloženy v e-shopu. Tyto typy se dají rozdělit do kategorií: ještě nebyla synchronizována, byla provedena rezervace a zadáno vyskladnění, objednávka byla zrušena, objednávka byla vyresetována ve skladu nebo objednávka není synchronizována a je třeba ji přeskočit.

Z důvodu komplexnosti procesu synchronizace objednávek jsem využil diagram aktivit (viz. obrázek 2.2) pro jeho lepší názornost.



Obrázek 2.2 Activity diagram synchronizace objednávek

2.3.1.3 Synchronizace zásilek

Synchronizace stavu zásilek je důležitou součástí procesu objednávání a dodávání zboží. V minulosti bylo nutné dotazovat se na API jednotlivých dopravců pro získání informace o stavu zásilek. Avšak v současné době nově probíhá synchronizace přes skladový systém, což usnadňuje a zrychluje tento proces. Synchronizace probíhá každých 15 minut.

Pro synchronizaci se načítá seznam objednávek, pro které je potřeba aktualizovat stav zásilek. Tyto objednávky musí splňovat několik kritérií, jako například specifický stav, který je konfigurovatelný v nastavení e-shopu, povolenou synchronizaci se skladem a typ objednávky *„normal“*.

Následně jsou získané stavy zásilek rozděleny do čtyř typů: zákazník převzal zásilku, dopravce převzal zásilku, zásilka byla vrácena a zásilka je připravena.

Pro každý typ zásilek je nutné provést několik kroků. Nejprve se načte informace o objednávkách tohoto typu a informace o notifikaci zákazníka, pokud je to relevantní. Poté se aktualizuje stav objednávky v databázi a uloží se nový stav do historie objednávky. Pokud se jedná o zásilku vyzvednutou zákazníkem s `payment_status` 0, nastaví se `payment_status` na 1 a `payment_modifiedon` na aktuální datum a čas, což indikuje úspěšné vyzvednutí a zaplacení dobírky. Pokud má být zákazník notifikován o změně, tedy `notify` je rovno 1, je zákazník informován o změně stavu zásilky emailem.

Proces informování zákazníka o změně stavu zásilky začíná načtením informací o vybrané objednávce, stavu, o kterém se má zákazníka notifikovat, a jazyku zákazníka. Následně se načte překládací modul a přeloží se texty emailu, jako jsou předmět emailu, text objednávky, text pro datum vytvoření a text pro současný stav objednávky. Pokud je zákazník registrován, přidá se text k faktuře a sestaví se odkaz na fakturu. Poté se sestaví kompletní text emailu, který obsahuje přeložený předmět a text emailu, doplněný o proměnné. Následně se nakonfiguruje odesílač mailů a určí se správná doména jako odesílatel. Poté je dosazen předmět a text a email je odeslán.

Pokud je objednávka vytvořena z Heuréky pomocí Heuréka API, je potřeba o změně stavu informovat i Heuréku. Pro zajištění této funkcionalit se využívá Heuréka API, které je integrováno do systému. Pokud dojde ke změně stavu objednávky, systém tuto informaci předá Heuréce, aby bylo možné zobrazit aktuální stav objednávky v Heuréce.

Celý proces synchronizace zásilek byl vylepšen tím, že se nově provádí přes skladový systém nezávisle na dopravcích. Díky tomu se zjednodušila komunikace s jednotlivými dopravci a snížila se závislost na jejich API.

2.3.1.4 Synchronizace produktů a výrobců

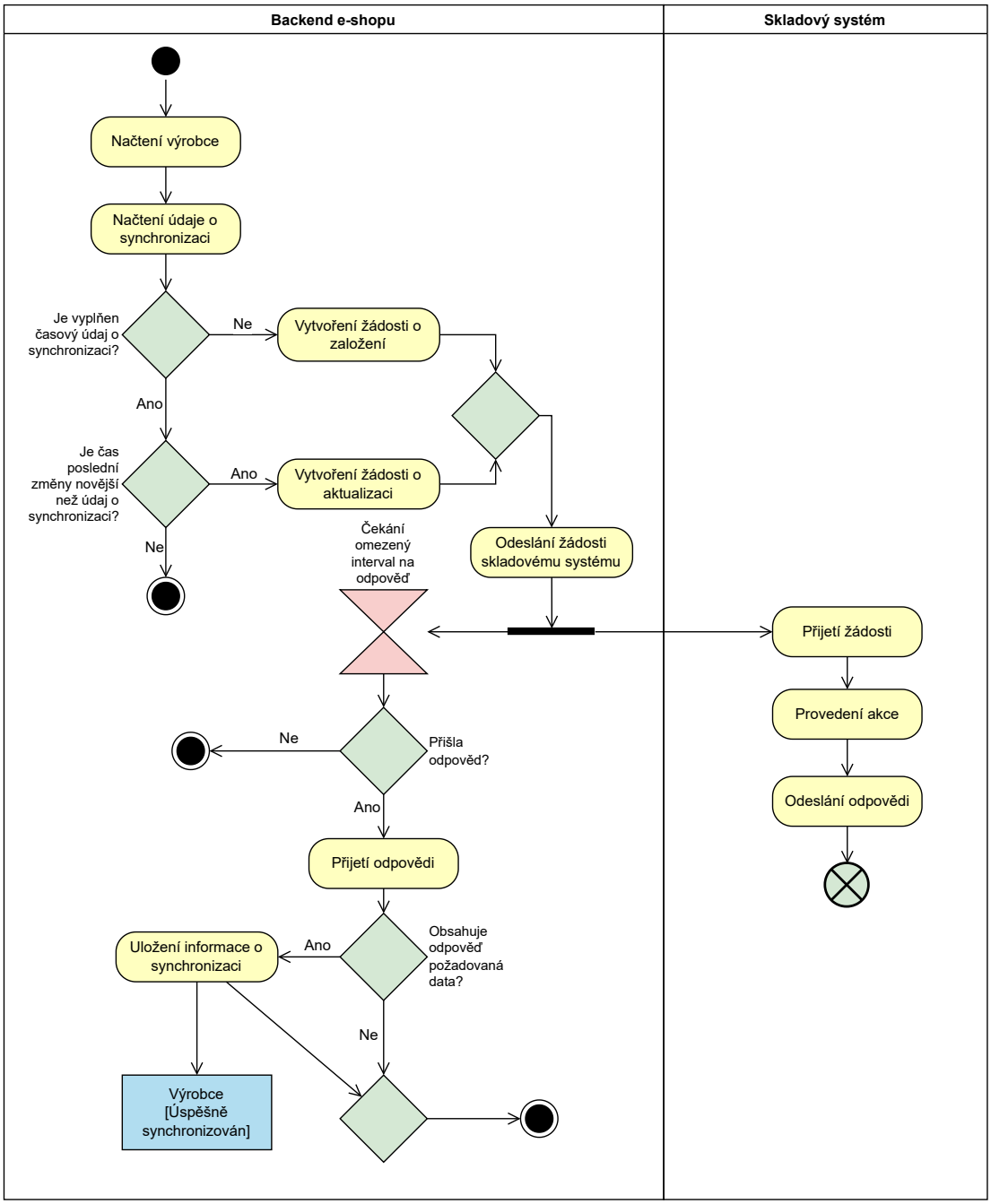
Synchronizace produktů a výrobců se skladovým systémem probíhá každých 15 minut a zahrnuje nejen samotné produkty a výrobce, ale také čárové kódy a obrázky produktů.

Nové a produkty, stejně tak jako výrobci, se synchronizují pomocí atributu `last_sync`, který označuje čas poslední synchronizace. Pokud je atribut prázdný, znamená to, že produkt nebo výrobce ještě nebyl synchronizován. Pokud byl produkt nebo výrobce od poslední synchronizace změněn se pozná pomocí atributu `date_modified`.

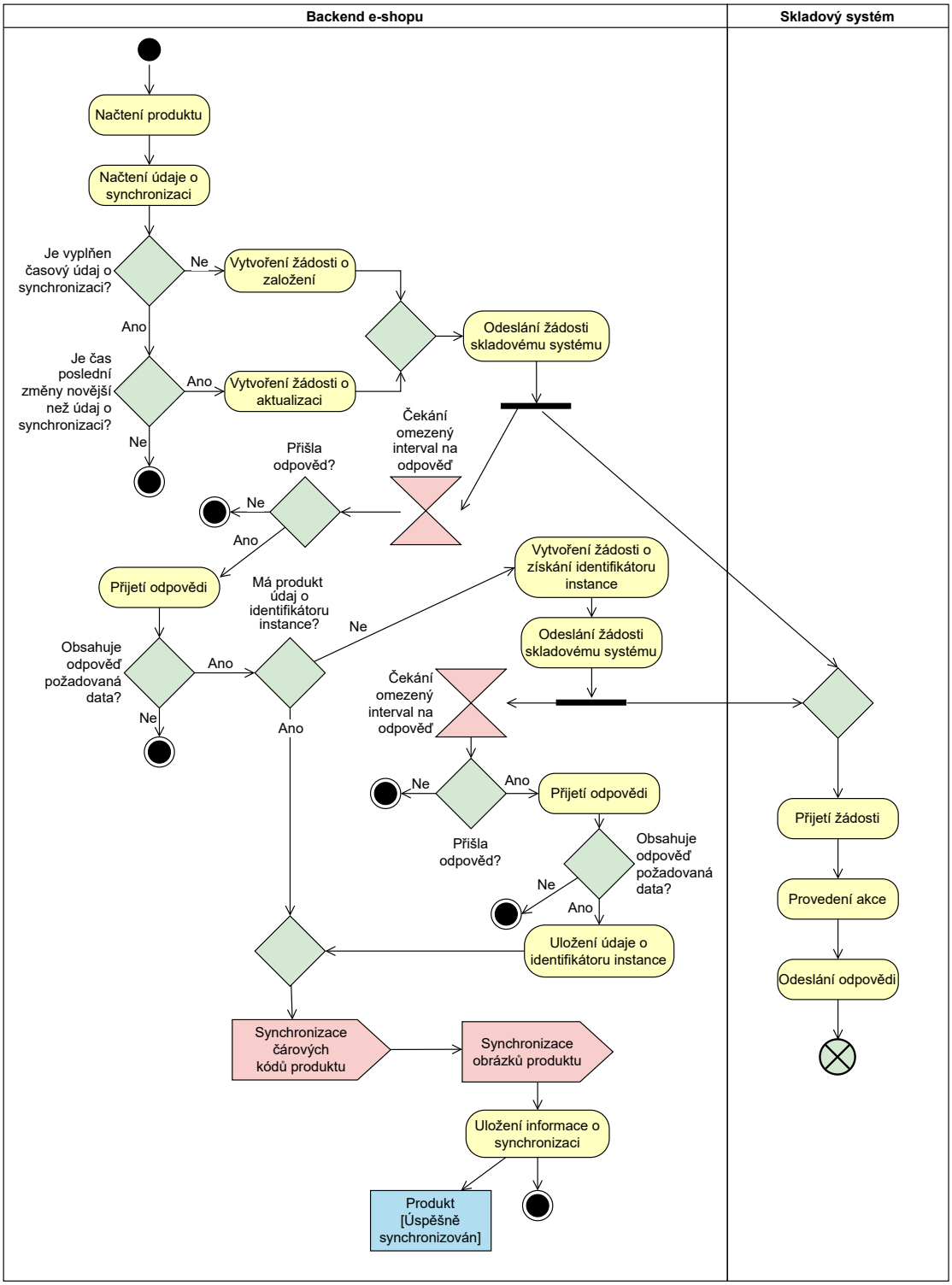
Procesy synchronizace každého výrobce i produktu je popsány pomocí diagramů aktivit 2.3 a 2.4. Backend e-shopu načítá maximálně 50 produktů najednou, aby se minimalizovala doba čekání na synchronizaci. Nové produkty a výrobci se synchronizují pomocí POST requestu a aktualizace se provádí pomocí PATCH requestu s aktuálními údaji o produktu.

Během synchronizace se také kontroluje, zda je čárový kód produktu uložen mezi čárovmi kódy pro tento produkt ve skladovém systému. Pokud není, je nově registrován. Stejně tak se kontroluje přítomnost obrázku pro produkt a případně se přidá nebo odstraní. Proces synchronizace čarových kódů je popsán diagramem aktivit 2.5, stejně tak jako synchronizace obrázků v driagramu 2.6.

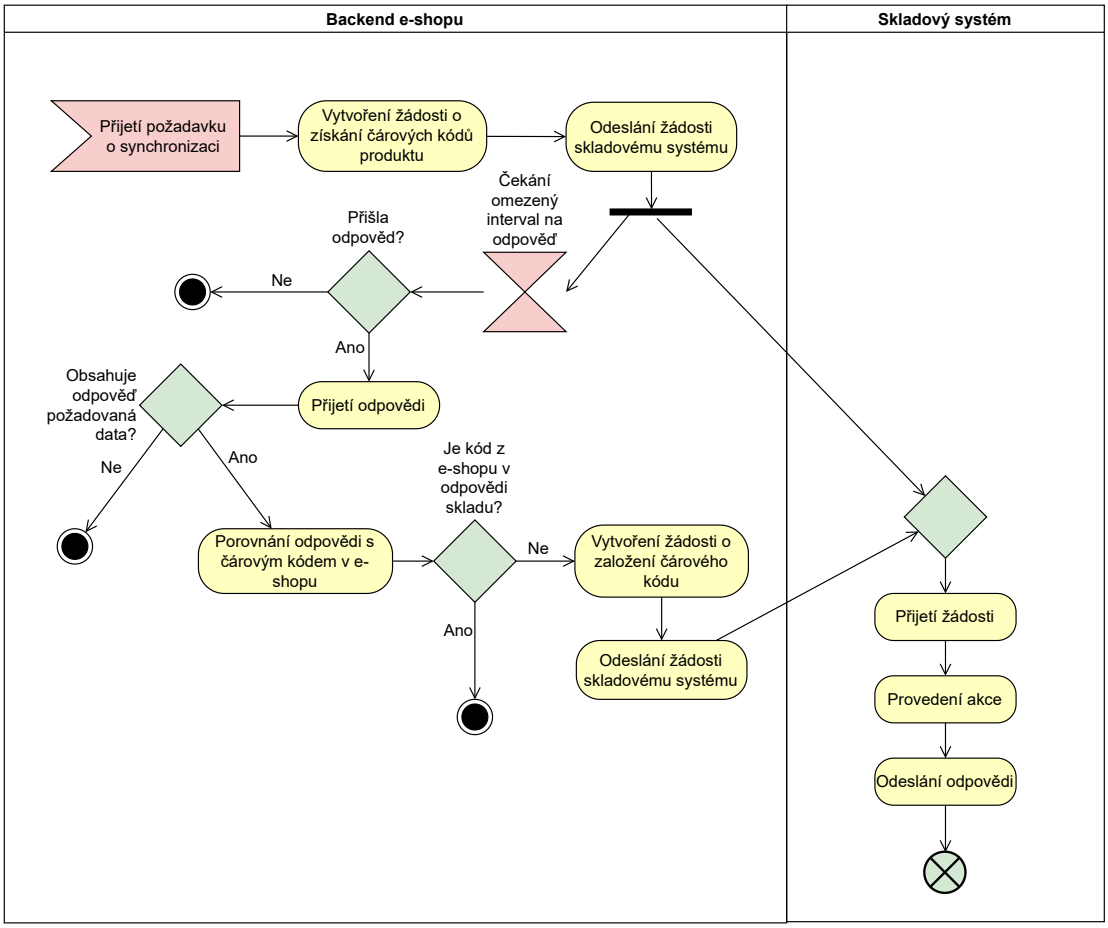
Po synchronizaci se aktualizuje údaj o poslední synchronizaci, aby se minimalizovalo zbytečné opakování synchronizace stejných produktů a výrobců.



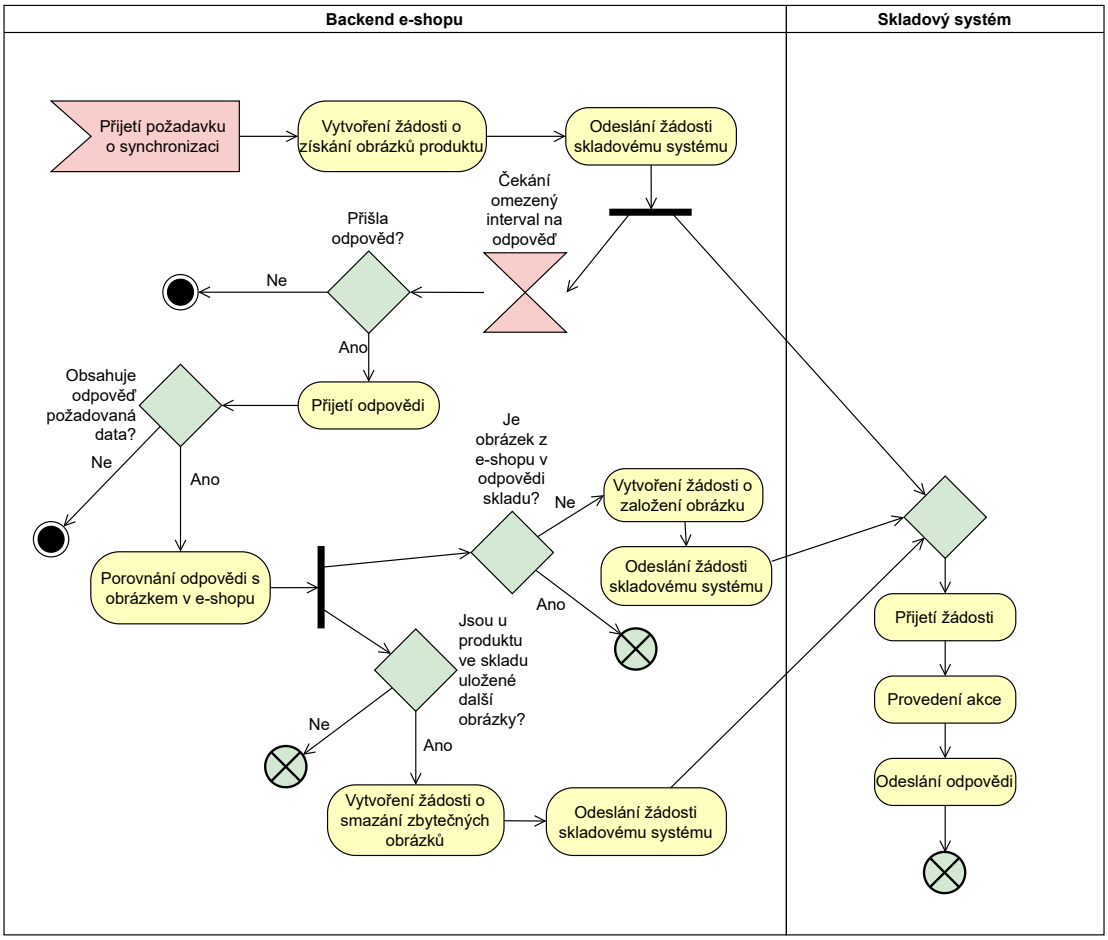
Obrázek 2.3 Activity diagram synchronizace výrobce



Obrázek 2.4 Activity diagram synchronizace produktů



Obrázek 2.5 Activity diagram synchronizace čárových kódu produktu



Obrázek 2.6 Activity diagram synchronizace obrázků produktu

2.3.2 Doprovodné pravidelné činnosti

Kromě synchronizace se skladem existují i další pravidelné činnosti, které je nutné provádět. Tyto činnosti zahrnují aktualizaci kurzu měn, kontrolu funkčnosti odkazů na YouTube, monitorování dostupnosti produktů pomocí hlídacího psa, rozesílání hodnocení a recenzí, překlad produktů, přepočet cache cen a synchronizaci s dodavateli. V následujících podkapitolách nastíním funčnost každé z těchto činností.

2.3.2.1 Aktualizace kurzu měn

Každý den o půlnoci se v e-shopu aktualizují kurzy měn. Současně se jedná o eura a polský zlotý. Kurz se aktualizuje dle aktuálního kurzu vystaveným Českou národní bankou na adrese: `http://www.cnb.cz/cs/financni_trhy/devizovy_trh/kurzy_devizoveho_trhu/denni_kurz.txt`

Ukázku výstupu lze najít v příloze: 14.

Tento výstup je třeba zpracovat, vybrat nové kurzy a nahradit jimi v databázi e-shopu jejich starý kurz.

Po získání kurzu se provádí kontola správnosti získaného kurzu. Kromě úspěchu navrácené hodnoty se explicitně kontroluje, zda získaný kurz vůči koruně spadá do očekávaného rozmezí. Pro eura je rozmezí povolených hodnot nastaveno mezi 23 a 35. Pro polský zlotý je rozmezí nastaveno mezi 5 a 10. Je však požadavek, aby v novém řešení neprobíhala taková explicitní kontrola, ale přešlo na kontrolování procentuální změny kurzu vůči předchozímu dnu. Při příliš velké odchylce by nový kurz nebyl akceptován a administrátor by byl upozorněn, aby provedl zásah.

Po kontrole se nové kurzy aktualizují v tabulce `oc_currency` včetně času poslední změny.

Nově vznikl požadavek mít alternativní možnost nastavit pro měnu fixní kurz dle vnitřního předpisu. Také je v současné administraci možnost nastavit srážku pro vybrané měny. I tuto funkcionalitu by mělo podporovat nové řešení.

2.3.2.2 Kontrola funkčnosti Youtube odkazů

U produktů na e-shopu může být přiložené video například s ukázkou produktu nebo instrukcemi k používání.

Na současné administraci funguje cron, který pravidelně kontroluje funkčnost odkazů na Youtube videa. Pokud je použité video smazáno z Youtube, je o tom emailem informován správce.

Pro tuto kontrolu se v současné administraci používá `https://noembed.com`. Tato služba nabízí jednotné url pro získání obsahu s podporou formátu *oEmbed* i pro stránky, které jej nenabízejí nebo jej nabízí nekonzistentně. [28]

Odesláním *GET* requestu na adresu:

`https://noembed.com/embed?url=https://www.youtube.com/watch?v= + id_videa`

získáme odpověd s informacemi o videu ve formátu JSON.

Skutečnost, že video neexistuje v tomto případě značí vyplněný atribut `error` v odpovědi.

V případě neexistence jednoho či více videí backend odešle email dle konfigurace na emailovou adresu správce se seznamem produktů, u kterých byl nalezen nefunkční odkaz.

Ukázky odpovědí pro existující i neexistující videa lze nalézt v příloze jako 16 a 17.

Tento způsob jsem otestoval a stále funguje. Nicméně je důležité mít na paměti, že YouTube neustále aktualizuje svou infrastrukturu a způsob, jakým jsou data na stránkách prezentována, může se velmi rychle změnit. V důsledku toho by mohl způsob kontroly pomocí *noembed* přestat fungovat.

Novým alternativním způsobem by mohlo být použití oficiálního placeného Google API. Existuje také možnost implementovat vlastní *webscraping* nástroj, avšak toto řešení může být rychle zastaralé a nepoužitelné vzhledem k neustálým změnám na stránkách YouTube.

2.3.2.3 Hlídací pes

E-shop nabízí zákazníkům vytvořit si u skladem nedostupných produktů „hlídacího psa“ *(anglicky watchdog)*. Tato funkce informuje zákazníka emailem o dostupnosti při naskladnění vybraného produktu.

Funkci „hlídacího psa“ obsluhuje cron, který se spouští v 0:00, 6:00, 12:00 a 18:00. Tento cron pravidelně kontroluje záznamy „hlídacích psů“ v databázi v tabulce `oc_product_watchdog` a počty naskladněných kusů hlídaných produktů z tabulky `oc_product`. Skladem jsou, pokud součet atributů `quantity` (externí sklad, u dodavatele) a `internal_quantity` (interní sklad, skladový systém) v `oc_product` je větší než 0, tedy existuje alespoň jeden naskladněný kus.

Pro takové záznamy následně e-shop sestaví odkaz na produkt na doméně, kde zákazník začal produkt sledovat, poté vytvoří emailovou zprávu a odešle jej zákazníkovi.

Produkty mohou být sdíleny mezi více e-shopy, tudíž je třeba, aby byla sestavená url pro správnou doménu.

Vytvoření emailové zprávy se skladá z přidání data vzniku „hlídacího psa“, názvu hlídaného produktu, sestavené url s odkazem na hlídaný produkt a názvu domény do předpřipravené zprávy.

Následně se musí email odeslat zákazníkovi. Je také potřeba, aby byla nastavena správná odchozí emailová adresa, konkrétně z adresy domény, kde zákazník začal produkt sledovat.

Po odeslání emailu ještě musí být nastaven čas uzavření „hlídacího psa“ v databáze, aby se odesílání emailu neopakovalo, ale také aby existoval záznam, kdy byl uzavřen.

2.3.2.4 Rozesílání žádostí o recenzi

Pokud zákazník dá souhlas s odesláním dotazníku, je mu 3 týdny po vyhotovení objednávky odeslán email s žádostí o recenzi produktů, které si zakoupil.

Tento proces zpracovává cron, který se odehrává každých 20 minut. Objednávky se souhlasem o zaslání žádosti o recenzi přidány uloženy do fronty v databázi reprezentované tabulkou `order_review_queue`.

Cron si načte záznamy v `order_review_queue`, pro které, kde je datum přidání starší než nastavený počet dní pro odeslání této žádosti. Současně je tento interval nastavený na 3 týdny, tedy 21 dní.

Pro každou objednávku, u které má být odeslán dotazník s žádostí o recenzi, se nejprve získá její identifikátor z tabulky `oc_order`. Poté se vytvoří email v jazyce zákazníka a doplní se do něj relevantní informace o objednávce, jako jsou název a doména obchodu, číslo objednávky a faktury (s odkazem na fakturu), datum vytvoření objednávky, seznam produktů (včetně názvu, modelu, počtu kusů a URL pro recenzi produktu) a logo obchodu. Tyto informace se získají z tabulek `oc_order` a `oc_product`. Poté se provede odeslání emailu.

V seznamu produktu je třeba dbát na produkt s identifikátorem 0, kde se jedná o dopravu, ne přímo o produkt na e-shopu. Dále pak vyžadují speciální zpracování i dárky.

Tento email pro zákazníka je ozdobený pomocí šablony domény, kde byla objednávka vytvořena.

Po vytvoření emailu je potřeba email odeslat z korektní domény obchodu.

Pokud se odeslání zdaří, je aktualizován stav objednávky v tabulce `oc_order` a je přidáno do historie objednávky v `order_history` nový záznam s datem odeslání a popisem *„Zákazníkovi byl odeslán e-mail s žádostí o recenzi.“*.

Pokud se odeslání nezdaří, je v historii objednávky vyvořen záznam s popisem *„Selhalo odeslání e-mailu s žádostí o recenzi zákazníkovi.“* a datem selhání. Stav objednávky v tomto případě neaktualizujeme.

Zpracované dotazníky pro objednávky jsou poté odstraněny z databáze.

2.3.2.5 Překlad produktů

V administraci lze nakonfigurovat, aby se nově přidané nebo upravené produkty a kategorie automaticky překládali do jiných jazyků. K těmto překladům se aktuálně využívá Google translate.

Pokud bude nalezen lepší způsob pro překlad, je možné využít jiné nástroje. Používá se totiž i k předkladu popisů, kde mohou nastat chyby způsobené limitací Google překladače. Je tedy často potřeba překlad ručně zkontrolovat.

Produkty k překladu se ukládají do fronty, jejich překlad následně zpracovává cron každý den o půlnoci.

U produktu je potřeba přeložit název produktu, krátký i dlouhý popis produktu a klíčová slova. Je také potřeba vytvořit novou url pro překládané produkty.

Příklad url:

Původní url produktu: `https://www.stylka.cz/pro-krasu/hrebeny-a-kartace/kartace-na-vlasy/cerny-kartac-tangle-teezer-original-panther-black`

Přeložená url ve slovenštině: `https://www.stylka.sk/krasa/hrebene-a-kefy/kefy/no-bb-011012-black-brush-original-tangle-teezer-or-bb-010119`

Obdobně je potřeba vytvořit tyto překlady pro kategorie. Překládají se opět název, popisy, url a klíčová slova. Po přeložení všechny informace uloží do databáze.

2.3.2.6 Přepočet cache cen

V obchodech se pravidelně přepočítávají cache cen produků. Tato činnost je poměrně náročná na výpočetní zdroje, spouští se tedy v různé časy pro různé domény. Celkově se spouští čtyřikrát za den pro každou doménu.

Pro každou kombinaci produktu, domény a zákaznické skupiny se vkládají informace do tabulky `oc_product_price`. Konkrétně se vybírá `product_id` z aktuálního produktu v tabulce `oc_product`, `domain` se vybírá z tabulky `oc_domain_setup` a `customer_group_id` se vybírá z aktuální zákaznické skupiny v tabulce `oc_customer_group`. Pokud existuje záznam pro aktuální zákaznickou skupinu v tabulce `oc_customer_group_discount_id`, použije se tento identifikátor, jinak se nastaví na 0.

Cena se vypočítá pomocí procedury `product_price_discount` v databázi, která vypočítá cenu a najde akce pro produkt s ohledem na prioritu domény. Následně se cena nastaví do sloupce `price` a sloupec `price_old` se nastaví na hodnotu z `price` v tabulce `oc_product`. Na konec se sloupec `modifiedon` nastaví na aktuální čas pomocí funkce `NOW()`.

Provádí se pouze pro produkty, které byly změněny a nepřepočítány, nebo pro produkty, pro které začala nebo skončila akce a ještě nebyli přepočítány.

Pokud už záznam s klíčem v tabulce existuje, změníme pouze `price` na novou vypočítanou cenu, `price_old` na starou cenu a aktualizuje datum změny.

2.3.2.7 Synchronizace s dodavateli

Jedná se o další proces synchronizace produktů v e-shopu s dodavateli, který zahrnuje automatické založení nových produktů od dodavatele v e-shopu a aktualizaci počtu kusů na externím skladě. Pro tuto úlohu se využívají služby dodavatele, který poskytuje seznam produktů ve formátu XML. Avšak, tato úloha je velmi komplexní a vyžaduje dbání na mnoho různých případů, které mohou nastat. Proto byla tato úloha odebrána z rámce této bakalářské práce po domluvě s vedoucím práce a je zde pouze zmíněna. Nastíním zde alespoň problémy odhalené analýzou, které způsobují vysokou komplexitu tohoto procesu.

Při načítání dat od dodavatele je nutné řešit různé situace. Jedním z nich jsou duplicity, které je potřeba detekovat a zabránit jejich vzniku. Dále je potřeba provádět párování na existující produkty v e-shopu. To lze provést buď podle identifikátoru uloženého v databázi, nebo podle

EAN kódu. Avšak, může nastat problém s různými modely a druhy, kdy EAN kód nemusí být jednoznačný. Dalším problémem jsou nesmyslná data, které je třeba odfiltrovat. V případě založení nového produktu je nutné načíst obrázek od výrobce. V neposlední řadě je problémem také mapování výrobců, kdy je potřeba zajistit konfigurovatelnost a umožnit přejmenovávání výrobců, odebírání některých řetězců z názvu výrobce a práci s velkými a malými písmeny.

2.3.3 Sesbírané požadavky

Na základě analýzy a konzultací se zadavatelem byly sesbírány požadavky na nově vznikající komponentu. Níže se zaměřím podrobně na jednotlivé funkční a nefunkční požadavky.

2.3.3.1 Funkční požadavky

Funkční požadavky vycházejí především z současného řešení automatizace činností e-shopu od *Jagu s.r.o.* a nových požadavků zadavatele. V požadavcích vymezuji všechny funčnosti, které jsou naléhány na systém.

FP1: Synchronizace zásob Zajistit pravidelnou synchronizaci stavu zásob produktů se skladovým systémem v intervalu každých 5 minut. Požadavek zahrnuje získání počtu volných kusů na skladě, aktualizaci interního počtu kusů na skladě v e-shopu.

FP2: Synchronizace objednávek Synchronizace objednávek mezi e-shopem a skladem v obou směrech (od e-shopu k skladu a zpět) každou minutu. Požadavek zahrnuje nové objednávky, změny a zrušení objednávek a informace o stavu produktů u objednávky. Rozlišuje se několik typů objednávek, včetně nezpracovaných, zrušených nebo resetovaných objednávek.

FP3: Synchronizace produktů Synchronizace probíhá každých 15 minut a zahrnuje nejen produkty, ale také jejich výrobce, čárové kódy a obrázky. Nové produkty, stejně tak jako výrobci, čárové kódy a obrázky, jsou založeny ve skladovém systému. U produktů a výrobců probíhají také aktualizace v případě, že sklad nemá aktuální informace. Odstraněné obrázky produktů jsou odstraňovány ze skladového systému pro šetření paměťových prostředků.

FP4: Synchronizace zásilek Synchronizace zásilek probíhá každých 15 minut. Stavy zásilek jsou rozděleny do čtyř typů a pro každý typ zásilek je prováděno několik kroků, jako aktualizace stavu objednávky v databázi a informování zákazníka o změně stavu zásilky emailem, pokud je to relevantní. Proces informování zákazníka zahrnuje načtení informací o objednávce a sestavení textu emailu. Pokud je objednávka vytvořena z Heuréky pomocí Heuréka API, systém předá informaci o změně stavu objednávky i Heuréce. Celý proces synchronizace nově probíhá přes skladový systém nezávisle na dopravcích.

FP5: Aktualizace kurzů měn Aktualizace kurzů měn probíhá každý den o půlnoci. Je zapotřebí aktualizovat kurzy měn vůči výchozí měně e-shopu na základě aktuálních kurzů poskytnutých Českou národní bankou. Aktualizované kurzy se uloží do databáze e-shopu včetně času poslední změny. Dále se požaduje možnost nastavit fixní kurz pro každou měnu a nastavit srážku pro vybrané měny. Kurzy se kontrolují na základě procentuální změny vůči předchozímu dni, a při příliš velké odchylce se nový kurz neakceptuje a administrátor je upozorněn, aby provedl zásah.

FP6: Kontrola Youtube odkazů Tento požadavek se týká ověření funkčnosti Youtube odkazů u produktů v e-shopu. Pro kontrolu je potřeba vybrat vhodnou službu, která odkaz zkontroluje. Pokud video na odkazu neexistuje, je potřeba odeslat správci emailovou zprávu se seznamem produktů, u kterých byl nalezen nefunkční odkaz.

FP7: Notifikace o dostupnosti produktů Notifikace zákazníků o dostupnosti produktů se týká monitorování pomocí funkce „hlídací pes“ na e-shopu. Je zapotřebí pravidelně kontrolovat databázi a odesílat emailové zprávy s odkazem na produkt a doménu pro každý nově naskladněný produkt. Email je zapotřebí odeslat z adresy domény, kde byl hlídací pes vytvořen, a po odeslání nastavit čas uzavření „hlídacího psa“ v databázi.

FP8: Žádost o recenzi Žádost o recenzi se zabývá procesem rozesílání žádostí o recenzi produktů zákazníkům. Pokud zákazník souhlasí při objednání z e-shopu s odesláním dotazníku, je potřeba mu 3 týdny po vyhotovení objednávky odeslat email s žádostí o recenzi produktů, které si zakoupil. Rozesílání probíhá každých 20 minut. Podle úspěchu či neúspěchu odelání emailu zákazníkovi musí být následně vhodně aktualizován stav objednávky.

FP9: Překlad produktů Funkční požadavek se týká překladu produktů a jejich popisů do různých jazyků. V administraci bude možné nastavit automatický překlad nově přidaných nebo upravených produktů a kategorií do jiných jazyků. Je potřeba vybrat vhodný nástroj pro překlad. Kromě názvu produktu je potřeba přeložit také krátký a dlouhý popis produktu a klíčová slova. Dále je zapotřebí přeložit kategorie a url adresy pro nově přeložené produkty a kategorie.

FP10: Přepočet cache cen Přepočet cache cen produktů se provádí čtyřikrát za den pro každou doménu. Pro každou kombinaci produktu, domény a zákaznické skupiny je potřeba vypočítat novou cenu. Cena se vypočítá pomocí procedury v databázi. Vypočítanou cenu je nakonec potřeba uložit do databáze.

2.3.3.2 Nefunkční požadavky

Nefunční požadavky vychází především z použitých technologií a konvencí v současném projektu.

NP1: Běh v Dockeru Zvolená technologie musí být přizpůsobena pro běh v kontejnerizovaném prostředí Docker, aby bylo možné snadno distribuovat a nasadit aplikaci.

NP2: Použití PHP a Symfony frameworku Zvolená technologie musí být použitelná v jazyce PHP s využitím Symfony frameworku, aby byla zajištěna jednotnost kódu a umožnilo se snadnější rozšiřování.

NP3: Využití Doctrine pro práci s databází Zvolená technologie musí využívat Doctrine ORM pro mapování objektů na relační databázi a usnadnění práce s ní, aby bylo možné snadno a bezpečně pracovat s databází.

NP4: Plánování v intervalu minut Zvolená technologie musí umožňovat plánování činností a to s přesností na minuty, aby bylo možné přesně naplánovat a vykonat potřebné úkoly v předem stanoveném čase.

NP5: Spouštět více činností najednou Zvolená technologie musí být schopná spouštět více činností v jednu minutu a řešit jejich vykonávání paralelně nebo postupně, aby bylo možné zvládnout zpracování velkého množství úkolů.

NP6: Ukládání činností do fronty Zvolená technologie musí umožnit uložení plánovaných činností do fronty v případě, že nejsou dostupné potřebné zdroje pro jejich okamžité vykonání, aby bylo zajištěno, že se žádný úkol neztratí a bude vykonán v nejbližší možné době.

NP7: Komunikace pomocí HTTP protokolů Zvolená technologie musí umožňovat komunikaci s ostatními systémy pomocí HTTP protokolů, zejména se skladovým systémem, aby bylo možné snadno a bezpečně integrovat aplikaci s jinými systémy a využívat jejich funkce.

NP8: Snadné přídaní nových činnosti Zvolená technologie také musí do budoucna umožnit snadné přidání nových činností, které nebudou implementovány v rámci této práce. Je zapotřebí, aby plánování a správa dalších přidaných činností byla dobře zdokumentována.

2.4 Nástroj pro tvoru exportů

V této sekci se zaměřím na analýzu nově vnikajícího nástroje pro tvorbu exportů. Analyzuji systémy, do kterých jsou v současné době data exportována, použité datové fomáty a prostředky pro filtrování dat. Dále se zaměřím na požadavky, které současné řešení nesplňuje, ale jsou kladeny na nově vznikající komponentu. Nakonec shrnu sesbírané funkční i nefunkční požadavky.

2.4.1 Systémy využívající datové exporty

V současné době jsou používány exporty do třech systémů. V jednotlivých podsekcích shrnu každý z jednotlivých systémů a formu příslušných exportů.

2.4.1.1 Tichý účto

Tento systém nepodporuje přímý import, je třeba vygenerovat data přímo ve formátu souboru, který program používá. Jak formát vypadá jsem byl schopný zjistit pomocí analýzy kódu v současném řešení.

Soubor se v současném backendu generuje pomocí jednoho rozsáhlého výběru nad daty v databázi. Formát souboru je v notaci *JSON*, kde každý záznam představuje jeden objekt a celkový soubor těchto objektů je pak uložen v jednorozměrném poli. Každý záznam reprezentuje právě jednu fakturu. Faktury jsou seřazeny vzestupně dle jejich identifikátorů. Výpis sloupců potřebných pro export do Tichý Účto lze nalézt v příloze: A.1

Výběr dat je specifikován jednoznačně v kódu, ale existuje možnost doplnit filtry. Filtry jsou omezení, která klademe na exportovaná data. Omezit se můžeme v tomto případě na časový rozsah faktur nebo také na druh faktury.

Filtry s názvy `since` a `until` definují rozsah kalendářních dat fakturace objednávek, které mají být exportovány.

Filtr `dataset` nabízí možnost zvolit podmnnožinu dat, ze kterých se bude exportovat. Jedná se o množiny `eshop` a `shop`. Množina `eshop` zahrnuje řádně vystavené faktury s identifikátorem začínajícím rokem. Množina `shop` zahrnuje faktury pokladny, které lze poznat podle toho, že jejich identifikátory začínají na znak „P“. Pokud není vybrána ani jedna z těchto možností, vyberou se opravné doklady, tedy faktury, jejichž idedentifikátor začínají na znak „O“.

2.4.1.2 ČSÚ

Podle zákona o statistice ([č. 89/1995 Sb.]) jsou fyzické i právnické osoby, které mají sídlo nebo obvyklé bydliště v České republice a provozují výrobu, služby nebo prodej, povinny poskytovat Českému statistickému úřadu (ČSÚ) potřebné statistické informace. To se vztahuje i na elektronické obchodníky (e-shopy), kteří splňují tyto podmínky.

I současně provozovaný e-shop má tedy právní povinnost poskytnout statistické informace ČSÚ v případě, že jsou vybrány úřadem pro vytvoření statistik. Proto dalším požadavkem pro exportovací nástroj je možnost přímého exportu dat ve formátu požadovaném statistickým úřadem.

Opět analýzou současného řešení jsem měl možnost zjistit, jak mají vypadat exportovaná data. V tomto případě data se jedná o soubor ve formátu *CSV*. V současném řešení lze přímo z databáze exportovat pro *Český statistický úřad* faktury a dluhopisy.

Faktury se exportují se pro konkrétní měsíc a rok. Informace se generují z příslušné objednávky, kde identifikátor faktury začíná rokem fakturace a datum fakturace přiléhá vybranému roku a měsíci. Tyto faktury jsou pak seřazeny vzestupně dle jejich identifikátorů.

Seznam s popisem sloupců lze nalézt v příloze: A.2

Alternativou je generovat dobropisy. Opět jsou pro vybraný rok a měsíc. Vybírají se podle sloupce `wrong_order_id`. Jejich identifikátory pak začínají na písmeno „O“ následované rokem. Exportovaná data se pak liší se od faktur pouze v několika sloupcích.

2.4.1.3 Ježek

Poslední referenčním systémem, pro který je v současné době potřeba exportovat data, je účetní systém *Ježek* od firmy *Ježek software s.r.o.*

Stejně jako v předchozím případech, současně používaný export generuje především informace o fakturách.

Tento software podporuje import ve formátu XML, je tedy potřeba data vybraná z databáze tomuto formátu přizpůsobit

Formát exportovaného XML lze nalézt v příloze: 15

2.4.2 Datové formáty v současném řešení

V této kapitole se zaměřím na datové formáty, které jsou použity v současném řešení pro export dat do systémů, kterým se věnovala předchozí kapitola v předchozí kapitole.

2.4.2.1 JSON

Zkratka pro *JavaScript Object Notation*. Je to notace pro formát typicky používaný pro přenos nebo ukládání dat. Jeho účelem je být dobře čitelný pro člověka, přitom zároveň být lehce parsovatelný a zpracovatelný pro stroje. Je založen na standartu pro programovací jazyk JavaScript vydaným v prosinci 1999. [29]

JSON je založený na dvou stukturách:

- sbírka dvojic, skládající se z klíče a hodnoty, v programování realizovaný jako objekt,
- uspořádaný seznam hodnot, v programování často přezdívaný pole (*anglicky array*).

[29]

2.4.2.2 Formát CSV

Zkratka pro *Comma Separated Values*, v překladu *Čárkou oddělené hodnoty*. Jedná se o jednoduchý a standardizovaný textový formát pro reprezentaci tabulkových dat.

Pravidla: (dle `https://opendata.gov.cz/standardy:csv`)

1. Soubor CSV má kódování UTF-8 (ve variantě bez BOM)
2. Jednotlivé řádky tabulky jsou zapsány jako řádky textového souboru oddělené pomocí CRLF (znaky s UTF-8 kódy U+000D U+000A)
3. Údaje v řádku tabulky jsou zapsány jako řetězce oddělené čárkou (znak `,` s UTF-8 kódem U+002C)
4. Pokud údaj sám obsahuje čárku `,`, nový řádek `CRLF` či uvozovku `"`, musí být uzavřen v uvozovkách (znak `"` s UTF-8 kódem U+0022)
5. Uvozovku v hodnotě je třeba zdvojit, tj. místo `"` bude zapsána jako `""`.

[30]

Na prvním řádku definujeme názvy sloupců, neboli hlavičku tabulky. Na dalších řádcích pak tabulku vyplňujeme hodnotami příslušných sloupců.

Tento jednoduchý formát umožňuje jak dobrou čitelnost pro člověka, včetně podpory v populárních tabulkových editorech jako *Microsoft Excel* či *LibreOffice Calc*, tak jednoduché parsování a zpracování počítačem.

Příklad tabulkových dat v CSV:

```
Poskytovatel,PočetDatovýchSad,PočetDistribucí
Český úřad zeměměřický a katastrální,129213,348446
Český statistický úřad,511,513
HLAVNÍ MĚSTO PRAHA,215,1294
```

[30]

2.4.2.3 XML

XML byl publikovaný v roce 1994 W3C (*World Wide Web Consortium*. Zkratka pro *Extensible Markup Language*. Jedná se o textový datový formát. Původně vytvořen aby řešil problémy s publikováním velkého objemu elektronického obsahu. V současné době se používá často pro výměnu dat mezi elektronickými systémy. [31]

2.4.2.4 Rozdíly formátů

Zatímco formát CSV je tabulkový formát, kde je potřeba dbát především na sloupce a oddělovače, formáty XML a JSON mají jasně určenou strukturu, ale je zde potřeba pracovat se zanořováním.

Zatímco formáty XML a JSON lze obousměrné převádět:

XML:

```
<Header>
  <Text>
    "Hello World!"
  </Text>
</Header>
```

JSON:

```
{
  "header" : {
    "text" : "Hello World!"
  }
}
```

Takové zanořování nelze provádět ve formátu CSV. Existují způsoby, kterými bychom mohli zanořování v CSV napodobit, ale nejsou přímo definovány samotným formátem.

Jeden ze způsobů jak bychom toho mohli docílit je transformací podatributů jako samotných názvů sloupců, například formou `Atribute/Subatribute`:

XML:

```
<Header>
  <Text>
    "Hello World!"
  </Text>
  <Text2>
    "Hi!"
  </Text2>
</Header>
```

CSV:

```
Header/Text,Header/Text2
"Hello World!","Hi!"
```

Konkrétně export dat do systému Ježek takové zanořování pro některé atributy používá. Je tedy potřeba tuto možnost vhodně navrhnout.

2.4.3 Filtrování dat

K filtrování dat jsou požity filtry. Filtry rozšiřují funkčnost komponenty pro vytvoření exportu. Při zavolání komponenty k vygenerování exportu je vedle formátu a vybíraných dat potřeba také specifikovat omezení, která se mají použít pro generování. Pro takový účel nám slouží filtry, s kterými každý export voláme.

V současném řešení lze použít několik typů filtrů, například časový rozsah dat, numerický rozsah některého z atributů, nebo splňující nějaký regulární výraz. Konkrétní export pak voláme s určenými filtry.

Například je možné specifikovat rozsah data fakturace generovaných objednávek. Dále je možné specifikovat typ faktury podle identifikátoru, konkrétně pro pokladnu, e-shop, nebo dobropisy, kde identifikátory začínají specifickým řetězcem.

V novém řešení je požadováno, aby bylo možné přidávat libovolné filtry ke konfiguracím a vhodně je předat při tvorbě exportu. Tím bude umožněna větší flexibilita při vytváření exportů a přizpůsobení se různým potřebám uživatelů.

2.4.4 Nově vznikající komponenta

Jak bylo zmíněno v předchozích kapitolách, v současném řešení jsou data pro export přímo specifikována ve výběru z databáze pouze s možností definovat některé filtry jako časové rozmezí nebo druh faktury. Výběr z databáze je napsán pomocí jazyka SQL a nachází se v kódu projektu. Pro uživatele administrace je tedy v současné době nemožné definovat si vlastní export dat.

Jedinou možností administrátora pro změnu některého s existujících exportů nebo přidání nového exportu je tedy kontaktovat správce, který pro takové úpravy musí zasahovat do kódu projektu. Pro takové úpravy je potřeba znalost jazyka projektu, v tomto případě PHP, struktury projektu a databáze, ale také znalost jazyka SQL.

V novém projektu kvůli nevýhodám současného řešení vznikl požadavek na vznik nové komponenty, která má za účel oddálit se od zmíněných nevýhod a poskytnout uživatelům administrace i správci e-shopu nové možnosti pro export dat.

Nově vzikající komponenta pro exportování dat se soustředí především na konfigurovatelnost. Konfigurovatelnost nové komponenty znamená, že umožňuje uživatelům velkou flexibilitu v nastavení exportu podle svých potřeb. To zahrnuje možnost výběru formátu exportovaných dat, exportování různých sloupců z databáze a provádění různých operací nad výběrem dat. Díky této konfigurovatelnosti je možné vytvořit exporty přesně dle potřeb administrátora nebo pro integraci s novými účetními systémy, aniž by bylo nutné zasahovat do kódu projektu nebo mít hlubší znalosti SQL.

2.4.5 Sesbírané požadavky

Na základě analýzy SQL dotazů v současném řešení, operacemi prováděnými nad daty vybranými z databáze a konzultací se zadavatelem jsem sesbíral funkční a nefunkční požadavky na nově vznikající komponentu.

2.4.5.1 Funkční požadavky

Funkční požadavky vychází především z funkcionalit poskytovaných v současném řešení pro exportování dat. Dále vychází z konzultací se zadavatelem a jeho požadavky na nově vznikající komponentu.

FP11 - Výběr formátu souboru V konfiguraci bude možné zvolit formát souboru exportovaných dat. Současně požadované formáty jsou XML, CSV a JSON. Je potřeba, aby bylo snadné rozšířit řešení o nové formáty.

FP12 - Výběr názvu atributů a podatributů V konfiguraci bude možné zvolit názvy atributů *(v případě CSV sloupců)* v exportovaném souboru. Dále bude umožněno zvolit podatributy, na které se exprtovaná data mají mapovat v případě formátu XML.

FP13 - Výběr dat V konfiguraci bude možné zvolit pro export libovolná data o objednávkách a produktech z omezené množiny tabulek. Povolené tabulky budou objednávky a produkty, ale také tabulky s instancemi produktů a popisy jednotlivých produktů.

FP14 - Operacemi nad daty V konfiguraci bude možné nastavit operace nad daty, které se načítají z databáze. Mezi požadované operace patří:

- sloučení dvou sloupců do jednoho řetězce,
- aritmetický výpočet nad daty,
- zřetězení s jiným řetězcem,
- nahrazování řetězců,
- zaokrouhlování čísel,
- porovnání hodnot,
- suma nad více sloupci,
- formátování kalendářních dat,
- přičítání dnů ke kalendářním datům,
- převedení datového typu,
- zkrácení řetězce.

FP15 - Uložení konfigurace Konfigurace bude možné ukládat v systému. Uživatel pak může konfiguraci přidat, upravit nebo odebrat.

FP16 - Filtrování dat Uložené konfigurace budou nabízet možnost doplnění filtrů následně konkrétní export.

FP17 - Seřazení dat V konfiguraci bude možné nastavit, v jakém pořadí mají být exportovaná data seřazena. Seřazovat bude možné vzestupně nebo sestupně a to i podle více atributů.

2.4.5.2 Nefunkční požadavky

Nefunční požadavky vycházejí z požitých technologií a konvencí v současném projektu, ale také z konzultací se zadavatelem.

NP9 - Omezené prostředky systému Systém má omezené výpočetní i paměťové prostředky, proto je při tvorbě exportu potřeba, aby zvolená technologie načítala z databáze pouze data potřebná pro tvorbu exportu a použít algoritmy s rozumnou asymptotickou složitostí pro operace nad daty.

NP10 - Úprava konfigurace bez zásahu do kódu Je požadováno, aby byla zvolená technologie umožňovala upravovat konfigurace v systému bez přímého zásahu do kódu projektu.

NP11 - Tvorby konfigurace bez znalosti SQL Je požadováno, aby konfiguraci bylo možné vytvořit uživatelem i bez znalosti jazyka SQL.

Kapitola 3
Návrh

3.1 Automatizace pravidelných činnosti

V této kapitole se zaměřím na automatizaci činností v aplikaci. Představím různé možnosti implementace plánování úloh pomocí utility cron. V řešení se však snažím využít moderních nástrojů, které umožní abstrahovat použitou utilitu a tím zjednodušit implementaci, čitelnost kódu, dokumentaci a testování. Proto se podívám dále na balíčky typu *Docker image* (viz sekce 3.1.2) a na balíčky do použitého PHP frameworku *Symfony* (viz sekce 3.1.3).

3.1.1 Utilita cron

Plánovač úkolů, který současný backend používá je cron. Cron je oblíbený program pro plánování úkolů v lixunových/Unix based operačních systémech. Funguje na Unixu jako *daemon*.

Démon *(anglicky daemon)* je program, který běží na pozadí bez interakce s uživatelem. Démoni jsou často spouštěni jako služby systému, aby poskytovali určité funkce, jako je například plánování úloh nebo poskytování síťových služeb. [32]

Při spuštění cron hledá crontab soubory, které si načte do paměti. Následně zkontroluje všechny načtené crontab soubory a pokud jsou nějaké úkoly, které mají být v současné minutě spušteny, vykonná je.

Samotné crontab soubory jsou jednoduché textové soubory, nebo symbolické odkazy na takové soubory. Nesmí být přístupné pro zápis nebo spuštění pro nikoho jiného než majitele. Crontab soubory obsahují instrukce pro cron. Tyto instukce se skládají z příkazů a času, v který se mají spustit. Cron hledá crontab soubory v */etc/crontab* (hlavní systémový crontab), */etc/cron.d* (složka pro ukládání dalších systémových crontabů) a */var/spool/cron* (složka pro crontaby definované uživately). [33]

Crony jsou výhodné především pro automatizování úkolů, které se mají odehrávat pravidelně na serveru, nebo jiném systému, který běží 24/7.

3.1.1.1 Syntax crontabu

Prázdné řádky, počáteční mezery a tabulátory jsou ignorovány. Řádky, které začínají znakem `#` jsou komentáře a nejsou zpracovány cronem. Komentáře nejsou povoleny na stejném řádku jako příkazy, v opačném případě by byly považovány za součást příkazu nebo nastavení prostředí.

Každý zpracovávaný řádek je buď nastavení prostředí nebo příkaz pro cron. Nastavení prostředí mají syntax `name = value`.

Name určuje název proměnné a value hodnotu. Hodnota může být umístěna v jednoduchých nebo dvojitých uvozovkách.

Některé proměnné prostředí jsou nastavené automaticky cron *daemonem*. Pro příklad `SHELL`, `HOME` a `LOGNAME`. Proměnné `SHELL` i `HOME` narozdíl od `LOGNAME` mohou být přepsány nastavením crontabu. Dále cron načítá hodnotu `MAILTO`.

Cron při provedení příkazu registruje výstup ze standardního `stdout` i chybového `stderr` a pokouší se odeslat tento výstup jako email pomocí příkazu `sendmail` na lokální emailovou adresu, pokud je definovaná a neprázdná. Tuto adresu můžeme změnit pomocí změny zmíněné proměnné `MAILTO` v crontabu. Můžeme i pracovat s výstupem. V případě zahození výstupu žádný mail odeslán není, i když je `MAILTO` definovaná.

Manuál pro Linux [34] popisuje náhradní řešení pro odesílání emailů pomocí `ssmtp` a odesílání mailů vzdálenému exchangeru. Můžeme tak okamžitě a odkudkoliv kontrolovat výsledky cronů. Host však musí nechat projít neautentifikované emaily.

Můžeme také využít proměnné `MAILFROM` pro nastavení odesílatele. V případě, že tato proměná je prázdná nebo nedefinovaná je použit odesílatel `root`.

`RANDOM_DELAY` zpožďuje čas spuštění úkolů o náhodný (za běhu cron *deamon* však konstantní) počet minut. [34]

3.1.1.2 Struktura crontabu

Řádky s příkazem pro cron obsahují pět polí značící minuty, hodiny, den v měsíci, měsíc a den v týdnu. Tyto pole určují časové podmínky, za kterých se má vykonat příkaz. Mohou být nahrazeny přezdívkami (speciálními řetězci) určenými níže. Šesté pole je pro příkaz, který se má vykonat.

Příkazy se vykonávají, když jsou splněny specifikované časové podmínky. Pokud je specifikovaný den v měsíci i den v týdnu, příkaz je spuštěn pokud splňuje alespoň jednu z těchto podmínek, nemusí splňovat obě najednou. [35]

Pro specifikaci dnů a měsíců lze použít tříznakové anglické zkratky, např. *jan* pro leden nebo *wed* pro středu. Tyto zkratky nahrazují čísla dnů a měsíců.

Pro určení časových podmínek se využívají speciální symboly:

- Symbol `*` zastupuje všechny možné hodnoty.
- Symbol `-` specifikuje uzavřený rozsah hodnot.
- Symbol `,` odděluje více hodnot nebo rozsahů hodnot.
- Symbol `/` specifikuje podmínku dělitelnosti hodnoty.

Například pro specifikaci dnů v týdnu můžeme použít `1,2` pro pondělí a úterý, nebo `0-4,8-12` pro neděli až čtvrtek a sobotu až středu následujícího týdne. Pro specifikaci opakování každé 3 hodiny můžeme použít `*/3` nebo `0-23/2` pro opakování každé 2 hodiny. [36] [34]

Jako příkaz je brán celý zbytek řádku nebo řetětězec do znaku `%`. Příkaz je vykonán pomocí */bin/sh* nebo jiným prostředím specifikovaným v proměnné *SHELL*. Syntax crontabu jde testovat pomocí přepínače -T. Ukázku crontab souboru lze nalézt v příloze: 12

3.1.1.3 Speciální řetězce pro crontab

Místo specifikace pomocí výše vysvětleného formátu lze nastavit periodicitu také přes speciální řetězce.

Zde uvádím seznam těchto řetězců:

- `@hourly`
  - příkaz se spustí jednou za hodinu, při započetí nové hodiny,

- `@daily` nebo `@midnight`
  - příkaz se spustí jednou za den o půlnoci,
- `@weekly`
  - příkaz se spustí jednou za týden o půlnoci v neděli,
- `@monthly`
  - příkaz se spustí jednou za měsíc, vždy první den v měsíci,
- `@yearly` nebo `@annually`
  - příkaz se spustí jednou za rok, vždy o půlnoci 1. ledna,
- `@reboot`
  - příkaz se spustí pouze jednou po spuštění systému.

[36, 34]

3.1.1.4 Nevýhody

Samotný cron má některé limitace. Před použitím bychom měli zvážit, zda je vhodný pro náš systém a úkoly, které v něm chceme automatizovat.

1. Nejkratší interval mezi úkoly je 60 vteřin. Nelze ho tedy použít pro častější úkoly.
2. Běží pouze na jednom systému. [33]
3. Nemá auto-retry mechanismum. Pokud úkol selže, cron ho spouští až v další plánovaný čas, nesnaží se ho spustit opakovaně. [36]

Jelikož automatizované činnosti v současném řešení využívají cron, s limitacemi minimálního intervalu a chybějícím auto-retry mechanismes nové řešení počítá. Změnou je prostředí běhu, které je nově v platformě Docker. Během cronů v tomto prostředí se zabývám níže v sekci *Cron v platformě Docker* v 3.1.2.

3.1.1.5 Alternativy

V této sekci uvádím některé z možných implementací cronu a jeho alternativy.

3.1.1.5.1 Cronie Alternativa *Cronie* obsahuje standartní UNIX cron *daemon*. Je založený na standartním vixie-cronu a má přidané bezpečnostní a konfigurační vylepšení, jako schopnout používat pam a SELinux. Také v sobě má zabudovaný anacron. [37]

3.1.1.5.2 Anacron Pomocí anacronu lze plánovat úkoly podobně. Narozdíl od cronu se nejedná o *daemon* a neběží neustále na pozadí systému. Anacron nepředpokládá, že systém poběží 24/7, a ukládá si časové známky pro úkoly ve dnech. Pokud máme např. nastavenou periodicitu pro úkol na 7 dní a spustíme systém s anacronem, anacron zkrontoluje, zda tento úkol již byl v posledních 7 dnech vykonán. Pokud ne, spustí tento úkol a po vykonání uloží novou časovou známku. U úkolů lze nastavit zpoždění v minutách. Když anacron zjistí, že má být úkol spuštěn, spustí ho po tomto zpoždění. Jeho velkou nevýhodou je však nejnižší periodicita ve dnech. [38, 39]

3.1.1.5.3 Fcron Plánovač periodických příkazů, který se snaží nahradit vixie-cron a implementuje většinu jeho funkcionalit. Narozdíl od původního cronu však fcron nepožaduje, aby systém běžel 24/7. Vedle plánování podle podmínek času, dne nebo měsíce umí plánovat na základě času stráveného během systému. Např. spustí příkaz každé tři hodiny běhu systému od spuštění. Dále umí naplánovat spuštění příkladu jednou za určený interval, za předpokladu, že v tom intervalu bude systém spuštěn.

Fcron umožňuje konfiguraci několika funkcí, jako například spouštění úkolů paralelně nebo po sobě, stanovení maximální průměrné zátěže systému jako podmínky pro spuštění úkolu, spuštění úkolů, které měly být vykonány během doby, kdy nebyl systém spuštěný, informování uživatele pomocí emailu o nespuštění úkolu a důvodu proč nebyl spuštěn, a vylepšenou správu odesílání emailů s výstupy. Tyto funkce poskytují uživatelům možnost přizpůsobit si chování fcronu podle jejich potřeb a optimalizovat tak plánování úkolů v jejich systému. [40]

3.1.1.5.4 systemd/Timers Plánování úkolů na systémech s *systemd* lze provést pomocí *systemd.timer*. Jedná se o často používanou alternativa *cronu*. Pro spuštění úkolů využívá *systemd.service*. Zmíněný *systemd.timer* specifikuje časové podmínky pro provedení, přičemž nejnižší jednotkou jsou mikrosekundy.

Hlavní výhodou *systemd.timer* je větší konfigurovatelnost oproti cronu. Lze například konfigurovat prostředí, ve kterém se úkol spustí, používat čas od spuštění systému pro vykonání úkolu, a nastavit spuštění úkolu při startu systému, pokud byl vynechán úkol v době, kdy systém neběžel. Navíc průběh úkolu se zaznamenává do logů.

Nevýhodou *systemd.timer* je, že je limitován na systémy běžící na *systemd*, což často nezahrnuje kontejnerizovaná prostředí. Dále nemá zabudovanou funkci pro odesílání emailů s výstupy úkolů, stejně jako cron. [41]

3.1.2 Cron v platformě Docker

Docker image je balíček obsahující všechny potřebné soubory, konfigurace a závislosti potřebné k vytvoření a spuštění aplikace v Docker kontejneru. *Docker* image může být vytvořen pomocí Dockerfile, což je textový soubor obsahující instrukce k vytvoření image. *Docker image* může být také stažen z Docker registru, který obsahuje předpřipravené images. *Docker image* se používá pro snadné a rychlé nasazení aplikace v různých prostředích, a to bez nutnosti instalovat a konfigurovat všechny potřebné závislosti ručně. [42]

Cron lze v Dockeru nainstalovat a spustit v konjtejneru pomocí]*Docker image* některé z Unix distribucí.

Alpine Linux Přichází s předinstalovaným crond

Debian/Ubuntu Cron lze nainstalovat pomocí apt utility

Centos Cronie lze nainstalovat pomocí yum utility [43]

PHP Docker image podporuje instalování balíčků pomocí utilit apt (Debian) a apk (Alpine).

V nové administraci se pro běh PHP používá Ubuntu image. Pro minimalizaci velikosti virtualizovaného prostředí a šetření jiných výpočetních zdrojů by bylo pro projekt přínosné použít Ubuntu i pro správu cronů oproti jiným.

Výhodou Alpine je již předinstalovaný crond, ale samotný běh nové image s dalším operačním systémem by bylo zbytečně náročné na zdroje.

Centos v tomto případě není příliš vhodný, jelikož v tomto projektu sdílí nevýhody Ubuntu i Alpine Linux.

3.1.3 Balíčky poskytující utilitu cron

V rámci rešerše jsem nalezl některé balíčky, které by se daly pro potřeby této práce použít:

- Zenstruck schedule-bundle `https://github.com/zenstruck/schedule-bundle`
- Opsxcq tasker `https://github.com/opsxcq/tasker`
- Symfonycorp croncape `https://github.com/symfonycorp/croncape`

3.1.3.1 The ScheduleBundle

Balíček *zenstruck/schedule-bundle* do frameworku Symfony slouží k plánování úkolů podporující příkazy, bash skripty i volání funkcí uvnitř Symfony aplikace.

Umožňuje definovat úkoly přímo v kódu. Na serveru se přidá jediný cron záznam do crontabu `php bin/console schedule:run`, který se spustí každou minutu. Tento cron se stará o pravidelné spouštění našich úkolů definovaných pomocí bundle.

Samotný úkol pak můžeme definovat jako PHP třídu implementující některý z typů úkolů poskytovaných tímto bundle.

Příklad samoplánujícího příkazu, vypisujicí `Hello world!` na výstup uvádím v ukázce 1.

Při běhu systému lze pomocí příkazu `php bin/console schedule:list` lze vypsat seznam úkolů a informace o případných chybách, které nastali při běhu. V ukázce 2 lze nalézt výpis příkazu.

S pomocí přepínače `--detail` lze získat i detailnější informace plánovaných úkolech. Nabízí tím užitečné rozhraní pro správu, testování a údržbu plánovaných procesů. [44]

3.1.3.2 Tasker

Jednoduchý *Docker image* pro plánování úkolů pomocí cronu. Podporuje spouštění příkazů a bash skriptů, nemá však přímou podporu pro volání funkcí v Symfony aplikaci. Toho ale můžeme docílit pomocí *Sympfony CLI commands*.

Periodicitu úkolů můžeme nastavit pomocí klasické syntaxe crontabu nebo ji také lze nastavit pomocí `every: <interval>`. Lze jednoduše konfigurovat odesílání emailů s výstupy úkolů, proměnné prostředí nebo co se má stát po vykonání úkolu v případě chyby či úspěchu. [45]

3.1.3.3 Symfony CLI Command

Umožňuje volat funkce Symfony aplikace pomocí příkazů z konzole.

Stačí umístit PHP třídu rozšiřující `Symfony\Component\Console\Command\Command` do složky `App\Command`

Název příkazu definujeme pomocí `protected static $defaultName = 'app:name';`

Nastavení příkazu jako popis či možnosti nastavujeme pomocí metody *configure* jako v ukázce 3 nebo pomocí anotací.

Samotný kód při spuštění příkazu píšeme do metody s názvem „execute“. Jak vypadá tato metoda lze nalézt v ukázce 4. [46]

```php
/**
 * Command to automatically say hello world
 */
#[AsCommand(
name: 'app:say-hello-world',
description: 'The command to say hello world.',
hidden: false
)]
class SayHelloWorldCommand extends Command implements SelfSchedulingCommand
{
    /**
     * @param InputInterface $input
     * @param OutputInterface $output
     * @return int
     * @throws Throwable
     */
    protected function execute(InputInterface $input,
        OutputInterface $output): int
    {
        $output->writeln('Hello World!');
        return Command::SUCCESS;
    }

    public function schedule(CommandTask $task): void
    {
        // run every minute
        $task->everyMinute();
    }
}
```

Výpis kódu 1 Ukázka automatizované činnosti pomocí The ScheduleBundle

```
root@84cdc44f9901:/var/www/api# php bin/console schedule:list

2 Scheduled Tasks Configured
============================

 ------------- ---------------------- ------------ --------------------------
  Type          Description            Frequency    Next Run
 ------------- ---------------------- ------------ --------------------------
  CommandTask   app:say-hello-world    * * * * *    2022-10-31T18:29:00+01:00
  CommandTask   app:update-currencies  00 1 * * 1   2022-11-07T01:00:00+01:00
 ------------- ---------------------- ------------ --------------------------

 ! [NOTE] For more details, run php bin/console schedule:list --detail

 [OK] No schedule or task issues.
```

Výpis kódu 2 Výpis seznamu úkolu v The ScheduleBundle

```php
protected function configure()
{
        $this->setDescription('description')
            ->addOption('option', null, InputOption::VALUE_NONE, 'Option');
        // ...
}
```

Výpis kódu 3 Ukázka metody configure

```php
protected function execute(InputInterface $input,
    OutputInterface $output): int
{
    $io = new SymfonyStyle($input, $output);
    // execute code/callbacks
    $count = countingCallback(...);
    // write to output
    $io->success(sprintf('Counted "%d".', $count));
    return 0;
}
```

Výpis kódu 4 Ukázka metody execute

3.1.4 Zvolené technologie

Po provedení rešerše jsem se rozhodl použít kombinaci The ScheduleBundle a Symfony Command, protože nabízí mnoho výhod. Zároveň kombinace těchto balíčků splňuje veškeré požadavky, které jsou na zvolenou technologii kladeny.

Jednou z hlavních výhod je přehledné a čitelné řešení, které umožňuje jednoduchou správu činností pomocí příkazu `php bin/console schedule:list`. Tento příkaz zobrazuje historii běhu, včetně informací o neúspěšných pokusech, což usnadňuje detekci problémů. Dále vypisuje chybové hlášky, které usnadňují následné řešení. Díky tomu je tento nástroj velmi spolehlivý.

Další výhodou je flexibilita, kterou The ScheduleBundle nabízí. Lze plánovat různé úlohy v různých časových intervalech a snadno je přizpůsobit potřebám aplikace. Balíček je také velmi snadno implementovatelný do Symfony aplikace a použití je velmi intuitivní. Kvalitní dokumentace pak pomáhá při implementaci a použití.

V neposlední řadě je důležitá rozšiřitelnost. Díky této kombinaci lze přidat vlastní plánovače úloh, které rozšíří možnosti plánování úloh dle specifických potřeb aplikace. Využití Symfony Command pro spuštění činností umožnuje náhradu The ScheduleBundle za jiný balíček a to bez rozbití funčnosti samotného příkazu.

Plánování úloh se provádí na pozadí aplikace, což minimalizuje vliv na výkon běžící aplikace a zajišťuje efektivní využití zdrojů.

Všechny tyto výhody dělají z The ScheduleBundle a Symfony Command vynikající nástroje pro automatizaci činností v Symfony aplikaci.

3.2 Nástroj pro exporty

V této kapitole se zaměřím na návrh nástroje pro exporty. Nejprve navrhnu konfiguraci nástroje, následně navrhnu API pro práci s uloženými konfiguracemi a generování exportů. Nakonec zvolím technologie pro implementaci navrženého nástroje.

3.2.1 Návrh JSON konfigurace podle funkčních požadavků

3.2.1.1 FP1 - Výběr formátu souboru

Formát souboru je určen pomocí klíče „exportFormat“ v konfiguračním objektu. Momentálně podporované formáty jsou CSV, JSON a XML.

Ukázka JSON souboru:

```
{
    "exportFormat": "csv",
    ...
}
```

3.2.1.2 FP2 - Výběr názvu atributů

Názvy atributů mohou být specifikovány pro každý sloupec pomocí klíče „name“.

Ukázka JSON souboru:

```
{
    ...
    "columns": [
        {
            "type": "string",
            "name": "greeting",
            "string": "Dobrý den"
        },
        ...
    ],
    ...
}
```

3.2.1.3 FP3 - Výběr dat

Data k exportu jsou definována v poli „columns“. Každý sloupec obsahuje informace o názvu sloupce, třídě a názvu atributu, který má být exportován. Třídy reprezentují jednotlivé datové tabulky, na které jsou mapované pomocí Doctrine ORM. Současně povolené jsou třídy *Order*, *OrderProduct*, *Product* a *ProductDescription*. V rámci těchto tabulek jsou povoleny všechny jejich mapované sloupce.

Ukázka JSON souboru:

```
{
    "columns": [
        {
            "type": "normal",
            "name": "id",
            "class": "Order",
            "columnName": "orderId",
        },
        ...
    ],
    ...
}
```

3.2.1.4 FP4 - Operace nad daty

Operace nad daty se provádějí na úrovni jednotlivých sloupců. Výběr operace je na základě atributu „expressionType“. Operace jsou následně definovány v atributu „expression“. Podporované operace jsou:

concat Sloučení dvou sloupců do jednoho řetězce.

arithmetic Základní aritmetické operace (+, -, *, /).

dateformat Formátování kalendárních dat.

dateAdd Přičítání dnů ke kalendářním datům.

case Výsledek na základě porovnávání hodnot.

round Zaokrouhlení čísla.

coalesce Výběr první existující hodnoty.

cast Převedení datového typu.

sum Suma nad více sloupci.

select Zanořený výběr sloupců

left Zkrácení řetězce

replace Nahrazení v řetězci.

Ukázka JSON souboru:

```
{
    ...
    "columns": [
        {
            "type": "expression",
            "name": "customerName",
            "class": "Order",
            "expressionType": "concat",
            "expression": [
                {
                    "type": "normal",
                    "name": "customer name",
                    "class": "Order",
                    "columnName": "firstname",
                },
                ...
            ]
        },
        ...
    ],
    ...
}
```

3.2.1.5 FP5 - Uložení konfigurace

Konfigurace může být uložena do souboru v JSON formátu v systému pomocí komponenty pro práci se soubory implementované Aloisem Koubou a Tomášem Hojkem nebo v databázi.

Současná databáze je relační a nepodporuje uložení JSON objektů. V případě, že by konfigurace byla uložena v databázi, je možné ji uložit jako string s celou konfigurací. Samostné zpracování JSON jako řetězce by pak prováděla samotná exportovací komponenta.

Při uložení jako souboru v systému můžeme uložit jako JSON soubor, ale nastává zde stejný problém, kdy zpracování samotného JSON musí provádět komponenta.

Pokud by byla uložena v databázi, navrhuji vytvořit tabulku `oc_export_config`, která bude vypadat následovně:

export_id int, primary key	export_name string	export_json_config string

Aby bylo umožněno uživatelům pracovat s uloženými konfiguracemi, bude backend nabízet frontendu e-shopu REST API, přes které bude moci uživatel zobrazovat, přidávat, upravovat a odebírat uložené konfigurace.

Samotný export pak bude nabízet REST API, kterému bude předán název konfigurace a příslušné filtry. Více API rozebírám v následující kapitole 3.2.2.

3.2.1.6 FP6 - Filtrování dat

Uložené konfigurace mohou obsahovat specifikace podmínek pro filtrování dat. Tyto podmínky jsou specifikovány v poli „conditions“. Níže je ukázka konfiguračního souboru s filtrováním podle časového rozsahu pomocí atributů `dateFrom` a `dateTo`.

```
{
    ...
    "conditions": [
        {
          "column" : "dateAdded",
          "class" : "Order",
          "conditionType" : "higherOrEqual",
          "conditionExpression" : ":dateFrom"
        },
        {
          "class" : "Order",
          "conditionType" : "lowerOrEqual",
          "column" : "dateAdded",
          "conditionExpression" : ":dateTo"
        }
    ],
    "filters" : [
        {
            "filterParameter" : "dateTo"
        },
        {
            "filterParameter" : "dateFrom"
        }
    ],
    ...
}
```

3.2.1.7 FP7 - Seřazení dat

Data mohou být seřazena podle více atributů. Seřazení je definováno v poli „orderBy“. Níže je ukázka konfiguračního souboru, který řadí data podle id objednávky vzestupně:

```
{
    ...
    "orderBy" : [
        {
            "class" : "Order",
            "column" : "orderId",
            "order" : "asc"
        },
        ...
    ],
    ...
}
```

3.2.2 Návrh API pro práci s exporty

V této sekci je popsáno navržené API pro práci s exporty. API obsahuje šest různých endpointů, z nichž každý slouží k jinému účelu.

`GET /exports` vrátí seznam všech existujících konfigurací pro export dat. Vrácená data obsahují identifikátory a názvy exportů v podobě pole, kde každý prvek obsahuje identifikátor a název, například:

```
{
    "exportId": 1,
    "exportName": "ČSÚ faktury"
}
```

`GET /exports/{exportId}` vrátí konfiguraci pro export dat s daným `exportId`. Kromě názvu a JSON konfigurace obsahuje i název a identifikátor vlastního exportu.

`POST /exports` přidá novou konfiguraci pro export dat. Při vytváření nové konfigurace je potřeba do těla požadavku doplnit „exportName“ s názvem exportu a „exportJsonConfig“ s řetězcem s JSON konfigurací. Ukázka dat, které by mohly být použity pro vytvoření konfigurace, lze nalézt v příloze 18.

`PUT /exports/{exportId}` upraví konfiguraci pro export dat s daným `exportId`. Při úpravě konfigurace je potřeba do těla požadavku, stejně jako při vytvoření konfigurace, doplnit „exportName“ s názvem exportu a „exportJsonConfig“ s řetězcem s JSON konfigurací.

`DELETE /exports/{exportId}` smaže konfiguraci pro export dat s daným `exportId`. Pokud data neexistují nebo jsou úspěšně smazána, API vrátí status code 204 No Content.

`GET /exports/{exportId}/generate` vygeneruje soubor s exportovanými daty podle konfigurace a předaných filtrů. Při generování dat je potřeba do těla požadavku doplnit pole filtrů, pokud je daný export požaduje.

Filtry pro vygenerování exportu jsou posílány jako parametry URL požadavku. Ty jsou součástí řetězce celé URL adresy. Tyto parametry jsou odděleny od základní adresy otazníkem `?` a jednotlivé parametry jsou odděleny ampersandem `&`.

Každý parametr má jméno a hodnotu, které jsou odděleny znakem rovnosti `=`. Například, pokud bychom chtěli přidat filtr na název exportu, kde hledáme exporty s názvem „sales“, by URL parametr vypadal následovně: `?exportName=sales`. Kompletní řetězec s filtry by pak mohl vypadat například takto:

```
?exportName=sales&startDate=2023-04-01&endDate=2023-04-30
```

3.2.3 Zvolené technologie

Zvolené technologie byly vybrány s ohledem na požadavky na projekt a na moje zkušenosti a znalosti v oblasti vývoje webových aplikací. Doctrine ORM bylo vybráno jako řešení pro práci s databází, protože umožňuje snadnou a efektivní práci s daty a zajišťuje bezpečnost aplikace pomocí předpřipravených SQL dotazů. Také bylo již využité v projektu, což zvyšuje efektivitu vývoje.

Byla zvolena kombinace technologií QueryBuilder a DQL pro sestavování dotazů v řešení. Tyto nástroje umožňují snadné a flexibilní konfigurace dotazů nad databází. Kromě toho byl vyvinut vlastní parser pro sestavování jednotlivých DQL dotazů, který byl speciálně navržen pro tento projekt a zajišťuje schopnost vytvářet složité dotazy podle požadované konfigurace. Tato kombinace technologií umožňuje sestavovat i složité dotazy a zaručuje, že budou odpovídat přesně definovaným požadavkům.

Pro ukládání a zpracování souborů v tomto projektu byla zvolena kombinace různých technologií a knihoven. Pro zajištění rychlosti a efektivity byla zvolena základní knihovna PHP pro práci se soubory. Dále byla použita třída StreamedResponse pro přenos souborů zpět k uživateli, což zajišťuje rychlou a plynulou komunikaci mezi serverem a klientem.

Kromě toho byl vyvinut vlastní parser pro sestavování souborů, který byl speciálně navržen pro tento projekt a splňuje specifické požadavky na sestavení a zpracování souborů. Parser umožňuje konfiguraci jako zobrazení hlavičky CSV, výběr oddělovače CSV a také poskytuje možnost mapování dat vybraných z databáze do podatributů XML. Celkově byla tato kombinace technologií zvolena s cílem dosáhnout rychlého a bezpečného vytvoření souboru a následné odeslání uživateli.

Kapitola 4
Implementace

Tato kapitola je rozdělena do třech částí. První z nich je 4.1, která se věnuje implementaci automatizovaných činností. Druhá část 4.2 se poté věnuje implementaci nástroje pro generování exportů. Veškerá implementace vznikala ve verzovacím systému použitým v rámci projektu, kterým je GitLab.

Automatizované činnosti byly implementovány uvnitř vývojové větve *„cron-dev“*. Nástroj pro generování exportů byl následně implementován ve vývojové větvi *„export-tool-dev“*.

Poslední část 4.3 se věnuje problémům, na které jsem během implementace narazil, ale také způsobům, jakými byly tyto problémy řešeny.

4.1 Implementace automatizovaných činností

V této kapitole popisuji typickou implementaci jedné z tříd, která je odpovědná za proces automatizované činnosti. Níže uvedené ukázky kódu s jejich rozborem popisují implementovanou třídu `UpdateExchangeRate`, která aktualizuje kurz měn.

Třída je implementována jako Symfony Console Command a také jako SelfSchedulingCommand, což umožňuje spouštění příkazu automaticky na základě plánu.

Pomocí anotací v `#[AsCommand(...)]`, jak vidíme v ukázce 5, nastavujeme název, popis a viditelnost příkazu. Tento příkaz můžeme volat v rámci Docker kontejneru PHP pomocí `php bin/console name`, kde `name` představuje název uvedený v anotaci, v případě této ukázky se jedná o `app:update-currency`. Zavoláním tohoto příkazu náš příkaz spustíme, konkrétně metodu `execute()`.

```php
/**
 * Command to update currency exchange rate
 */
#[AsCommand(
    name: 'app:update-currency',
    description: 'The command to update currency exchange rate.',
    hidden: false
)]
class UpdateExchangeRate extends Command implements SelfSchedulingCommand
```

Výpis kódu 5 Ukázka anotací a definice třídy

V konstruktoru třídy lze nastavit potřebné závislosti na jiné třídy v naší aplikaci, lze tak nastavit závislosti na službách *(services)* a repozitářích *(repositories)*, které pro vykonání činnosti potřebujeme. V případě akutalizace kurzu měn je použitá třída `CurrencyService` pro načítání a akutalizaci měn, dále `SettingRepository` pro získání různých nastavení e-shopu a nakonec `LoggerInterface` pro účely logování průběhu příkazu. Ukázku konstruktoru můžeme nalézt níže ve výpisu 6.

```
/**
 * @param CurrencyService $currencyService
 * @param SettingRepository $settingRepository
 * @param LoggerInterface $logger
 */
public function __construct(private readonly CurrencyService $currencyService,
                            private readonly SettingRepository $settingRepository,
                            private readonly LoggerInterface $logger)
    {
        parent::__construct();
    }
```

Výpis kódu 6 Ukázka konstruktoru

Povšimnout si lze také použité syntaxe pro nastavení závislostí. Jedná se o způsob přidaný do PHP ve verzi 8, kde bylo umožněno definovat proměné přímo uvnitř parametrů konstruktoru. Dále bych podotknul použití vlastnosti `readonly`, která byla přidána do PHP ve verzi 8.1. Tato vlastnost znemožňuje měnit obsah inicializovaných proměných.

Kód, který se vykoná při spuštění příkazu, je obsahem zmíněné metody `execute()`. Tato metoda má dva povinné parametry, `InputInterface` a `OutputInterface`. Tyto parametry jsou nastaveny samy prostředím, pokud příkaz spouštíme pomocí `php bin/console` i pokud je příkaz spuštěn automaticky pomocí plánovače. V ukázce 7 lze vidět implementaci této metody pro aktualizaci kurzů měn.

V této ukázce je představen průběh aktualizace kurzů. Metoda si stáhne aktuální kurz měn z webové stránky České národní banky. V rámci metody `updateCurrencyRate` je získán kurz z textové odpovědi webové stránky pomocí parseru, získaný kurz je následně porovnán s kurzem uloženým v databázi. Pokud je změna kurzu menší než 10 %, je nový kurz uložen do databáze. Poté se zapisuje informace o tom, zda byla aktualizace úspěšná nebo selhala.

Metoda `schedule()` nastavuje plán, podle kterého se příkaz spouští. Tato metoda má jeden parametr typu `CommandTask`. Předanému parametru pomocí některé z nabízených metod nastavíme jeho pravidelnost.

Třída nabízí mnoho metod, pomocí kterých lze nastavit pravidelnost příkazu, alternativně lze použít i syntax, který je použitý pro plánování příkazů v souboru crontab. S přesností na minuty tedy lze nastavit libovolnou pravidelnost.

Jak je vidět v ukázce 8, příkaz pro aktualizaci kurzů měn se spouští každý den o půlnoci. Toho je docíleno pomocí metody `daily()`.

```php
/**
 * @param InputInterface $input
 * @param OutputInterface $output
 * @return int
 * @throws Throwable
 */
protected function execute(InputInterface $input, OutputInterface $output): int
{
    // Create a new HTTP client
    $client = HttpClient::create();
    // Fetch the exchange rate data from the CNB website
    $response = $client->request('GET',
        'https://www.cnb.cz/cs/financni_trhy/devizovy_trh/' .
        'kurzy_devizoveho_trhu/denni_kurz.txt');
    $content = $response->getContent();

    $currencies = $this->currencyService->getAllCurrencies();

    // Update each currency
    foreach ($currencies as $currency){
        $this->updateCurrencyRate($content, $currency);
    }

    return Command::SUCCESS;
}
```

Výpis kódu 7 Ukázka metody execute

```php
/**
 * Runs every midnight
 * @param CommandTask $task
 * @return void
 */
public function schedule(CommandTask $task): void
{
    $task->daily();
}
```

Výpis kódu 8 Ukázka metody schedule

4.2 Implementace nástroje pro exporty

V této kapitole popíšu proces implementace nástroje pro exportování dat z aplikace. V první podkapitole se zaměřím na sestavení dotazu, který nám umožní získat data, která chceme exportovat. Následně se budu věnovat vytvoření souboru s exportovanými daty v různých datových formátech. V další podkapitole se budu zabývat implementací rozhraní, které bude umožňovat uživatelům nastavit parametry exportu a spustit ho. V poslední podkapitole budu diskutovat problémy, které jsem při implementaci nástroje řešil, a popíši způsoby, jakými jsem je vyřešil. Celkově tato kapitola popisuje kompletní proces vytvoření funkčního nástroje pro export dat z aplikace.

4.2.1 Sestavení dotazu

Tato podkapitola se zabývá implementací třídy GenericExportTool, která slouží jako parser pro sestavení dotazu pomocí QueryBuilderu. QueryBuilder je objektový návrhový vzor pro sestavování SQL dotazů. Implementovaná třída obsahuje metodu `exportReport`, která generuje export dat na základě JSON konfigurace.

Třída GenericExportTool umožňuje specifikovat sloupce, primární třídu, podmínky, spojené třídy a řazení výstupních dat. Tyto vstupy jsou přijímány v podobě pole JSON dat. Třída pak sestavuje dotaz na základě těchto vstupů a vrací výsledky dotazu jako pole.

Metoda `exportReport` začíná tím, že resetuje čítač aliasů. Poté načte vstupní JSON data a uloží je do příslušných proměnných. Následně vytvoří QueryBuilder objekt a přidá sloupce k dotazu.

Podle typu sloupce se pak do dotazu přidávají různé výrazy, které mají být vybrány, pomocí metody `addSelect`. Zde se kontroluje typ sloupce, zda se jedná o normální sloupec, řetězec, číslo nebo výraz.

V případě, že se jedná o sloupec tabulky, přidá se do dotazu podle názvu sloupce a třídy pomocí metody QueryBuilderu `addSelect`. Pokud by nějaká informace chyběla, vyvolá se výjimka. Obdobné je přidání sloupce, pokud se jedná o řetězec nebo číslo, který se také přidává se přímo do dotazu pomocí metody `addSelect`.

Komplikovanější situace nastává, pokud se jedná o výraz. Pro podporu zanořování a vyřešení problému s chybějícími SQL funkcemi v Doctrine ORM, kterému se věnuji podrobně níže v kapitole *Problémy s implementací* 4.3 , jsem se rozhodl vytvořit vlastní metodu pro vygenerování DQL, které je následně přidáno do dotazu.

DQL (Doctrine Query Language) je jazyk pro sestavování dotazů, který se používá pro objektově-relační mapování (ORM). DQL umožňuje psát dotazy na entitní objekty a jejich vlastnosti, ačkoliv se v pozadí jedná o SQL dotazy. DQL také podporuje pokročilé funkce, jako jsou agregace, spojování tabulek, řazení, omezení a mnoho dalšího. [47]

Implementovaná metoda `getExpressionDQL` umožňuje sestavit DQL podle typu výrazu, který je definován v parametru `expressionType`. Například pokud se jedná o aritmetický výraz, pak se volá metoda `getArithmeticDQL`, která sestaví DQL pro tento typ výrazu.

Metoda umožňuje pracovat s různými typy výrazů a dokáže zanořovat výrazy, čímž umožňuje vytvářet i složitější dotazy. Pokud se v parametru `expressionType` objeví neznámý typ výrazu, pak třída vyhodí výjimku. Úspěšně sestavené DQL se přidá se do dotazu pomocí metody `add` nabízené třídou QueryBuilder.

Poté se přidává primární třída jako zdroj a následně se přidávají spojené třídy, podmínky a parametry. Na závěr se nastaví řazení výsledků. Dotaz se následně vykonává pomocí metod `getQuery()->getResult()` nad sestaveným objektem třídy QueryBuilder.

4.2.2 Sestavení souboru

Tato podkapitola se zabývá implementací metod pro sestavení souboru s exportovanými daty. Práce se zaměřuje na tři základní formáty: CSV, XML a JSON. Pro každý z těchto formátů byla implementována metoda pro sestavení souboru v daném formátu z pole dat získaných výše zmíněnou třídou GenericExportTool.

Pro sestavení souboru s exportovanými daty je použita metoda `generateExport`, která má dva parametry: `exportId` a `filters`. Metoda získá konfiguraci exportu podle `exportId` a poté provede export dat s použitím zadaných filtrů.

Pokud je zadaný formát souboru neplatný, vyvolá se výjimka. Pokud je formát platný, pak je vytvořen soubor s názvem, který obsahuje název konfigurace exportu a datum vytvoření souboru.

Metoda `generateExport` využívá tři další metody: `generateCSVFile`, `generateXMLFile` a `generateJSONFile`. Tyto metody se starají o vytvoření souboru ve zvoleném formátu.

Metoda `generateCSVFile` vytváří soubor ve formátu CSV. Nejprve získá názvy sloupců z prvního řádku dat. Poté inicializuje CSV soubor a přidá názvy sloupců jako první řádek. Nakonec projde všechna data a přidá je do souboru jako jednotlivé řádky.

Metoda `generateXMLFile` vytváří soubor ve formátu XML. Nejprve vytvoří kořenový element a poté rekurzivně projde všechny data a přidá je do XML souboru jako jednotlivé elementy.

Metoda `generateJSONFile` vytváří soubor ve formátu JSON. Nejprve serializuje data do JSON formátu a poté je zapíše do souboru.

Celkově lze říci, že implementace těchto metod umožňuje snadné a efektivní sestavení souboru s exportovanými daty do různých formátů, přičemž lze snadno přidávat další formáty dle potřeby.

4.2.3 REST API

Rozhraní pro komunikaci s frontedem e-shopu jsem implementoval pomocí *controlleru*. Vytvořený ExportToolController je třída implementující REST API rozhraní pro správu a generování datových exportů. Tato třída je součástí implementace celého systému pro správu a zpracování datových exportů. ExportToolController obsahuje několik *endpointů* pro získání, vytvoření, aktualizaci a smazání konfigurace exportu, ale také pro generování samotného exportu.

Hlavní funkcí ExportToolControlleru je umožnit uživatelům systému vytvářet konfigurace exportů, které definují požadovaný formát exportovaných dat, zdrojové tabulky, sloupce, filtry a další parametry. Samotné exportování dat, ale také správa uložených konfigurací, jsou následně prováděny pomocí služby ExportToolService.

ExportToolController obsahuje *endpointy* pro získání všech konfigurací exportu, získání konfigurace exportu podle ID, vytvoření nové konfigurace exportu, aktualizaci existující konfigurace exportu a smazání konfigurace exportu. Každá z těchto akcí odpovídá na dotazy HTTP GET, POST, PUT a DELETE. Zmíněné *endpointy* vrací HTTP kód odpovídající úspěšné nebo neúspěšné operace, v případě úspěchu také odpověď ve formátu JSON.

Poslední akcí v ExportToolControlleru je generování datových exportů. Tato akce je opět nabízena přes *endpoint*, který umožňuje uživatelům systému generovat datové exporty podle specifikované konfigurace. Uživatel může předat filtry, které jsou definované vybranou konfigurací, ty jsou následně použity pro filtrování exportovaných dat. V případě úspěchu je výsledkem této akce StreamedResponse se souborem, který obsahuje exportovaná data.

Použití StreamedResponse umožňuje zpracovávat data postupně v částech, což šetří paměť a umožňuje přenos velkých souborů bez problémů s výkonem nebo dostupností serveru. Třída StreamedResponse představuje odpověď na HTTP požadavek, kde tělo odpovědi je přijímáno asynchronně po obdržení hlaviček. [48]

Asynchroní odesílání nám umožňuje efektivně přenášet velká množství dat přes API. Pokud bychom použili běžný Response, tak by celý soubor musel být nahrán do paměti před odesláním, což by mohlo způsobit problémy s nedostatkem paměti u serveru, zejména pokud by soubor byl velmi velký. Naopak použití StreamedResponse umožňuje zpracovávat data postupně po částech,

což šetří paměť a umožňuje přenos velkých souborů bez problémů s výkonem nebo dostupností serveru. Tento způsob je tedy vhodný pro situace, kdy chceme umožnit uživatelům stahovat velké soubory přes API.

4.3 Problémy s implementací

V souvislosti s implementací nástroje pro export dat jsem se setkal s problémy, které analýza neodhalila. Většina těchto problémů souvisela s omezeními Doctrine ORM a třídy pro vytváření dotazů QueryBuilder. V následujících podkapitolách projdu jednotlivé problémy a nastíním způsoby, kterými jsem je vyřešil.

4.3.1 Omezená množina SQL funkcí

Jedním z problémů byla omezená množina funkcí SQL oficiálně podporovaných Doctrine ORM. Některé z SQL funkcí používaných v současném řešení pro export dat v Doctrine ORM chybí. Jelikož má nový nástroj jako požadavek být schopný vytvářet stejné exporty jako současné řešení, bylo potřeba tento problém vyřešit.

Jako možné řešení se ukázala možnost povolit tyto funkce pomocí implementace vlastního parseru a metody pro vygenerování spouštěného SQL.

Parser zkontroluje, že na vstupu jsou skutečně korektní data — jedna z výhod použití Doctrine ORM je bezpečnost, v tomto případě konkrétně ochrana proti útokům typu SQL injection. Přidáním nové funkce tedy riskujeme snížení bezpečnosti celé aplikace, pokud parser nebude schopen odhalit nekorektní vstupy. Je tedy potřeba nový parser řádně otestovat na nekorektní vstupy, které se snaží docílit jiného cíle než na který je funkce zamýšlená.

V této části práce byla nutná implementace nových SQL funkcí, které nejsou oficiálně podporované Doctrine ORM, aby nový nástroj mohl produkovat stejné exporty jako stávající řešení. Aby se zabezpečilo, že nové funkce nezpůsobí bezpečnostní rizika, bylo potřeba implementovat vlastní parser, který ověřuje korektnost vstupních dat a ochrání aplikaci proti SQL injection útokům. Proto bylo nutné nové funkce pečlivě otestovat na nekorektní vstupy, které by mohly způsobit problémy. Pro implementaci těchto funkcí bylo nutné naimplementovat nové třídy, které dědí od třídy FunctionNode, a obsahují metody `getSql` a `parse`. Konkrétně se jedná o funkce `ROUND`, `REPLACE`, `LEFT`, `CAST`, `DATE_FORMAT` a `DATE_ADD`.

V ukázkách 9 a 10 uvádím implementaci metod `getSql` a `parse` pro SQL funkci `LEFT`. Pro sestavení SQL jsou využity třídy Parser, Lexer a SqlWalker poskytované Doctrine ORM.

```
public function parse(Parser $parser)
    {
        $parser->match(Lexer::T_IDENTIFIER);
        $parser->match(Lexer::T_OPEN_PARENTHESIS);
        $this->left = $parser->ArithmeticPrimary();
        $parser->match(Lexer::T_COMMA);
        $this->right = $parser->ArithmeticPrimary();
        $parser->match(Lexer::T_CLOSE_PARENTHESIS);
    }
```

Výpis kódu 9 Ukázka metody parse pro funkci LEFT

```
public function getSql(SqlWalker $sqlWalker): string
    {
        return "LEFT({$this->left->dispatch($sqlWalker)},
                     {$this->right->dispatch($sqlWalker)})";
    }
```

Výpis kódu 10 Ukázka metody getSql pro funkci LEFT

Třídy, které reprezentují jednotlivé přidané SQL funkce, bylo nutné registrovat v konfiguračním souboru *doctrine.yaml* [49]. V ukázce 11 lze vidět definici nových DQL funkcí s mapováním na třídy s SQL parsery. Jak lze také vidět v ukázce, pro SQL funkci `LEFT` byl zvolen DQL alias `LEFT_CHARS`. Název `LEFT` nebylo možné použít, jelikož je rezervovaný v základu Doctrine ORM pro operaci `LEFT JOIN`.

```
doctrine:
    dbal:
        ...
    orm:
        ...
        dql:
            string_functions:
                LEFT_CHARS: App\Doctrine\DQL\Left
                REPLACE: App\Doctrine\DQL\Replace
                CAST: App\Doctrine\DQL\Cast
            numeric_functions:
                ROUND: App\Doctrine\DQL\Round
            datetime_functions:
                DATE_FORMAT: App\Doctrine\DQL\DateFormat
                DATE_ADD: App\Doctrine\DQL\DateAdd
```

Výpis kódu 11 Ukázka souboru doctrine.yaml

Výsledkem této implementace je, že nový nástroj nyní produkuje stejné exporty jako současné řešení, ale zároveň je zajištěna bezpečnost aplikace.

4.3.2 Datové typy filtrů

Dalším problém, na který jsem v rámci implementace narazil, se týkal nastavitelných parametrů a filtrů. Pro získání dat vytvářeného exportu bylo nutné implementovat metodu, která umožní přidávat filtry do query builderu. Tato metoda však narazila na problém s datovými typy, kvůli kterým jsem musel vytvořit novou metodu.

Při přidávání filtrů bylo nutné zajistit, aby byly data vložená do filtrů korektního datového typu. Pokud bychom například přišli s parametrem ve formě řetězce, který se má porovnávat s časovým údajem, tak by mohlo dojít k nekonzistentním výsledkům. Proto bylo nutné zajistit správnou konverzi datových typů.

Metoda `addParameters` tedy obsahuje logiku, která zajistí, že data jsou konvertována na korektní datový typ v závislosti na jejich hodnotě. Například, pokud je hodnota řetězcem, metoda zkontroluje, zda je to datum a pokud ano, převede jej na objekt `DateTime`, který je následně použit pro vytvoření filtru. Pokud se nepodaří získat `DateTime` objekt, použije se řetězec.

Problém s datovými typy však nebyl jediným problémem, který jsem při implementaci musel řešit. Dalším problémem byla nutnost zajistit správné pojmenování a formátování filtrů. Musel jsem také zajistit kontrolu množství filtrů a způsob, jakým byly vkládány do dotazu.

Celkově byla implementace této metody nutná kvůli problémům s datovými typy a nutnosti zajistit správné pojmenování a formátování filtrů, což je důležité pro zajištění správného fungování celého nástroje pro export dat.

4.3.3 Duplicitní aliasy

Dalším problémem souvisejícím s třídou QueryBuilder byla nemožnost mít vícekrát použitý jeden alias tabulky. Aliasy jsou generovány pomocí metadat jednotlivých tříd, pokud se některá z tříd objevila v rámci celého dotazu vícekrát, třída QueryBuilderu před vykonáním dotazu vyhodila výjimku. Vzhledem k tomu, že QueryBuilder sám výstaví DQL dotaz, musel jsem tuto omezení obejít v rámci samotného DQL. Podařilo se mi to tak, že jsem přidával číselný index za alias při každém výskytu, když se alias v DQL nacházel vícekrát.

Mé řešení pro každý zanořený SELECT prochází všechny aliasy, které jsou v rámci daného SELECTu vytvořeného QueryBuilderem použity a nahrazuje je novými s číselným indexem. Řešení projde celé DQL a pokud najde alias, který se tam nachází vícekrát, přidá za něj číselný index, takto nahradí tak všechny výskyty aliasu v DQL.

Tento postup umožnil použít stejné tabulky v rámci jednoho QueryBuilderu, což bylo nezbytné pro správné sestavení složitějších dotazů.

Kapitola 5
Testování

V této kapitole se zaměřím na popis testovacích metod a nástrojů, které jsem v rámci práce použil. Testování je klíčovou součástí vývojového cyklu a bez něj by bylo obtížné zajistit kvalitní produkt. Díky testování lze také minimalizovat náklady, jelikož odhalení a oprava chyb včas bývají mnohem levnější než následné řešení problémů v produkčním prostředí.

Veškerý kód, který vznikl v této práci, byl testován pomocí statické analýzy, které se věnuji v sekci 5.1. Funkčnost řešení automatizace pravidelých byla dále testována pomocí unit testů, těm se vějuji v sekci 5.2. Nástroj pro generování exportů byl testován za pomocí akceptačních testů, kterým je věnována sekce 5.3.

Testování pomohlo objevit mnohé chyby, které díky tomu byly následně opraveny.

5.1 Statická analýza

Statická analýza softwaru je technika, která se používá k vylepšení kvality kódu a identifikaci potenciálních chyb v softwarovém systému. Statické analýzy se provádějí bez spuštění softwaru a bez zohledňování konkrétního vstupu. Nástroje pro statickou analýzu nezkoumají, zda kód splňuje specifikaci, ale hledají místa, kde by mohly být porušeny rozumné nebo doporučené programovací praktiky. [50]

Statická analýza může také identifikovat problémy s programovým stylem, jako jsou například názvy proměnných nebo použití složených závorek v podmínkách a cyklech. Existuje mnoho nástrojů pro statickou analýzu, ať už open source nebo komerčních, které se liší v použitých technikách analýzy a pokrytí typů chyb. [50]

Jeden z nástrojů, který jsem použil je PHPStan. Jedná se o populární statický analyzátor kódu pro programovací jazyk PHP. PHPStan se zaměřuje na detekci typových chyb jako jsou například nekompatibilní typy argumentů funkcí, chybějící metody nebo vlastnosti v objektech, nesprávné použití proměnných a mnohem více. [51]

V rámci projektu jsem využil PHPStan integrované jako součást GitLab CI, takže při každém odeslání změn pomocí akce `git push` do repozitáře se automaticky spustí PHPStan a zkontroluje se kód. Pokud nástroj najde nějakou chybu, pipeline selže a vývojáři jsou upozorněni na problém.

Dále jsem používal statickou analýzu zabudovanou přímo v použitém vývojovém prostředí PHPStorm. Tímto způsobem jsem byl schopen kontrolovat kód již během vývoje a opravovat chyby a problémy v průběhu implementace.

Díky použití PHPStanu a inspekcím kódu ve vývojovém prostředí jsem značně snížil počet chyb a problémů v kódu a zlepšil kvalitu výsledného produktu.

5.2 Unit testy

Unit testy jsou testovací proces, který se zaměřuje na ověření funkčnosti jednotlivých modulů programu, tedy na ověření, zda jednotlivé části programu pracují správně nezávisle na ostatních částech. Unit testy jsou zaměřeny na ověření funkčnosti kódu na nejnižší možné úrovni a jsou prováděny v izolaci od zbytku systému. Cílem je odhalit chyby v rané fázi vývoje a zajistit tak kvalitu kódu a stabilitu aplikace. [52, 53]

Unit testy jsem použil, protože jsem chtěl zajistit, že implementované funkcionality jsou správně navrženy a fungují přesně tak, jak mají. To může zlepšit kvalitu kódu a minimalizovat počet chyb v produkčním prostředí.

Pro psaní unit testů jsem použil PHPUnit, jedno z nejpoužívanějších a nejrozšířenějších testovacích frameworků pro PHP. PHPUnit nabízí mnoho funkcí a výhod, jako jsou snadná integrace s ostatními nástroji a frameworky, podpora mockování objektů, dělení testovacích dat a mnoho dalšího. [54]

Testoval jsem funkcionalitu implementovaných automatizovaných činností (implementovaných jako *Symfony Command*). Většinu příkazů jsem testoval celé pomocí třídy *CommandTester* poskytované *Symfony* frameworkem. U ostatních příkazů jsem v rámci testů ověřoval přímo funčnost použitých metod. Tyto testy objevily některé chyby, které způsobovaly nefunkčnost kódu.

Díky unit testům jsem byl schopen opravit chyby a zajistit správnou funkčnost kódu. Toto mi pomohlo minimalizovat riziko, že nějaké chyby v kódu proniknou do produkčního prostředí a způsobí problémy pro uživatele.

5.3 Akceptační testy

Akceptační testy mají za cíl ověřit, zda systém dělá správné věci a zda má požadované vlastnosti. Tyto testy mají poskytnout zákazníkovi důvěru, že aplikace funguje správně a má požadované funkce. Akceptační testy jsou někdy nazývány funkčními testy a provádějí se na kompletním systému. Příkladem akceptačního testu může být ověření, zda smazání položky v uživatelském rozhraní ovlivní správně seznam položek v otevřené objednávce. [55]

Cílem této kapitoly je popsat průběh akceptačních testů provedených na nástroji pro generické exporty. Testování bylo zaměřeno na ověření schopnosti nástroje generovat stejné soubory jako současné řešení. Testování proběhlo s využitím dat ze vzdálené testovací databáze. Použitá testovací data obsahovala celkově 510 objednávek a 2000 produktů.

Byla provedena čtyři testování, která se lišila obdobím objednávek a jejich typem. První test zahrnoval platné objednávky z období 1. 1. 2019 až 1. 1. 2020, druhý test platné objednávky z období 1. 1. 2020 až 1. 1. 2021, třetí test opravné objednávky z období 1. 1. 2019 až 1. 1. 2020 a poslední test byl proveden na prázdných datech.

Pro testování byly použity předem vytvořené konfigurace, které měly za úkol vytvořit stejný export jako současné řešení. Testovány byly všechny tři formáty – JSON, XML a CSV. U formátu CSV byly testovány různé oddělovače, včetně CSV s hlavičkou a bez. Dále bylo testováno XML mapování.

Během testování byly zjištěny některé problémy, které bylo potřeba opravit. První řada testů neprošla kvůli problémům se znakem apostrof, což bylo nakonec úspěšně opraveno. Druhá řada testů zase neprošla některými formáty kvůli chybám v konfiguraci, které byly po následných opravách odstraněny. Po těchto úpravách se ale objevila chyba v XML mapování, která byla rovněž úspěšně opravena.

Při testování nad prázdými daty nastaly chyby při generování souborů. Proto byly zavedeny kontroly při generování souborů na takové případy a úspěšně se nyní vytváří prázdné soubory.

Výsledkem akceptačních testů je, že nástroj je schopný na základě testovacích dat generovat stejné exporty jako současné řešení.

Kapitola 6

Možná rozšíření

Při tvorbě konfigurací k akceptačním testům nástroje pro export dat se rychle ukázalo, že existuje několik důvodů, které způsobují nepřehlednost konfigurace, ale také obtížnou tvorbu konfigurací. Konfigurační soubor pro datový export může být matoucí při velkém počtu sloupců a různých operací. Navíc, vypisování všech sloupců, operací a použitých tabulek v rámci formátu JSON je časově náročné a jednoduše se v konfiguraci mohou objevit chyby.

Pro usnadnění tvorby konfigurací a zlepšení použitelnosti navrhuji, aby bylo v budoucnu vytvořeno uživatelské rozhraní na frontendu, které umožní uživatelům vytvářet konfigurace v přehledném a moderním prostředí.

Uživatelské rozhraní by mělo uživatelům umožnit jednoduše a přehledně vytvořit celou konfiguraci. Vniklé řešení by mělo být dynamické a schopné zjišťovat si dostupné výrazy a potřebné atributy využitím API, aby nástroj zůstal dobře udržitelný a rozšířitelný. Uživatelé administrace tak získají možnost efektivně vytvářet i komplexní exporty bez nutné znalosti dokumentace nástroje a bez potřeby vytvářet konfiguraci ve formátu JSON.

Navrhuji také rozšířit API, které vzniklo v rámci práce, aby bylo možné zmíněné uživatelské rozhraní vytvořit. Zde uvádím svůj návrh, jak by vniklé API mohlo vypadat:

`GET /exports/settings/allowed-types` vrátí povolené typy sloupců v konfiguraci a jejich požadované atributy.

```
{
    "allowedTypes" : [
        {
            "name" : "normal",
            "requiredAtributes" : [
                {
                    "name" : "class"
                },
                {
                    "name" : "columnName"
                },
                ...
            ]
        },
        ...
    ]
}
```

`GET /exports/settings/allowed-expressions` vrátí povolené výrazy v konfiguraci a jejich požadované atributy.

```
{
    "allowedExpressions" : [
        {
            "name" : "arithmetic",
            "requiredAtributes" : [
                {
                    "name" : "leftExpression"
                },
                {
                    "name" : "operator"
                },
                {
                    "name" : "rightExpression"
                }
            ]
        },
        ...
    ]
}
```

`GET /exports/settings/allowed-filetypes` vrátí povolené formáty souborů a jejich možná nastavení.

```
{
    "allowedFiletypes" : [
        {
            "name" : "csv",
            "settings" : [
                {
                    "name" : "showHeader"
                },
                {
                    "name" : "delimiter"
                }
            ]
        },
        {
            "name" : "json",
            "settings" : []
        }
        ...
    ]
}
```

`GET /exports/settings/allowed-classes` vrátí povolené třídy pro exportování dat a jejich atributy.

```
{
    "allowedClasses" : [
        {
            "name" : "Order",
            "attributes" : [
                {
                    "name" : "orderId",
                    "type" : "int"
                },
                {
                    "name" : "total",
                    "type" : "float"
                },
                ...
            ]
        },
        ...
    ]
}
```

Dále navrhuji rozšířit nástroj o nepovinné parametry, ale také o požadované datové typy parametrů. Přidání takových možností zamezí chybám se špatně předaným formátem, ale také usnadní uživatelům práci s nástrojem, což povede ke zlepšení uživatelského zážitku a efektivity při tvorbě konfigurací.

```
{
    ...
    "filters" : [
        {
            "filterParameter" : "dateTo"
            "type" : "DateTime",
            "required" : true
        },
        {
            "filterParameter" : "dateFrom",
            "type" : "DateTime",
            "required" : true
        }
    ],
    ...
}
```

V neposlední řadě by nástroj mohl být rozšířen o možnost, která by uživatelům umožnila omezit počet řádků, které se budou exportovat. Tímto způsobem by mohli uživatelé omezit množství dat a usnadnit si práci s výsledným souborem.

Kapitola 7
Závěr

V rámci své práce jsem se věnoval důkladné analýze stávajícího řešení automatizace činností a exportu dat. Dále jsem vedl konzultace s vedoucím práce ohledně problémů současné implementace a o vizi nového řešení.

Na základě této analýzy byly navrženy a implementovány dvě nové komponenty, které umožní automatizaci a správu e-shopu s minimálním zásahem uživatelů. Kladl jsem důraz na udržitelnost, konfigurovatelnost a rozšířitelnost těchto komponent.

První komponenta se zaměřuje na automatizaci pravidelných činností, jako je aktualizace kurzů měn, kontrola funkčnosti odkazů na YouTube, sledování produktů, rozesílání žádostí o recenze, překlad produktů a přepočet cache cen. Dále automatizuje synchronizační činnosti se skladovým systémem e-shopu. Tato komponenta značně usnadňuje práci provozovatelů e-shopu.

Druhá komponenta umožňuje generické exportování dat z e-shopu do různých formátů a je užitečná pro všechny, kteří potřebují pravidelně exportovat data z e-shopu a zpracovávat je v jiných aplikacích. Nové řešení umožňuje spravovat existující exporty, ale také vytvářet nové bez nutnosti znalosti SQL.

Pro zjednodušení tvorby konfigurací do vzniklé komponenty pro export dat bude zapotřebí dále rozšířit nabízené REST API a vybudovat nad ním uživatelské rozhraní.

Při tvorbě nových komponent jsem využil moderní technologie, které umožnily vytvořit robustní, bezpečné a snadno rozšířitelné řešení. Výsledkem mé práce jsou nové komponenty, které usnadní a zefektivní práci provozovatelů e-shopu i uživatelů administrace.

Rovněž jsem získal zkušenost s rolí zadavatele v rámci spolupráce se studentským týmem v předmětu *Softwarový týmový projekt*, kteří vyvíjeli související komponentu pro odesílání emailů.

Příloha A

Příloha

A.1 Výpis sloupců exportu pro Tichý Účto

- `0`
  - řetězec `"0"`
- `x1`
  - prázdný řetězec `""`
- `x2`
  - prázdný řetězec `""`
- `x3`
  - prázdný řetězec `""`
- `s1`
  - datum fakturace
  - `invoice_date` z `oc_order` ve formátu „ddmmyy“
  - například:
    - `"14.11.19"`
- `s2`
  - datum splatnosti
  - `invoice_date` z `oc_order` + interval 13 dní ve formátu „ddmmyy“
  - například:
    - `"27.11.19"`
- `s3`
  - číslo faktury
  - `'f/'` + `invoice_id` z `oc_order`
  - například:
    - `"f/11111111"`

- `s4`
  - `order_id` + `':'` + první znak měny + `':'` + `payment_lastname` + `' '` + `payment_firstname`
  - v `payment_firstname` a `payment_lastname` se nahrazují `':'` za `'_'`
- `PZ`
  - řetězec „PZ“
- `firma_nazev`
  - název firmy
  - pokud je `taxRegNum` prázdné, pak prázdný řetězec `""`
  - pokud není prázdné, pak řetězec „00000“
- `firma_ic_prirazene_z_adresare`
  - prázdný řetězec `""`
- `s6`
  - pokud `shipping_method` není „Osobní“ nebo `payment_lastname` je prázdný řetězec, pak písmeno `B`, jinak písmeno `H`
  - obě písmena `B` i `H` jsou obaleny v jednoduchých závorkách
- `total`
  - celková cena objednávky zaokrouhlená na jedno desetinné číslo
- `datum_dph`
  - `invoice_date` ve formátu „dd.mm.yy”“
- `datum_kh`
  - `invoice_date` ve formátu „dd.mm.yy“
- `dph_21_zaklad`
  - suma cen produktů v objednávce s daňovou sazbou 21 % s přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=21`
  - pokud žádné produkty v objednávce nemájí daňovou sazbu 21 %, pak je hodnota `null`
- `dph_21_dan`
  - suma daně u produktů s daňovou sazbou 21 %
  - suma cen produktů v objednávce s daňovou sazbou 21 % vynásobená hodnotou `tax/100` přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=21`
  - pokud žádné produkty v objednávce nemájí daňovou sazbu 21 %, pak je hodnota `null`
- `dph_15_zaklad`
  - suma cen produktů v objednávce s daňovou sazbou 15 % s přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=15`

- pokud žádné produkty v objednávce nemají daňovou sazbu 15 % , pak je hodnota `null`

- `dph_15_dan`
  - suma daně u produktů s daňovou sazbou 15 %
  - suma cen produktů v objednávce s daňovou sazbou 15 % vynásobená hodnotou `tax/100` přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=15`
  - pokud žádné produkty v objednávce nemají daňovou sazbu 15 %, pak je hodnota `null`

- `dph_10_zaklad`
  - suma cen produktů v objednávce s daňovou sazbou 10 % s přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=10`
  - pokud žádné produkty v objednávce nemají daňovou sazbu 10 %, pak je hodnota `null`

- `dph_10_dan`
  - suma daně u produktů s daňovou sazbou 10 %
  - suma cen produktů v objednávce s daňovou sazbou 10 % vynásobená hodnotou `tax/100` přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=10`
  - pokud žádné produkty v objednávce nemají daňovou sazbu 10 %, pak je hodnota `null`

- `dph_0_zaklad`
  - suma cen produktů v objednávce bez daňové sazby s přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=0`
  - pokud nejsou žádné produkty v objednávce bez daňové sazby, pak je hodnota `null`

- `platba_zaloha`
  - prázdný řetězec `""`

- `x4`
  - prázdný řetězec `""`

- `x5`
  - prázdný řetězec `""`

- `poznamka`
  - řetězec s poznámkou k objednávce, obalený v jednoduchých závorkách
  - poznámky jsou automaticky generovaná
  - skládá se z:
    * „Dopravce: “ + `shipping_method`
    * „Platba: “ + `payment_method`
    * „Heureka kosik: “ + „ano“, pokud je `referrer` heuréka košík, jinak „ne“
    * „Datum zaplaceni: “ + `payment_modifiedon` ve formátu „dd.mm.yy“
    * „Telefon: “ + `telephone`

- `oznacene_vety`
  - písmeno `N`
- `mena_2`
  - Pokud `currency` není „CZK“, pak se vyplní `currency`, jinak prázdný řetězec
- `kc_2`
  - Pokud není `currency` „CZK“, pak se vypočítá cena v českých korunách vynásobením `total` a `value` s přesností na dvě desetinná čísla.
- `nazev0`
  - Pokud je vyplněné `taxRegNum`, jedná se o firmu a vyplníme, jinak vyplníme řetězcem, který obsahuje prázdné jednoduché závorky
    - Hodnotou je `payment_company`, pokud tato hodnota není vyplněná, jako název použijeme `payment_firstname` + `' '` + `payment_lastname`.
    - Řetězec je obalený v jednoduchých závorkách
- `dic0`
  - `taxRegNum` jako řetězec zabalený v jednoduchých závorkách

A.2 Výpis sloupců exportu faktur pro ČSÚ

- `ico`
  - IČO firmy
- `variabilni symbol`
  - číslo objednávky `order_id`
- `datum vystaveni`
  - `invoice_date` ve formátu „dd.mm.yy“
- `datum plneni`
  - `invoice_date` ve formátu „dd.mm.yy“
- `rada`
  - pokud je `currency` „CZK“, pak řetězec „FVE“, jinak řetězec „EEFV“
- `mezera`
  - prázdný řetězec `""`
- `text`
  - spojený řetězec `invoice_id`, `currency`, `payment_lastname` a `payment_firstname`
- `stredisko`
  - podle domény číslo 1-6, jinak 7
- `zaklad 0`
  - suma cen produktů v objednávce bez daňové sazby s přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=0`
  - pokud nejsou žádné produkty v objednávce bez daňové sazby, pak je hodnota `null`
- `zaklad 15`
  - suma cen produktů v objednávce s daňovou sazbou 15 % s přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=15`
  - pokud žádné produkty v objednávce nemají daňovou sazbu 15 % , pak je hodnota `null`
- `dan 15`
  - suma daně u produktů s daňovou sazbou 15 %
  - suma cen produktů v objednávce s daňovou sazbou 15 % vynásobená hodnotou `tax/100` přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=15`
  - pokud žádné produkty v objednávce nemají daňovou sazbu 15 %, pak je hodnota `null`
- `zaklad 21`
  - suma cen produktů v objednávce s daňovou sazbou 21 % s přesností na jedno desetinné místo

- objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=21`
- pokud žádné produkty v objednávce nemají daňovou sazbu 21 % , pak je hodnota `null`

- `dan 21`
  - suma daně u produktů s daňovou sazbou 21 %
  - suma cen produktů v objednávce s daňovou sazbou 21 % vynásobená hodnotou `tax/100` přesností na jedno desetinné místo
  - objednané produkty, pro které platí, že sedí `order_id` naší objednávky a `tax=21`
  - pokud žádné produkty v objednávce nemají daňovou sazbu 21 %, pak je hodnota `null`
- `cena`
  - celková cena objednávky zaokrouhlená na celá čísla
- `domain`
  - název webové domény, například: „www.stylka.cz“
- `currency`
  - tříznaková zkratka měny, například „CZK“ pro českou korunu
- `platebni metoda`
  - `payment_method`
  - řetězec se způsobem platby
  - například:
    - „Platba kartou“
    - „Běžný bankovní“
    - „Poštovní spořitelna / Era“
    - „Na dobírku“
    - „Hotově“

A.3 Ukázky souborů a komunikace mezi systémy

```
# use /bin/sh to run commands, no matter what /etc/passwd says
SHELL=/bin/sh
# mail any output to `paul', no matter whose crontab this is
MAILTO=paul
#
CRON\_TZ=Japan
# run five minutes after midnight, every day
5 0 * * *       $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
# run at 2:15pm on the first of every month -- output mailed to paul
15 14 1 * *     $HOME/bin/monthly
# run at 10 pm on weekdays, annoy Joe
0 22 * * 1-5    mail -s "It's 10pm" joe%Joe,%%Where are your kids?%
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ..., everyday"
5 4 * * sun     echo "run at 5 after 4 every sunday"+
```

[34]

Výpis kódu 12 Ukázka crontab souboru

```
[
  {
    "product_instance": {
      "id": 1,
      "type": "BATCH",
      "product_id": 3,
      "serial_number": null,
      "batch_code": "12345678914"
    },
    "quantity": 3,
    "blocked": 3
  }
]
```

Výpis kódu 13 Ukázka odpovědi od skladového systému

```
04.11.2022 #214
země|měna|množství|kód|kurz
Austrálie|dolar|1|AUD|15,950
Brazílie|real|1|BRL|4,916
Bulharsko|lev|1|BGN|12,486
Čína|žen-min-pi|1|CNY|3,446
Dánsko|koruna|1|DKK|3,282
EMU|euro|1|EUR|24,420
Filipíny|peso|100|PHP|42,343
Hongkong|dolar|1|HKD|3,151
Chorvatsko|kuna|1|HRK|3,242
Indie|rupie|100|INR|30,139
Indonesie|rupie|1000|IDR|1,572
Island|koruna|100|ISK|16,784
Izrael|nový šekel|1|ILS|6,961
Japonsko|jen|100|JPY|16,821
Jižní Afrika|rand|1|ZAR|1,372
Kanada|dolar|1|CAD|18,292
Korejská republika|won|100|KRW|1,747
Maďarsko|forint|100|HUF|6,088
Malajsie|ringgit|1|MYR|5,210
Mexiko|peso|1|MXN|1,268
MMF|ZPČ|1|XDR|31,495
Norsko|koruna|1|NOK|2,394
Nový Zéland|dolar|1|NZD|14,565
Polsko|zlotý|1|PLN|5,215
Rumunsko|leu|1|RON|4,995
Singapur|dolar|1|SGD|17,579
Švédsko|koruna|1|SEK|2,250
Švýcarsko|frank|1|CHF|24,759
Thajsko|baht|100|THB|66,167
Turecko|lira|1|TRY|1,328
USA|dolar|1|USD|24,737
Velká Británie|libra|1|GBP|27,918
```

Výpis kódu 14 Ukazka souboru s kurzy měn

```
<ZavazkyPohledavky>
        <ZavazekPohledavka>
                <DATUM_VYSTAVENI Value="{invoice_date}" />
                <DATUM_SPLATNOSTI Value="{payment_date}" />
                <PZ Value="P" />
                <UCTOVAT_PREDPIS Value="1" />
                <DOKLAD_CISLO Value="{order_id}" />
                <DAL_SYMBOL Value="{order_id}" />
                <POPIS Value="Maloobchodní prodej" />
                <FIRMA>
                        <NAZEV Value="{company?: payment_firstname.payment_lastname}"/>
                        <ADRESA1 Value="{payment_address}"/>
                        <CISLO_POPISNE_1 Value=""/>
                        <CISLO_ORIENTACNI_1 Value=""/>
                        <MISTO Value="{payment_city}"/>
                        <PSC Value="{payment_postcode}"/>
                        <ICO Value="{regNum}"/>
                        <DIC Value="{taxRegNum}"/>
                </FIRMA>
                <MENA>
                        <MENA_ZKRATKA Value="{currency}" />
                        <MENA_KURZ Value="{value}" />
                </MENA>
                <DPH_DATUM Value="{invoice_date}"/>
                <DPH_UZP Value="{invoice_date}"/>
                <DPH_ZAKLAD_Z Value="{dph_21_zaklad}"/>
                <DPH_Z Value="{dph_21_dan}"/>
                <DPH_ZAKLAD_S Value="{dph_15_zaklad}"/>
                <DPH_S Value="{dph_15_dan}"/>
                <DPH_ZAKLAD_D Value="{dph_10_zaklad}"/>
                <DPH_D Value="{dph_10_dan}"/>
                <DPH_ZAKLAD_N Value="{dph_0_zaklad}"/>
                <CELKEM Value="{total}"/>
                <BANKOVNI_UCET Value="{bank_account[0]}"/>
                <BANKOVNI_KOD Value="{bank_account[1]}"/>
        </ZavazekPohledavka>
        <ZavazekPohledavka>
        ...
        </ZavazekPohledavka>
        ...
</ZavazkyPohledavky>
```

Výpis kódu 15 Ukázka XML exportu do účetnictví Ježek

```
{
    "version":"1.0",
    "thumbnail_height":360,
    "url":"https://www.youtube.com/watch?v=dQw4w9WgXcQ",
    "html":"
        <iframe
            width=\"200\"
            height=\"113\"
            src=\"https://www.youtube.com/embed/dQw4w9WgXcQ?feature=oembed\"
            frameborder=\"0\"
            allow=\"accelerometer;
                autoplay;
                clipboard-write;
                encrypted-media;
                gyroscope;
                picture-in-picture\"
            allowfullscreen
            title=\"Rick Astley - Never Gonna Give You Up (Official Music Video)\">
        </iframe>",
    "height":113,
    "type":"video",
    "provider_url":"https://www.youtube.com/",
    "title":"Rick Astley - Never Gonna Give You Up (Official Music Video)",
    "author_url":"https://www.youtube.com/c/RickastleyCoUkOfficial",
    "author_name":"Rick Astley",
    "thumbnail_url":"https://i.ytimg.com/vi/dQw4w9WgXcQ/hqdefault.jpg",
    "provider_name":"YouTube",
    "width":200,
    "thumbnail_width":480
}
```


Výpis kódu 16 Ukázka odpovědi noembed – existující video


```
{"url":"https://www.youtube.com/watch?v=lxBx15w-jd0","error":"404 Not Found"}
```


Výpis kódu 17 Ukázka odpovědi noembed – neexistující video

```
{
    "exportName": "Ježek",
    "exportJsonConfig": "{\n
    \"exportFormat\": \"csv\",\n
    \"columns\": [\n
      {\n
        \"type\": \"normal\",\n
        \"name\": \"date\",\n
        \"class\": \"Order\",\n
        \"columnName\": \"invoiceDate\"\n
      },\n
      {\n
        \"type\": \"normal\",\n
        \"name\": \"id\",\n
        \"class\": \"Order\",\n
        \"columnName\": \"invoiceId\"\n
      }\n
    ],\n
    \"primaryClass\": \"Order\",\n
    \"conditions\": [\n
      {\n
        \"column\": \"invoiceDate\",\n
        \"class\": \"Order\",\n
        \"conditionType\": \"higherOrEqual\",\n
        \"conditionExpression\": \":dateFrom\"\n
      }\n
    ],\n
    \"joinedClasses\": [],\n
    \"parameters\": [\n
      {\n
        \"filterParameter\": \"dateFrom\"\n
      }\n
    ],\n
    \"orderBy\": [\n
      {\n
        \"class\": \"Order\",\n
        \"column\": \"invoiceId\",\n
        \"order\": \"asc\"\n
      }\n
    ]\n
  }"
}
```

Výpis kódu 18 Ukázka těla HTTP žádosti

Bibliografie

1. OPENCART. *OpenCart - Open Source Shopping Cart Solution* [online]. [cit. 2023-04-23]. Dostupné z: `https://www.opencart.com`.
2. JAGU S.R.O. *Jagu - Software na míru* [online]. [cit. 2023-02-12]. Dostupné z: `https://www.jagu.cz`.
3. KOUBA, Alois. *Backend administrace e-shopu*. 2022. Dostupné také z: `https://dspace.cvut.cz/handle/10467/102088`. Bakalářská práce. České vysoké učení technické v Praze.
4. HOJEK, Tomáš. *Backend a CI administrace e-shopu*. 2023. Dostupné také z: `https://dspace.cvut.cz/handle/10467/107230`. Diplomová práce. České vysoké učení technické v Praze.
5. KRASNER, Glenn E; POPE, Stephen T et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*. 1988, roč. 1, č. 3.
6. RED HAT. *What is a REST API?* [online]. 2021. [cit. 2023-05-08]. Dostupné z: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.
7. THE PHP GROUP. *PHP Manual* [online]. [cit. 2023-04-23]. Dostupné z: `https://www.php.net/manual/en/intro-whatis.php`.
8. LERDORF, Rasmus; TATROE, Kevin; KAEHMS, Bob; MCGREDY, Ric. *Programming Php*. "O'Reilly Media, Inc.", 2002.
9. SYMFONY. *What is Symfony?* [Online]. Dostupné také z: `https://symfony.com/what-is-symfony`.
10. ZANINOTTO, François; POTENCIER, Fabien. *The definitive guide to Symfony*. Apress, 2007.
11. ORACLE CORPORATION. *MySQL Reference Manual* [online]. [cit. 2023-04-23]. Dostupné z: `https://dev.mysql.com/doc/refman/8.0/en/introduction.html`.
12. DOCTRINE PROJECT. *Getting Started with Doctrine* [online]. [cit. 2023-04-23]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-orm/en/2.9/tutorials/getting-started.html#what-is-doctrine`.
13. NGINX. *What is NGINX?* [online]. 2023. [cit. 2023-04-24]. Dostupné z: `https://www.nginx.com/resources/glossary/nginx/`.
14. DOCKER INC. *Docker: Accelerated, Containerized Application Development* [online]. [cit. 2023-04-23]. Dostupné z: `https://www.docker.com`.

15. KEYCLOAK PROJECT. *Keycloak - Open Source Identity and Access Management* [online]. [cit. 2023-02-12]. Dostupné z: https://www.keycloak.org.
16. SHAKEBUGS. *Collaboration in Software Development* [online]. 2022. [cit. 2023-05-08]. Dostupné z: https://www.shakebugs.com/blog/collaboration-in-software-development.
17. GITENTIAL. *Teamwork in Software Development* [online]. 2022. [cit. 2023-05-08]. Dostupné z: https://gitential.com/teamwork-in-software-development-8-benefits-of-building-up-a-collaborative-team.
18. GROWIN. *The 7 Most Common Pitfalls of Software Development Teams* [online]. 2021. [cit. 2023-05-08]. Dostupné z: https://www.growin.com/blog/pitfalls-of-software-development-teams.
19. TRAN, Trung. *Common Challenges and Strategies for Managing Software Development Teams* [online]. 2022. [cit. 2023-05-08]. Dostupné z: https://www.orientsoftware.com/blog/managing-software-development-teams.
20. MILLARD, Elizabeth. *Voice Communication Creates Stronger Bond Than Text* [online]. 2020. [cit. 2023-05-08]. Dostupné z: https://www.verywellmind.com/voice-communication-creates-stronger-bond-than-text-5082922.
21. SLACK TECHNOLOGIES. *About Us - Slack* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://slack.com/about.
22. GITLAB. *The DevSecOps Platform - GitLab* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://about.gitlab.com.
23. IBM. *What is DevSecOps?* [online]. [cit. 2023-05-08]. Dostupné z: https://www.ibm.com/topics/devsecops.
24. LANG, Jean-Philippe. *Redmine* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.redmine.org.
25. SMARTBEAR SOFTWARE. *About Swagger* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://swagger.io/about.
26. RADVIEW SOFTWARE. *What is Mock Testing?* [online]. 2023. [cit. 2023-05-08]. Dostupné z: https://www.radview.com/glossary/what-is-mock-testing.
27. SIMIC, Peter. *Why Train Developers?* [online]. [cit. 2023-05-08]. Dostupné z: https://www.shakebugs.com/blog/why-train-developers.
28. AYLWARD, Lee. *Noembed* [online]. [cit. 2023-03-08]. Dostupné z: https://noembed.com.

č. 89/1995 Sb. *Zákon o statistice* [online]. Parlament České republiky, 1995. [cit. 2023-04-09]. Dostupné z: https://www.zakonyprolidi.cz/cs/1995-89.

29. *JSON.org* [online]. [cit. 2023-03-04]. Dostupné z: https://www.json.org/json-en.html.
30. KLÍMEK, Jakub. *Otevřená data* [online]. 2021-03-18. [cit. 2023-03-04]. Dostupné z: https://opendata.gov.cz/standardy:csv.
31. W3C. *XML* [online]. [cit. 2023-03-04]. Dostupné z: https://www.w3.org/XML/.
32. NEMETH, Evi; SNYDER, Garth; HEIN, Trent R.; WHALEY, Ben. *Unix and Linux System Administration Handbook*. 4th. Addison-Wesley Professional, 2017.
33. VIXIE, Paul; MAŠLÁŇOVÁ, Marcela; DEAN, Colin; MRÁZ, Tomáš. *Cron* [online]. 2021. [cit. 2023-03-04]. Dostupné z: https://man7.org/linux/man-pages/man8/cron.8.html.

34. VIXIE, Paul. *crontab(5) — Linux manual page* [online]. [cit. 2023-03-04]. Dostupné z: https://man7.org/linux/man-pages/man5/crontab.5.html#top_of_page.
35. LINAS, L. *Hostinger Tutorials - crontab* [online]. [cit. 2023-03-04]. Dostupné z: https://www.hostinger.com/tutorials/wp-content/uploads/sites/2/2021/09/crontab-syntax.webp.
36. LINAS, L. *Hostinger Tutorials - cron job* [online]. [cit. 2023-03-04]. Dostupné z: https://www.hostinger.com/tutorials/cron-job.
37. MRÁZ, Tomáš. *Cronie* [online]. [cit. 2023-04-23]. Dostupné z: https://github.com/cronie-crond/cronie.
38. TECMINT. *Cron vs Anacron: Schedule Jobs Using Anacron on Linux* [online]. 2022. [cit. 2023-05-07]. Dostupné z: https://www.tecmint.com/cron-vs-anacron-schedule-jobs-using-anacron-on-linux.
39. ROOT.CZ. *Anacron* [online]. 2023. [cit. 2023-05-07]. Dostupné z: https://www.root.cz/man/8/anacron/.
40. GODOUET, Thibault. *Fcron* [online]. [cit. 2023-04-23]. Dostupné z: http://fcron.free.fr.
41. FREEDESKTOP.ORG. *systemd.timer* [online]. [cit. 2023-04-23]. Dostupné z: https://www.freedesktop.org/software/systemd/man/systemd.timer.html.
42. DOCKER INC. *Docker Docs* [online]. [cit. 2023-04-23]. Dostupné z: https://docs.docker.com.
43. KULATUNGA, Jason. *Sparktree* [online]. [cit. 2023-02-26]. Dostupné z: https://blog.thesparktree.com/cron-in-docker.
44. BOND, Kevin. *The ScheduleBundle* [online]. [cit. 2023-03-08]. Dostupné z: https://github.com/zenstruck/schedule-bundle.
45. DOCKER HUB. *strm tasker* [online]. 2019. [cit. 2023-05-08]. Dostupné z: https://hub.docker.com/r/strm/tasker.
46. SYMFONY. *Symfony - Running Crons* [online]. [cit. 2023-03-08]. Dostupné z: https://symfony.com/doc/current/the-fast-track/en/24-cron.html.
47. DOCTRINE PROJECT. *Doctrine Query Language* [online]. [cit. 2023-05-10]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.15/reference/dql-doctrine-query-language.html.
48. SYMFONY. *Symfony - Streaming a Response* [online]. [cit. 2023-05-08]. Dostupné z: https://symfony.com/doc/current/components/http_foundation.html#streaming-a-response.
49. SYMFONY. *How to Register custom DQL Functions* [online]. [cit. 2023-05-09]. Dostupné z: https://symfony.com/doc/current/doctrine/custom_dql_functions.html.
50. AYEWAH, Nathaniel; PUGH, William; HOVEMEYER, David; MORGENTHALER, J David; PENIX, John. Using static analysis to find bugs. *IEEE software*. 2008, roč. 25, č. 5, s. 22–29.
51. MIRTES, Ondřej. *PHPStan* [online]. 2016-2023. [cit. 2023-05-05]. Dostupné z: https://phpstan.org.
52. IEEE. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*. 1990, s. 1–84. Dostupné z DOI: 10.1109/IEEESTD.1990.101064.

53. PARKIN, Rodney; AUSTRALIA, I. Software unit testing. *The Independent Software Testing Specialists, IV & V Australia*. 1997.

54. BERGMANN, Sebastian. *PHPUnit – The PHP Testing Framework* [online]. [cit. 2023-05-07]. Dostupné z: `https://phpunit.de`.

55. MILLER, Roy; COLLINS, Christopher T. Acceptance testing. *Proc. XPUniverse*. 2001, roč. 238.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `export` ..... konfigurační soubory pro nástroj pro export dat
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF