Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Knihovna pro posílání e-mailů za využití fronty v jazyce PHP

DIPLOMOVÁ PRÁCE

ve studijním programu Aplikovaná informatika

oboru Podniková informatika

Autor: Bc. Petr Olišar

Vedoucí diplomové práce: Doc. Ing. Vilém Sklenák, CSc.

Praha, duben 2018

Prohlášení

Prohlašuji, že diplomovou práci s názvem Knihovna pro posílání e-mailů za využití fronty v jazyce PHP jsem zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 25. dubna 2018 …….

Bc. Petr Olišar

Poděkování

Děkuji především svému vedoucímu Doc. Ing. Vilému Sklenákovi, CSc. za to, že se ujmul mého tématu, za velice vstřícný přístup, trpělivost a za cenné rady, kterými udával směr mé diplomové práce.

Také děkuji svým kolegům z práce za konzultace týkající se návrhu aplikace, za poskytnutou infrastrukturu a za pomoc se získáváním analytických dat.

V neposlední řadě bych chtěl poděkovat své rodině, která mi umožnila studium a byla mi velkou oporou během studia.

Abstrakt

Tato diplomová práce je zaměřena na představení problematiky rozesílání e-mailů pomocí fronty. Vychází z reálných potřeb společnosti, ve které pracuji, a kde také probíhal sběr analytických dat, na jejichž základě mohly být učiněny závěry diplomové práce. Záměrem práce je porovnat dva přístupy: odesílání e-mailů pomocí databázové fronty a odesílání pomocí speciálního software určeného pro práci s frontou RabbitMQ. Jedním ze dvou hlavních cílů je detailně popsat fungování aplikace RabbitMQ. Druhým hlavním cílem práce je najít optimální řešení pro rychlost rozesílky, a to bez většího vytížení serveru během generování a rozesílání e-mailů. Toto optimální řešení je promítnuto do knihovny, která umožní oddělit aplikaci, jež e-maily odesílá, od generování a grafické podoby e-mailu. Výsledkem práce je knihovna, publikovaná jako open source, která je otestována zátěží rozesílky milionů e-mailů týdně. V práci jsou také uvedeny detailní statistiky, které porovnávají rychlost rozesílání emailů v návaznosti na zatížení serveru (procesor, paměť, databázi, pevné disky) při dané rozesílce.

Klíčová slova

Databáze, e-mail, fronta, RabbitMQ.

Abstract

The master´s thesis is based on introduction of problematics of sending e-mails with a help of a queue. It reflects needs of a company where I work and where the data collection has been done. The intent of my thesis was to compare two approaches: sending emails by database´s front or sending emails with help of special software designated for work with RabbitMQ. The one of the two main aims of the thesis is to describe working with RabbitMQ. To find an optimal solution for swiftness of the e-mail´s delivery is the second purpouse, while the server remains without significant overload. The aim is to create an application which will be capable of separating sending and creating emails. The application is now publicated as an opensource and it has been tested by delivery frequence of millions of e-mails weekly. There are detailed statistics in the master´s thesis. They compare swiftness of the delivery following the load of the server during the whole process of sending e-mails.

Keywords

Database, e-mail, queue, RabbitMQ.

Obsah

1 Úvod ... 1
1.1 Vymezení tématu práce ... 1
1.2 Současný stav implementace, RabbitMQ do jazyka PHP ... 1
1.3 Cíle práce ... 2
1.4 Rešerše ... 2
1.5 Předpoklady a omezení práce ... 3
1.6 Struktura práce ... 3
1.7 Výstupy a přínosy práce ... 3
2 Specifikace ... 4
2.1 Výchozí stav ... 4
2.2 Základní požadavky ... 4
2.3 Základní princip ... 5
2.3.1 Implementace prostřednictvím databázové fronty ... 6
2.4 Požadavky na architekturu ... 7
2.5 Případy užití ... 7
2.6 Diagram tříd ... 9
2.6.1 Definice UML diagramu ... 9
2.6.2 Hlavní entity e-mailu ... 9
2.6.3 Vytvoření entity e-mailu ... 10
2.6.4 Odeslání e-mailu ... 11
2.6.5 Struktura entity e-mailu ... 12
2.6.6 Komplexní pohled na knihovnu ... 13
2.7 Struktura databáze ... 13
3 Možnosti odesílání e-mailů v jazyce PHP ... 17
3.1 Nativní funkce mail ... 17
3.2 SMTP (stream_socket_client) ... 17
3.3 Přímé odesílání ... 18
3.4 Pomocí fronty ukládané do databáze ... 18
3.5 Pomocí RabbitMQ ... 19
4 Odesílání e-mailů pomocí databázové fronty ... 20
4.1 Hlavní logika zpracování databázové fronty ... 20
4.2 Generování newsletterů do fronty ... 21

4.2.1 Kde se vezmou newslettery ... 21
4.2.2 Rychlost generování a odesílání na SMTP server ... 21
4.3 Instalace ... 22
4.3.1 Ansible ... 22
4.3.2 Composer ... 23
4.4 Implementace knihovny ... 23
4.4.1 Vytvoření e-mailu ... 23
4.4.2 Uložení e-mailu do fronty ... 25
4.4.3 Odeslání e-mailu ... 25
4.5 Problémy při implementaci databázové fronty ... 26
4.5.1 Problém s RAM pamětí ... 26
4.5.2 Problém s odpojením připojení do databáze ... 28
5 Technologie RabbitMQ ... 29
5.1 Jak funguje message broker ... 29
5.2 Jak funguje RabbitMQ ... 30
5.2.1 Potvrzování zpráv ... 30
5.2.2 Publisher ... 31
5.2.3 Exchange ... 31
5.2.4 Queue ... 32
5.2.5 Consumer ... 32
5.3 Exchange ... 32
5.3.1 Výchozí (default) Exchange ... 33
5.3.2 Fanout Exchange ... 33
5.3.3 Adresovaný (direct) Exchange ... 34
5.3.4 Tématický (topic) Exchange ... 35
5.3.5 Headers Exchange ... 36
5.3.6 Dead letter Exchange ... 37
5.3.7 Exchanges ... 37
5.4 Queue (Fronta) ... 38
5.4.1 Prefetch ... 38
5.4.2 Negativní potvrzení ... 39
5.5 K čemu se používá ... 39
5.5.1 Zrychlit fungování aplikace ... 39
5.5.2 Docílit zpracování v reálném čase ... 39
5.6 Integrace RabbitMQ a PHP ... 41

5.7 Nevýhody ... 41
5.7.1 Protokol TCP ... 41
5.7.2 Zátěž systému ... 41
5.7.3 Složitost implementace ... 42
6 Odesílání e-mailů pomocí RabbitMQ ... 43
6.1 Vložení dat do fronty ... 43
6.2 Čtení z fronty ... 44
7 Dopad na výkon ... 47
7.1 Rychlost RabbitMQ serveru ... 47
7.1.1 Publikování a konzumování naráz ... 47
7.1.2 Publikování a konzumování zvlášť ... 47
7.1.3 Publikování a konzumování naráz + potvrzení přijetí a zpracování ... 48
7.1.4 Publikování a konzumování naráz + potvrzování + persistence ... 48
7.2 Rychlost rozesílky ... 49
7.2.1 Jeden SMTP server ... 49
7.2.2 Tři SMTP servery ... 49
7.2.3 Nová knihovna pomocí databázové fronty ... 50
7.2.4 Nová knihovna pomocí RabbitMQ ... 51
7.3 Zatížení serveru ... 52
7.3.1 Legenda ke grafům ... 52
7.3.2 Databázová fronta ... 54
7.3.3 RabbitMQ ... 55
8 Závěr ... 58
Použitá literatura ... 60
A Přílohy ... 62
A.1 Slovníček pojmů ... 62
A.1.1 View (pohled) ... 62
A.1.2 SMTP ... 62
A.1.3 Fronta ... 62
A.1.4 Messaging broker ... 62
A.1.5 Databáze ... 62
A.1.6 Adapter ... 62
A.1.7 Rozhraní (interface) ... 63
A.1.8 Entita ... 63
A.1.9 PHP ... 63

A.1.10 Composer ... 63
A.1.11 PSR-4 ... 63
A.1.12 Balíček ... 63
A.1.13 API (Application Programming Interface) ... 64
A.1.14 Use case diagram ... 64
A.1.15 PDO ... 64
A.1.16 Garbage collector ... 64
A.1.17 Message-oriented middleware ... 64
A.1.18 AMQP (Advanced Message Queuing Protocol) ... 64
A.1.19 (TTMP) Streaming Text Oriented Messaging Protocol ... 64
A.1.20 Message Queuing Telemetry Transport ... 64
A.1.21 TTL – time to live ... 65
A.1.22 RPC (Remote procedure call) ... 65
A.1.23 TCP (Transmission Control Protocol) ... 65
A.1.24 ORM (Object-relational mapping) ... 65
A.1.25 ElastSearch ... 65

Seznam obrázků

Obrázek 1 Případ užití komunikace mezi aplikací a mail serverem. ... 6
Obrázek 2 Model případů užití ... 7
Obrázek 3 Vazba mezi typem e-mailu a e-mailem ... 9
Obrázek 4 Vytvoření e-mailu ... 10
Obrázek 5 Odeslání e-mailu ... 11
Obrázek 6 Struktura entity e-mailu ... 12
Obrázek 7 Kompletní diagram vazeb v knihovně mailsender/core ... 13
Obrázek 8 Sekvenční diagram vzoru message brooker ... 29
Obrázek 9 Jak funguje RabbitMQ ... 30
Obrázek 10 Výchozí (default) Exchange ... 33
Obrázek 11 Fanout exchange ... 33
Obrázek 12 Adresovaný (direct) Exchange ... 34
Obrázek 13 Tématický (topic) Exchange ... 35
Obrázek 14 HeadersExchange ... 36
Obrázek 15 Dead letter Exchange ... 37
Obrázek 16 Závislost počet consumerů a prefetch count ... 38
Obrázek 17 Proces komunikace mezi skladovou a e-shopovou aplikací ... 40
Obrázek 18 Rychlost - publikování a konzumování naráz ... 47
Obrázek 19 Rychlost - pouze publikování ... 48
Obrázek 20 Rychlost - pouze konzumování ... 48
Obrázek 21 Rychlost - publikování a konzumování naráz + potvrzování ... 48
Obrázek 22 Rychlost - Publikování a konzumování naráz + potvrzování + persistence ... 49
Obrázek 23 Jak funguje CPU load? ... 53
Obrázek 24Problém s pamětí - RAM ... 54
Obrázek 25Problém s pamětí - odkládací prostor RAM ... 54
Obrázek 26Problém s pamětí - CPU load ... 55
Obrázek 27Problém s pamětí - zápis/čtení disku ... 55
Obrázek 32 Dvanáct consumerů (prefetch 1) - paměť ... 56
Obrázek 33 Dvanáct consumerů (prefetch 1) – CPU load ... 56
Obrázek 34 Dvanáct consumerů (prefetch 1) - zápis/čtení disku ... 56
Obrázek 35 Dvanáct consumerů (prefetch 1) - MySQL ... 57
Obrázek 36 Šest consumerů (prefetch 20) - paměť ... 57
Obrázek 37 Šest consumerů (prefetch 20) ... 57

Seznam tabulek

Tabulka 1 Případ užití 8
Tabulka 2 Typy e-mailů 14
Tabulka 3 Entita s e-mail 15
Tabulka 4 Rychlost rozesílky newslettrů - Jeden SMTP server 49
Tabulka 5 Rychlost rozesílky newsletterů - Tři SMTP servery 50
Tabulka 6 Rychlost rozesílky newsletterů - Nová knihovna pomocí databázové fronty (problém s RAM) 50
Tabulka 7 Rychlost rozesílky newsletterů - Nová knihovna pomocí databázové fronty 51
Tabulka 8 Rychlost rozesílky newsletterů - Nová knihovna pomocí RabbitMQ (12 consumerů, prefetch 1) 51
Tabulka 9 Rychlost rozesílky newsletterů - Nová knihovna pomocí RabbitMQ (6 consumerů, prefetch 1) 52
Tabulka 10 Rychlost rozesílky newsletterů - Nová knihovna pomocí RabbitMQ (6 consumerů, prefetch 20) 52

1 Úvod

1.1 Vymezení tématu práce

Diplomová práce se zaměřuje na problematiku zpracovávání front. Konkrétněji se zabývá zpracováním front v jazyce PHP, jež detailně popisuje. Zpracování front je poměrně široké téma, proto jsem se jej rozhodl zúžit na jeden z typických případů, na který se fronta používá: na rozesílání e-mailů.

E-maily se pomocí fronty řeší typicky ze dvou důvodů.

1) Potřeba zrychlení odpovědi serveru, kterou posílá klientovi. Odeslání e-mailu je časově náročná činnost, která typicky trvá od necelé vteřiny do nízkých jednotek vteřin. Může to však být až řádově několik minut. Podle RFC 5321 by měl být server nastaven tak, aby čekal i desítky minut, dokud nevyprší čas spojení. Důležitou roli zde hraje použitá technologie [1].
2) Nutnost rozesílání sady e-mailů. Jedná se o předem vytvořenou sadu e-mailů a cílem je co nejrychleji ji odeslat na SMTP server a doručit cílovému adresátu. Příkladem takové sady e-mailů může být newsletter. Jedná se o předem vytvořený e-mail, který se rozešle na množství e-mailových adres.

Tato práce se zabývá dvěma základními způsoby řešení front:

- Standardním postupem, při němž se odesílaný e-mail uloží do databáze a poté se cronem zpracovává FIFO frontou.
- Použitím RabbitMQ, což je služba běžící na pozadí a zajišťující odbavování front. Blíže bude tato technologie rozebrána v kapitole 6.

Technologie RabbitMQ má velice široké použití a v této práci se věnuji i jejím dalším využitím. RabbitMQ je messaging broker. Může propojovat jednu část aplikace s jejími jinými částmi, s jinými aplikacemi nebo dokonce s aplikacemi napsanými jiným programovacím jazykem.

Poslední oblastí, kterou se tato práce zabývá, je knihovna php-amqplib a její alternativy/rozšíření, které zprostředkovávají práci s RabbitMQ v jazyce PHP.

1.2 Současný stav implementace, RabbitMQ do jazyka PHP

Klientem RabbitMQ je knihovna php-amqplib, jejímž autorem je Vadim Zaliva. Poslední stabilní verze vyšla 11. února 2018. RabbitMQ svou oficiální knihovnu nevyvíjí a v dokumentaci pracuje právě s knihovnou php-amqlib.

Nad touto knihovnou existuje spousta nadstaveb, které ji integrují do různých fremeworků. Za všechny zde uvedu 2 příklady pro nejčastější frameworky:

vladimir-yuldashev/laravel-queue-rabbitmq pro Laravel a **php-amqplib/RabbitMqBundle** pro Symfony.

Existuje také spousta knihoven, které buď rozšiřují php-amqplib nebo usnadňují práci s ní. Říká se jim Bunny. Například **jakubkulhan/bunny** od českého vývojáře.

1.3 Cíle práce

Má práce vychází z reálných potřeb společnosti, ve které pracuji. E-mailů se v ní posílá velké množství a jejich odesílání má i silný dopad na stabilitu fungování ostatních aplikací.

Prvním cílem práce je **navrhnout, popsat a vytvořit systém odesílání e-mailů**, který se bude moci co nejvíce (ideálně úplně) odstřihnut od hlavní databáze a hlavního serveru. Tento krok by měl mít ten efekt, že odesílání jakéhokoli množství e-mailů nebude mít žádný vliv na výkon hlavních aplikací společnosti.

Podcílem je vytvořit systém tak, že se kompletně oddělí výsledná podoba e-mailu od místa, kde se pro něj generují data.

Druhá část práce se věnuje technologii RabbitMQ, pomocí které se odesílání bude realizovat. Od tohoto kroku si slibuji optimální využití výpočetního výkonu a přesné načasování jiných služeb, které se nyní spouští velice často po malých dávkách, protože musí být odbaveny velice rychle. Spouštějí se tedy mnohdy úplně zbytečně.

Z těchto praktických cílů vychází i cíle teoretické:

1. porovnání zatížení serverů a délky odesílání e-mailů před nasazením první a druhé části aplikace.
2. podrobné popsání technologie RabbitMQ, její přínosy a práci s ní.

Teoretické poznatky jsou v praxi demonstrovány na knihovně odesílající e-maily.

Posledním cílem je popsat jiné možnosti řešení front v jazyce PHP a pro porovnání naprogramovat knihovnu, která odesílá e-maily pomocí fronty řešené pomocí databáze a cronu. Tato knihovna by měla mít stejné API jako knihovna používající RabbitMQ proto, aby byly vzájemně zaměnitelné.

1.4 Rešerše

Knihovna, která by zajišťovala zasílání e-mailů z fronty a využívala RabbitMQ jako open-source, v době psaní mé diplomové práce neexistuje.

Nenašel jsem ani žádnou jinou knihovnu, která by se zabývala vytvářením e-mailů tak, jak jsem je popsal v cílech práce.

1.5 Předpoklady a omezení práce

Diplomová práce předpokládá čtenáře zběhlého v programovacím jazyce PHP. Aplikace ke svému běhu vyžaduje webový server, PHP 7.1, composer, MySQL, RabbitMQ a linux. Potřebné technologie jsou jednoduše instalovatelné v rámci naprogramované aplikace.

1.6 Struktura práce

Diplomová práce se skládá ze slovníčku pojmů, kde se čtenář seznámí se všemi specializovanými výrazy, jejichž pochopení mu umožní další práci s textem. Dále se skládá ze specifikace, jež je členěna na výchozí stav, budoucí stav, popis technologií a use case diagramy. V neposlední řadě je tu rozbor technologií po teoretické stránce a dvě knihovny. Jsou to knihovny sloužící k odesílání e-mailů pomocí fronty, takže navenek se chovají totožně, jen na pozadí jsou použity různé technologie.

Cron i RabbitMQ, dvě hlavní technologie, kterými se diplomová práce zabývá, jsou z pohledu zpracování fronty členěny do 2 samostatných kapitol. Nejdříve se čtenář seznámí s odesíláním e-mailů pomocí fronty odesílané cronem; včetně instalace aplikace a detailním popisem implementace. Poté je stejný problém popsán za pomoci RabbitMQ. Tato technologie je popsána podrobněji. Prvním důvodem je, že je relativně nová, druhým důvodem je fakt, že nabízí mnohem více možností než cron.

V závěru je shrnuta náročnost a přínos obou knihoven, a to včetně náročnosti na zatížení serveru, rychlosti zpracování a nasazení v reálném provozu.

1.7 Výstupy a přínosy práce

Výstupem práce je dokument popisující práci s frontami v PHP. Tato práce je demonstrována na zasílání e-mailových zpráv. Detailně se věnuje technologii RabbitMQ a možnostem jejího využití k práci s frontami.

Praktickým výstupem je knihovna na posílání e-mailů. Knihovna je soubor rozhraní a základních tříd. Implementace těchto rozhraní je demonstrována v druhé knihovně, která implementuje RabbitMQ i řešení pomocí MySQL databázové fronty. Tato knihovna zároveň implementuje knihovnu Doctrine pro objektově relační mapování. Obě knihovny jsou publikovány na serveru GitHub jako open source.

2 Specifikace

2.1 Výchozí stav

Tato práce se zaměřuje pouze na newslettery, kterých se posílá v naší společnosti velké množství, a podají tedy relevantní data o náročnosti generování e-mailů. Kdykoli budu v diplomové práci mluvit o e-mailech, mám tím na mysli právě newslettery, pokud není výslovně řečeno jinak.

E-maily se generují na hlavním serveru, na kterém běží e-shop. To znamená, že při generování e-mailů se musí skript připojit do stejné databáze, ze které běží e-shop, administrace, skladová aplikace, …

Generovat nové newslettery do databázové fronty můžeme pouze v noci, když není pracovní doba skladu, není téměř žádná návštěvnost stránek a není tak téměř žádné vytížení hlavního serveru. Rozesílka začíná půl hodiny po půlnoci a končí po cca 20 hodinách. Z toho je patrno, že rozesílání emailů trvá téměř celý den.

Veškeré e-maily se mohou odesílat pouze z hlavního serveru. Je to kvůli tomu, že jsou poměrně pevně spjaty s logikou, která se tam nachází (jazykové soubory, view), a do e-mailu se navíc vkládají data tříd, které jsou třeba pro opětovné vygenerování e-mailu.

Každý typ e-mailu má vlastní šablonu, do které se vkládají dynamická data generovaná v modelu. Další důležitou vlastností je, že každý typ e-mailu je ještě vázaný na doménu, pro kterou se posílá. Každá doména má totiž unikátní design.

2.2 Základní požadavky

Hlavním cílem diplomové práce je porovnat původní koncept, přesunutí databázové fronty na jiný server (a řešit kompletně odeslání z tohoto jiného serveru) a řešit frontu pomocí služby, která je k tomu určena (RabbitMQ). Zároveň musí být aplikace, která je součástí diplomové práce, dostatečně univerzální, aby šla bez větších potíží použít i pro jiné projekty.

Hlavním cílem společnosti, ve které pracuji je, aby vše fungovalo podobným způsobem jako nyní, jen se zvýšila efektivita odesílání e-mailů.

Požadavky na zlepšení fungování jsou následující:

1) Vytvořit **nezávislý, modulární systém**, který půjde použít téměř v jakékoli aplikaci.
2) Systém by měl **umožňovat co nejjednodušeji vygenerovat hlášky pro překladatele**, a to včetně náhledu e-mailu. Při expanzi na nové domény totiž narážíme na problémy s překlady. Není pro nás úplně snadné vygenerovat všechny e-maily, které se pro danou doménu budou posílat, získat z těch e-mailů hlášky k překladu a poté je zaslat překladatelům k překladu.

3) **Zrychlení** je dalším požadavkem. Výchozí stav byl již nastíněn: newslettery se ve stávající situaci odesílají takřka celý den. Je tedy potřeba vytvořit stabilní základ pro rozesílání velkého množství e-mailů. Velkým množstvím je myšleno okolo 6 milionů e-mailů měsíčně. O finální odeslání se stará API poskytovatele hostingu, takže stačí odeslat potřebný e-mail jeho službě, prostřednictvím protokolu SMTP, a on se poté postará o všechny důležité náležitosti, které e-mail musí mít (aby například nespadl do spamového koše nebo nebyl označen za chybný/nebezpečný).

Knihovna, která je pro tento účel vytvořena, musí být dostatečně flexibilní, aby byla schopna e-maily odesílat i jinam než na současnou službu, přes kterou se e-maily odesílají, a to bez velkého zásahu.

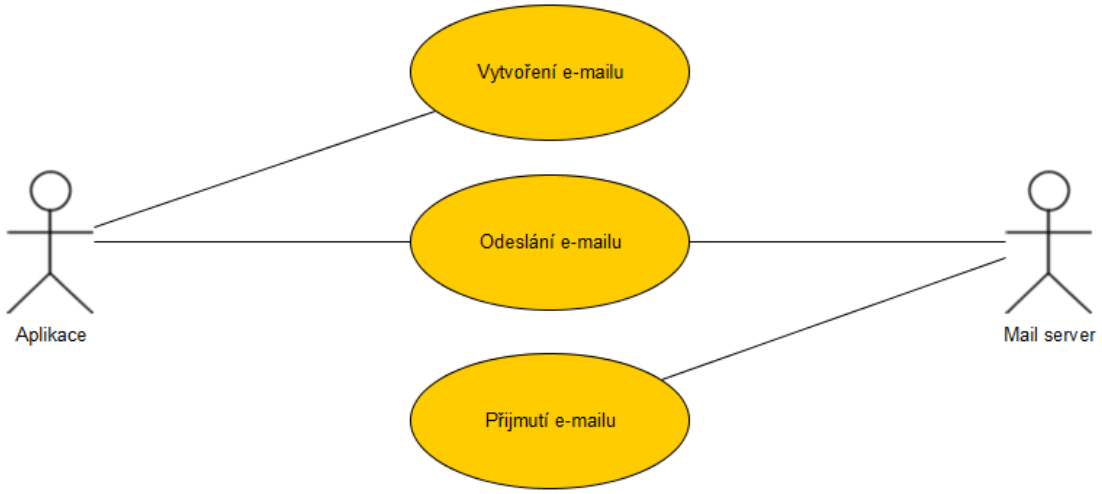
Druhů e-mailů, které je třeba rozeslat, je také relativně mnoho. Mezi nejdůležitější patří například e-maily spojené s objednávkou, e-maily s informacemi o registraci na prodejně nebo o potvrzení registrace zákazníka. Protože se e-maily rozesílají z jedné fronty v dávce, ve které může být i více než milion e-mailů, je třeba e-maily nějakým způsobem třídit dle důležitosti. Například potvrzení objednávky nebo potvrzení registrace mají větší prioritu než newslettery. Takovéto prioritní e-maily pak musejí být schopny „předběhnout“ ty méně důležité e-maily.

2.3 Základní princip

Knihovna pro rozesílání e-mailů je postavena tak, aby byla napsána pouze jednou, a používala se jak v aplikaci, tak v mail serveru. V praxi to znamená, že třídy, které jsou napsané v rámci knihovny, jsou univerzální a na každé straně dělají něco trochu jiného.

Na obrázku 1 Případ užití komunikace mezi aplikací a mail serverem je vidět, že odeslání e-mailu je jednou z hlavních funkcí jak aplikace, tak mail serveru. Z pohledu uživatele knihovny (programátora) vše končí odesláním e-mailu z aplikace. E-mail je však dále přijat frontou na mail server a až následně opravdu odeslán zamýšlenému adresátovi. Třída, která se stará o odeslání, je přitom pokaždé stejná, jen používá jiný adapter. Jednou se jedná o adapter, který vloží e-mail do fronty (v první fázi do databáze, ve druhé fázi do RabbitMQ fronty). Po druhé se přečte e-mail z fronty a odešle adresátovi pomocí jiného adapteru (PHPMailer, nativní funkce mail, Nette Mailer, …).

Tento princip velice zjednoduší vývoj, protože není třeba znát více rozhraní (aplikace-server a server-mailová schránka).



Obrázek 1 Případ užití komunikace mezi aplikací a mail serverem.

Z diagramu je také patrné, že k vytvoření e-mailu dojde na straně aplikace. Zároveň se ale požaduje odlehčení aplikační databáze z důvodu, aby se v ní e-maily neuchovávaly. **Informace pro e-mail se** tedy **musí vzít jedním ze třech následujících způsobů:**

- Aplikace bude uchovávat informace potřebné k vytvoření e-mailu a ty tedy budou duplikovány mezi aplikaci a mailový server.
    - Toto řešení je naprosto nevhodné, protože ani nesníží počet dotazů do databáze, a navíc vede k duplikování dat v databázi
- Aplikace bude mít přístup přímo k databázi mailového serveru.
    - Toto řešení má výhodu v rychlosti. Získání informací je pomalejší, než kdyby byly součástí aplikační databáze, ale je rychlejší než 3. způsob.
    - Možnou nevýhodou je relativně silné provázání aplikace a mailového serveru. Ale nemusí to nutně vadit vždy, a pakliže to nevadí, je toto řešení nejlepší.
- Připojení pomocí REST API k mail serveru, který vrátí požadované informace pomocí http GET odpovědi.
    - Velice škálovatelné řešení, které perfektně rozbije závislost aplikace a mailového serveru.
    - Toto řešení je nejpomalejší ze všech 3 variant, protože se musí nejprve zpracovat dotaz a poté postavit celá odpověď (request response).

2.3.1 Implementace prostřednictvím databázové fronty

Základ tvoří knihovna mailsender/core, což je v podstatě soubor rozhraní plus některé základní entity. Kolem této knihovny lze vystavět jakýkoli adaptér pro kterýkoli Framework.

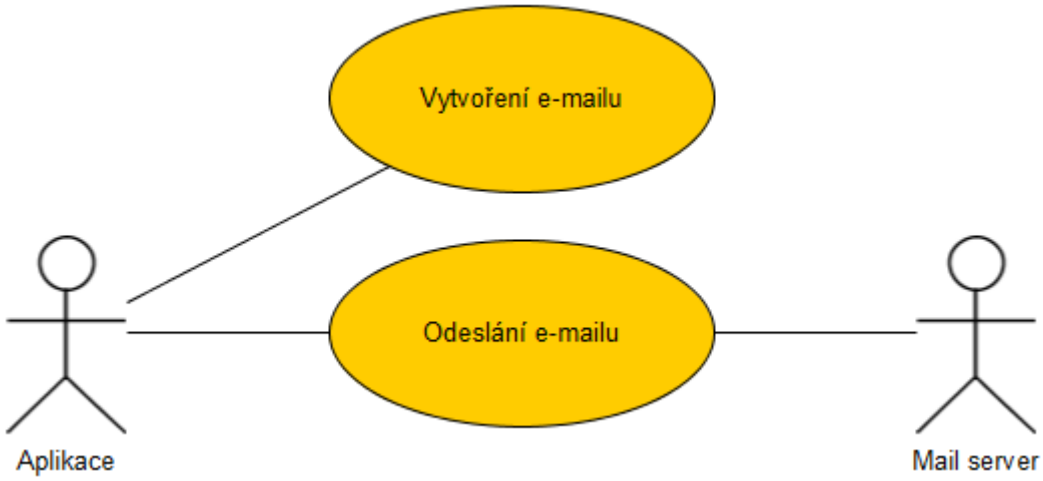
2.4 Požadavky na architekturu

Jako programovací jazyk jsem zvolil PHP, protože zbytek aplikace je v něm také naprogramován a znám jej nejlépe. PHP musí být ve verzi 7.1. Je to nejstarší stabilní verze, která již má poměrně dobrou typovou kontrolu metod a jejich parametrů (včetně podpory pro null) a zároveň se jedná o nejrychlejší verzi PHP vůbec [2]. V takovém množství e-mailů, jaké musí být knihovna schopna poslat, je důležitá každá milisekunda, a i málo procentní navýšení výkonu může být signifikantní.

Knihovna je naprogramovaná plně objektově orientovaně, podle nejlepších postupů a za použití návrhových vzorů tak, aby byla v budoucnu snadno rozšiřitelná buď pro interní účely nebo pro účely komunity, která by se ji rozhodla použít.

2.5 Případy užití

Vzhledem k poměrně úzkému zaměření je model jednoduchý a obsahuje jen málo případů užití. Všechny případy užití mají společné to, že do nich nezasahují žádní skuteční uživatelé, ale jen části systému.



Obrázek 2 Model případů užití

Prvním případem užití celého mailového systému je vytvoření e-mailu.

Tabulka 1 Případ užití

Název případu užití	**Vytvoření e-mailu**
Identifikace případu užití	UC1
Cíl případu užití	Vytvořit entitu e-mailu
Primární aktér	Aplikace
Pomocný aktér	-
Vstupní podmínky	Existuje typ e-mailu
Výstupní podmínky	Vytvořený e-mail, připravený k odeslání
Základní scénář	Krok / Role / Akce 1 / Aplikace / Připojení se do databáze e-mailového serveru a získání typu mailu. 2 / Aplikace / Doplnění dat do mailu. Dat, která nejsou vůbec nebo jsou jinak než načtená data z typu e-mailu. 3 / Aplikace / Vložení výsledného e-mailu do databáze e-mailového serveru. 4 / Aplikace / Vložení právě vloženého e-mailu do databáze i do databázové fronty.

Alternativní scénáře	Krok 4 se nahradí za vložení do fronty RabbitMQ.
	• 2 varianty
	  ○ Vložení ID na e-mail
	  ○ Vložení celého e-mailu

2.6 Diagram tříd

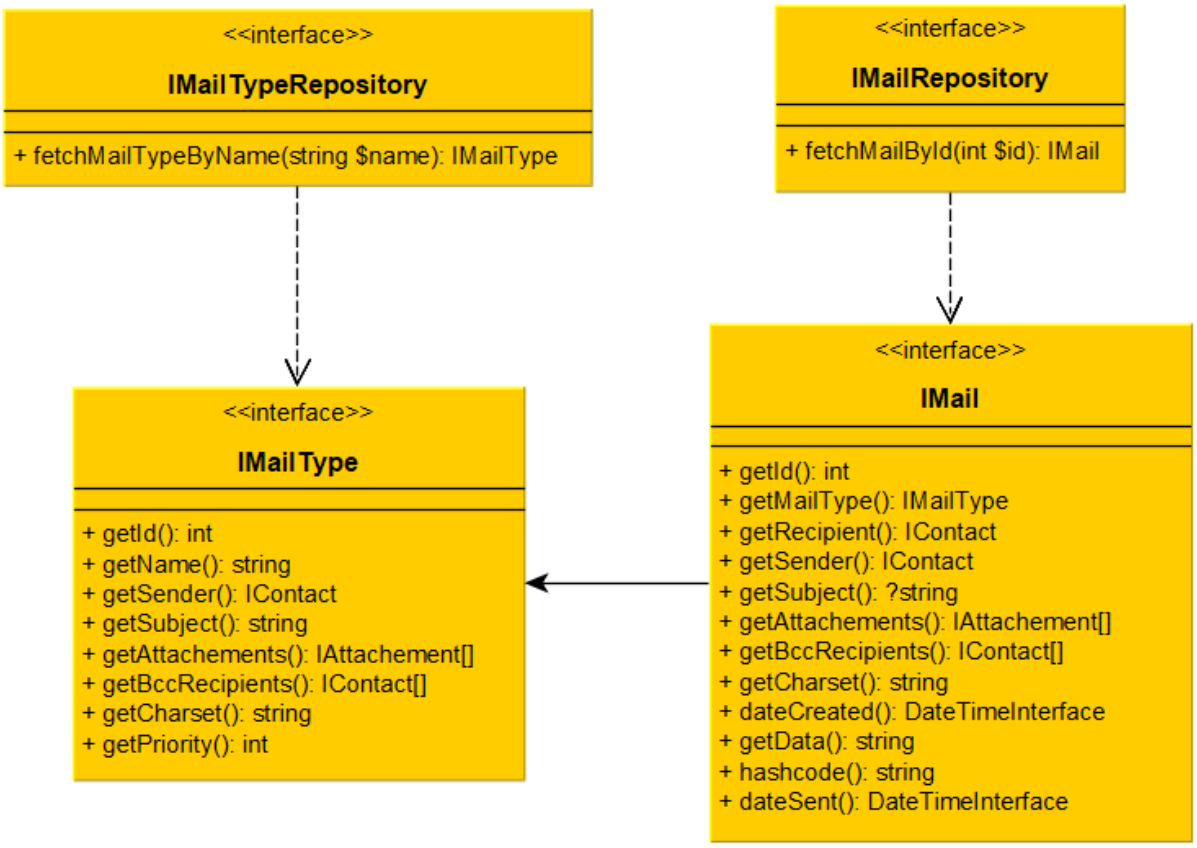
Diagram tříd jsem se rozhodl rozdělit do několika celků pro jeho rozsáhlost; na jednom obrázku by byl špatně čitelný. Diagramy tříd se věnují knihovně mailsender/core, a detailně popisují její logiku.

2.6.1 Definice UML diagramu

Protože UML definuje vazby relativně volně, na tomto místě upřesním některé typy vazeb.

- Plná šipka – Představuje situaci, kdy jedna třída obsahuje jinou třídu v atributu třídy.
- Přerušovaná šipka – Vyjadřuje závislost jedné třídy na jiné. Záviset může skrze konstruktor, set nebo get metodu.

2.6.2 Hlavní entity e-mailu



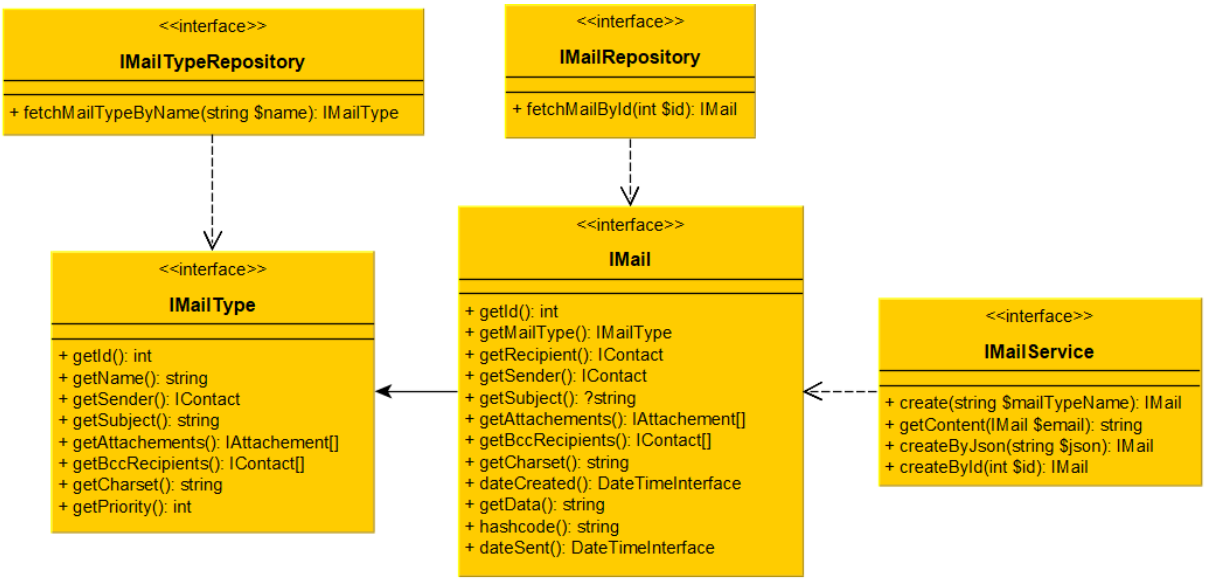
Obrázek 3 Vazba mezi typem e-mailu a e-mailem

Ústřední třídou celého systému je třída Mail. Může se jednat o jakoukoli třídu, která implementuje rozhraní IMail. Instance rozhraní IMail může být vytvořeno třemi způsoby.

1) Neexistující e-mail se vytvoří pomocí objektu, který implementuje IMailType.
2) E-mail už v databázi existuje a objekt Mail je vytvořen v objektu, který implementuje rozhraní IMailRepository, a je naplněn hodnotami z databáze.
3) Serializovaný objekt v JSON

Tvorbu e-mailu popisuje kapitola 3.6.3. Nyní je podstatné, že MailType představuje typ e-mailu. Může se jednat například o typ e-mailu „potvrzení objednávky“. Nese si v sobě i základní informace, které jsou poté předány e-mailu. Tyto informace jsou pro všechny e-maily daného typu společné, nicméně mohou být v aplikaci podle potřeby přepsány.

2.6.3 Vytvoření entity e-mailu



Obrázek 4 Vytvoření e-mailu

Vytvořit e-mail se samozřejmě může v podstatě kdekoli. Ideální je k tomu nicméně služba IMailService, která má k tomu **3 určené metody**.

2.6.3.1 Způsob první – e-mail ještě nikde neexistuje a je tedy nově vytvářen.

Zavolá se metoda create, která jako parametr přijímá název typu e-mailu. Tento název musí být v tabulce unikátní a na jeho základě se z databáze načte typ e-mailu. Vytvoří se prázdný e-mail a naplní se defaultními daty, která jsou uložena v typu e-mailu. Takto vytvořený e-mail se vrátí jako návratová hodnota z metody create. Jediné, co je nutné doplnit a nemůže být součástí typu e-mailu, je příjemce, takže zde se e-mail obohatí o příjemce. A pokud není potřeba nic jiného nastavit, může se e-mail odeslat (typicky na mailový server). Odeslání se věnuje další podkapitola.

2.6.3.2 Způsob druhý – e-mail existuje v databázi a je z ní načten

Jedná se o nejjednodušší způsob, protože se jen načte e-mail z databáze a vrátí se jako entita IMail. Takto načtená entita má v sobě už i přímo odkaz na typ e-mailu.

Tento způsob se používá při odesílání e-mailu z mailového serveru. Při použití databázové fronty je to jediný možný způsob. Při použití RabbitMQ je to jeden ze dvou možných způsobů.

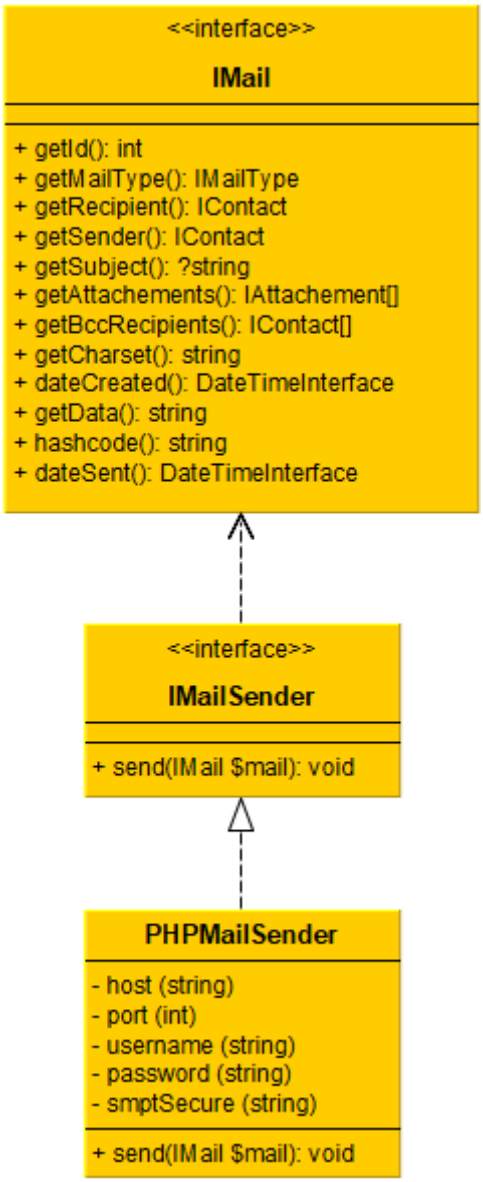
2.6.3.3 Způsob třetí – načtení e-mailu ze serializovaného JSON řetězce

Tento způsob se používá pouze při odeslání pomocí RabbitMQ. RabbitMQ dovoluje odeslat do fronty jakýkoli formát. Může se tedy do fronty předat přímo serializovaná entita e-mailu, která se potom z řetězce znovu vytvoří.

Výhodou je, že teoreticky není nutné se vůbec připojovat k databázi a rozesílka by tedy měla být rychlejší. Na druhou stranu to znamená větší požadavky na velikost RAM paměti, než když se do RabbitMQ fronty pošle pouze ID e-mailu a celý e-mail se poté načte z databáze.

Dopady těchto dvou přístupů jsou jedním z předmětů zkoumání diplomové práce.

2.6.4 Odeslání e-mailu

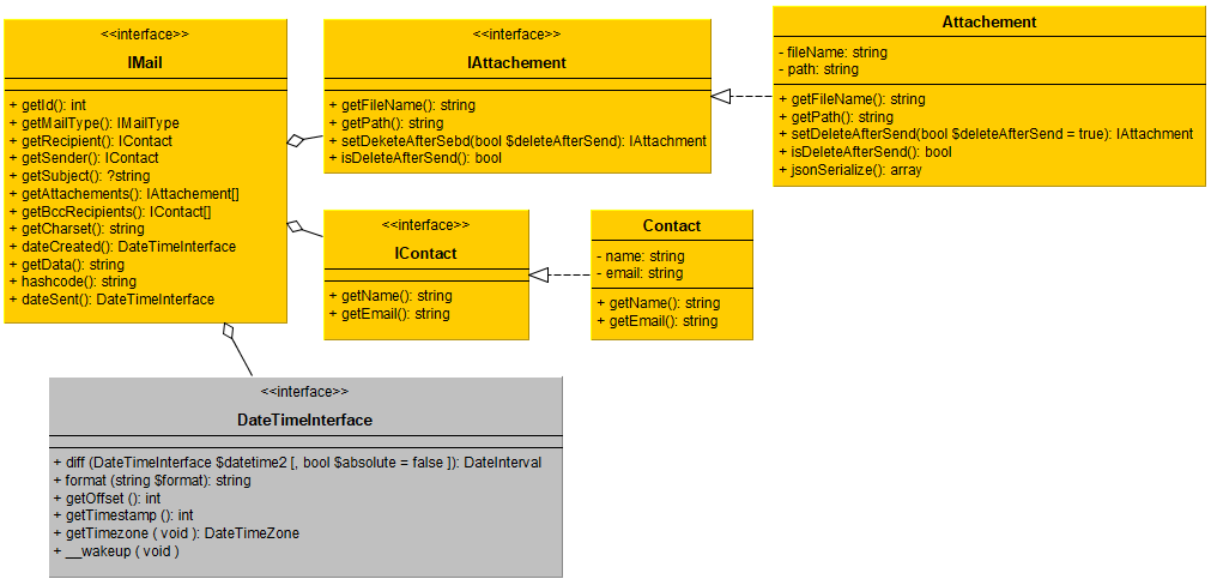


Obrázek 5 Odeslání e-mailu

O odeslání e-mailu se stará třída, která implementuje rozhraní IMailSender. Odesláním se myslí uložení do databáze mailového serveru, uložení do fronty RabbitMQ nebo i odeslání e-mailu přímo z aplikace. Všechny tyto způsoby jsou možné a třída, která to bude zajišťovat, musí pouze implementovat rozhraní IMailSender.

Na diagramu je vyobrazeno přímé odeslání pomocí třídy PHPMailSender, jež se pomocí protokolu SMTP připojí k SMTP serveru a e-mail následně odešle.

2.6.5 Struktura entity e-mailu



Obrázek 6 Struktura entity e-mailu

Diagram výše zobrazuje, z jakých prvků je tvořeno rozhraní, které definuje entitu e-mail. Pro přehlednost, je pouze odebrána vazba na typ e-mailu, která byla vyobrazena na předchozích diagramech.

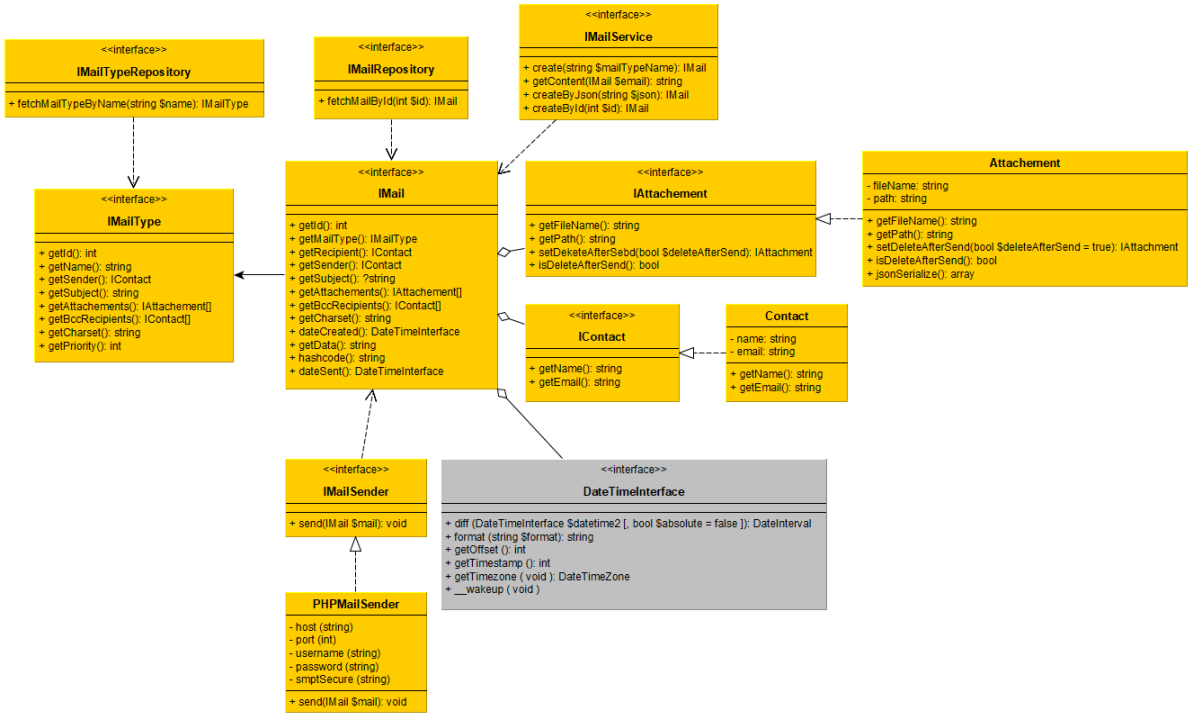
Entita, která implementuje rozhraní IMail může používat jakékoli entity pro příjemce, odesilatele a přílohy. Součástí knihovny jsou nejmenší možné entity, které splňují rozhraní a které jsou zároveň dostatečné pro odeslání formuláře.

Každá entita uživatele musí vracet jméno a e-mailovou adresu. Jméno může být i prázdný řetězec, ale nemůže zůstat zcela nevyplněno.

Entita přílohy definuje cestu k příloze a název souboru, který se má do e-mailu vložit. Zároveň si nese informaci o tom, jestli se má příloha po odeslání smazat nebo ne. Pokud bude přílohou generovaný dokument (např. faktura), tak je vhodné po odeslání e-mailu přílohu smazat, protože by soubor jinak zabíral místo na disku. Tato informace je zodpovědností entity Attachement a proto je v rozhraní striktně definováno, že tuto informaci musí být entita schopna přijmout a vrátit.

DateTimeInterface je přímo součástí jádra PHP, proto je tato třída odlišena barvou. Není součástí knihovny, ale je součástí každého interpretera PHP.

2.6.6 Komplexní pohled na knihovnu



Obrázek 7 Kompletní diagram vazeb v knihovně mailsender/core

2.7 Struktura databáze

Hlavní knihovna je víceméně soubor rozhraní, takže výsledná entita e-mailu nebo typu e-mailu může být rozšířena o další sloupce. Toto je minimální/optimální struktura, kterou musí každý e-mail splňovat, nicméně rozšířit o další sloupce se může neomezeně.

2.7.1.1 Typ e-mailu

Typ e-mailu představuje výchozí hodnoty pro e-mail. Při vytvoření entity Mail se naplní atributy právě z entity MailType. Může zároveň nést informaci o tom, jak se má vykreslit e-mail při odesílání. O tom pojednává kapitola 4.

Tabulka 2 Typy e-mailů

Sloupec	Hodnota	Popis
id	1	Auto-increment hodnota e-mailu. Generuje se automaticky v databázi.
name	Newsletter	Název typu e-mailu. Slouží pouze pro jednodušší orientaci mezi typy e-mailů.
code	newsletter.cz/2018/3/9	Unikátní název typu e-mailu. Podle této hodnoty se načte v IMailService entita MailType a naplní se Email výchozími hodnotami.
sender	{"name": "Company", "email": "newsletter@company.cz"}	Entita IContact serializovaná jako JSON řetězec. Po načtení z databáze se z ní stane opět entita.
subject	[LIKVIDACE ZÁSOB] Až -80 % v AVION Shopping Park. NEVÁHEJTE!	Předmět e-mailu
attachments	NULL	Přílohy uložené ve formátu JSON. Po načtení z databáze se z JSONu stane pole entit IAttachment.
bcc_recipients	NULL	Skryté kopie uložené ve formátu JSON, které se po načtení převedou na pole entity IContact.
charset	utf-8	Znaková sada e-mailu.
priority	0	Priorita e-mailu. Newslettery mají nejnižší prioritu zatím co například potvrzení objednávky, registrace, … nejvyšší.

2.7.1.2 E-mail

Tabulka 3 Entita s e-mail

Sloupec	Hodnota	Popis
id	30	Auto-increment hodnota e-mailu. Generuje se automaticky v databázi.
mail_type_id	1	
recipient	{"name": "Petr Olišar", "email": "olisar@company.cz"}	Entita IContact serializovaná jako JSON řetězec. Po načtení z databáze se z ní stane opět entita.
sender	{"name": "Company ", "email": "newsletter@company.cz"}	Entita IContact serializovaná jako JSON řetězec. Po načtení z databáze se z ní stane opět entita. Předdefinována z typu e-mailu.
subject	[LIKVIDACE ZÁSOB] Až -80 % v AVION Shopping Park. NEVÁHEJTE!	Předmět e-mailu. Předdefinovaný z typu e-mailu.
attachments	NULL	Přílohy uložené ve formátu JSON. Po načtení z databáze se z JSONu stane pole entit IAttachment. Předdefinované z typu e-mailu.
bcc_recipients	NULL	Skryté kopie uložené ve formátu JSON, které se po načtení převedou na pole entity IContact. Předdefinovaný z typu e-mailu.
charset	utf-8	Znaková sada e-mailu. Předdefinovaná z typu e-mailu.
date_crated	2018-03-26 13:38:45	Datum uložení e-mailu do databáze.
data	{"version": "A", "urlPiece": "likvidace-brno", "preFooter": "", "preHeader": "Platí od 22. 3. do 31. 3. nebo do vyprodání zásob. Těšíme se na Vás!", "utmCampaign": "77-2018-brezen-9-likvidace-brno", "languageCode": "cs", "urlOnlineVersion": "2018-brezen/likvidace-brno", "customerRegistered": true}	Data, která potřebuje šablona pro své vykreslení. Data jsou ve formátu JSON a v mail serveru decriptována. Jsou to data z aplikace, které si nemůže mail server zjistit sám.
hashcode	fe38db81cf0400e77d135c9e98a3c5ae	Hashcode představuje řetězec, který se musí zadat vždy, když chce uživatel zobrazit on-line verzi daného e-mailu. Je to zejména kvůli personalizovaným

		e-mailům, aby nebyly zobrazitelné nepovolanou osobou.
date_sent	NULL	Datum odeslání. Po úspěšném odeslání se upraví sloupec date_sent, abychom měli informaci, že došlo k úspěšnému odeslání e-mailu.

3 Možnosti odesílání e-mailů v jazyce PHP

Jazyk PHP má v zásadě **dvě možnosti odesílání e-mailů**:

- přes nativní funkci mail
- pomocí SMTP (stream_socket_client)

3.1 Nativní funkce mail

PHP může využívat nativní funkci `mail`, pokud má při instalaci přístup k systémovému binárnímu **sendmail**u. V php.ini se nastaví SMTP server, takže se může použít v základu i jiný server než lokální.

Následující kód je vše, co je potřeba pro použití nativní funkce `mail`. Tato funkce poskytuje jen základní funkčnost, přijímá 4 parametry a vrací boolean.

```
mail(string $to, string $subject, string $message[, string
$additional_headers[, string $additional_parameters ]]): bool
```

V dnešní době, kdy se PHP stává (poměrně rychle) striktně typovaným objektově orientovaným jazykem, je to, dle mého názoru, málo. Vzniklo proto několik knihoven, které implementují `mail` funkci a obalují ji objektově orientovaným rozhraním. Asi nejznámější knihovnou je PHPMailer. Tato knihovna vznikla v roce 2008 a nyní je v páté verzi. Kromě toho, že poskytuje kvalitní objektově orientované rozhraní, také výrazně zjednodušuje odesílání e-mailů. Například odesílání souborů přes nativní mail funkci přenáší zbytečnou složitost na programátora. Ten musí ručně nastavit content-type, ukončit řádky podle specifikace atd. Zatímco využití knihovny PHPMailer to zjednoduší na pouhý jeden řádek:

```
$email->AddAttachment( $fileToAttached, 'NameOfFile.pdf');
```

3.2 SMTP (stream_socket_client)

Pokud vývojář pracuje na aplikaci a musí při tom použít již existující e-mailový server, tak může mít problémy. Nativní funkce `mail` se může nastavit tak, aby e-mailový server byl jiný než localhost, nicméně toto nastavení nedovoluje nastavit přihlašovací údaje. Větším problémem je však to, že na jednom serveru může běžet více aplikací a každá může využívat jiný e-mailový server.

Právě v takových případech se používá připojení pomocí socketu k jiným SMTP serverům. Takové spojení mohou opět zprostředkovat externí knihovny, jako například PHPMailer. Připojení pomocí socketu umožňuje zadat přihlašovací údaje a připojit se tak téměř k jakémukoli externímu e-mailovému serveru.

3.3 Přímé odesílání

Nejjednodušší možností, jak odeslat e-mail, je odeslání v průběhu zpracování skriptu. Jedná se o lineární kód a vykoná se v přesně stanovený okamžik. Navíc uživatel, kterému má přijít e-mail, ví, jestli se odesílání podařilo nebo ne.

Obrovským negativem je však právě dané lineární zpracování v rámci běhu skriptu, který odbavuje požadavek návštěvníka. Skript, a tedy i návštěvník, musí čekat, až se e-mail kompletně sestaví a odešle. To velmi zpomaluje zpracování skriptu. Dalším a mnohem větším problémem je, že pokud se z nějakého důvodu odeslat e-mail nepodaří, tak je e-mail nenávratně ztracen.

Oba tyto problémy řeší postup, který uloží e-mail do fronty a pomocí cronu (nebo jiné technologie schopné zpracovat frontu) se odešle později.

3.4 Pomocí fronty ukládané do databáze

Tento způsob poměrně elegantně řeší problémy, které jsou způsobené přímým odesíláním e-mailů. Pokud se má odeslat e-mail, tak se místo toho, aby byl poslán přímo na výstup (sendmail), uloží do databáze. Insert do databáze je řádově rychlejší operace, než odeslání e-mailu a čekání na výsledek. Takto uložený e-mail se poté odešle pomocí cronu. Pokud z nějakého důvodu odeslání selže, tak se daný e-mail z databáze nesmaže a není problém se pokusit jej odeslat znovu později nebo po vyřešení závady.

Tato metoda tedy řeší problém přímého odesílání. Avšak přináší nový problém. Z podstaty věci se cron spouští pravidelně v nastaveném intervalu. Což by bylo v pořádku, pokud by se e-maily odesílaly rovnoměrně v průběhu celého dne. Ony se ale odesílají velmi variabilně. V noci se neposílají téměř vůbec (nebavíme-li se o plánované rozesílce – například newsletterů), naopak dopoledne je špička. Server ale nemůže být nastaven tak, že v noci nepošle nic a dopoledne se bude spouštět cron každých deset vteřin. Toto platí zejména tehdy, pokud je daná webová aplikace celosvětová. Musí se tedy učinit nějaký kompromis, například pouštění cronu každou minutu.

To znamená, že každou minutu se server pokusí odeslat e-maily, které má uložené v databázi ve frontě. V noci se každou minutu spustí skript, který se podívá do databáze, jestli je co odeslat, úplně zbytečně. Přes den to zase nemusí být dostatečné, protože se například mohou odesílat newslettery. Těch může být velké množství (řádově statisíce až miliony) a nejde odeslat půl milionu e-mailů jedním skriptem, takže se vezme část a ta se odešle. Odeslaná část, se může odeslat dřív než za minutu a pak by bylo žádoucí dřív spustit nový skript. Může se ale také odeslat za více než minutu a pak běží 2 skripty přes sebe. Není možné server optimálně vytížit. Tento problém řeší specializovaný program RabbitMQ.

3.5 Pomocí RabbitMQ

RabbitMQ řeší všechny 3 problémy. Uživatel nemusí čekat, než se mu vygeneruje stránka, protože komunikace s RabbitMQ frontou je velice rychlá. Mail se neztratí, i když se ho nepovede poslat (záleží na nastavení). A server je optimálně vytížen – RabbitMQ je škálovatelný.

Princip je velice jednoduchý a vlastně podobný jako při ukládání do databáze. E-mail se uloží do RabbitMQ fronty, která se zpracovává, a jakmile na něj přijde řada (nebo ihned, pokud ve frontě nic není), tak se odešle.

4 Odesílání e-mailů pomocí databázové fronty

Pokud je žádoucí odesílat e-mail později, musí se někam uložit. Míst, kam jej uložit, je mnoho, avšak jsou pro to různě vhodná. Například soubor. Ten se pro ukládání příliš nehodí, protože souborové operace trvají příliš dlouho. E-mail se může uložit i do paměti. V takovém případě by práce s ním byla extrémně rychlá. Problém ovšem nastává v momentě, kdy dojde k problému a server takzvaně spadne. To vyústí k nenávratné ztrátě e-mailu, protože po restartu už v paměti samozřejmě nebude.

Mnohem lepší nápad je využít databáze, kterou již aplikace pravděpodobně stejně využívá. Pro tyto účely je navíc výborně optimalizovaná díky atomickému zápisu a rychlému čtení.

4.1 Hlavní logika zpracování databázové fronty

Základem je skript, který si vytáhne záznamy z databáze, které zpracuje. Pointou je, že tento skript musí být spouštěn často. Problém může nastat, pokud se spustí stejný skript vícekrát naráz tak, že se navzájem překrývají. V takovém případě může dojít k situaci, že první skript si načetl data a začal je zpracovávat. Druhý skript si načte stejná data (nebo jejich podmnožinu) a začne také zpracovávat. Takovýmto způsobem pak může dojít ke zpracování dat dvakrát nebo může dokonce dojít k neočekávané chybě, jelikož data, která se očekávají v průběhu zpracování, jsou již odstraněna nebo změněna. Tomuto problému se říká konkurenční přístup.

Aby se předešlo problému s konkurenčním přístupem, tak si každý skript nejdříve zamkne předem definovaný soubor záznamů (například 50), které označí nějakým příznakem. Dobrým zvykem je tento příznak pojmenovat job _id, protože se jedná o unikátní identifikátor (id) a jedná se o určitou činnost, která bude se souborem dat prováděna (job). Vytvoření zámku je jeden dotaz do databáze, a tedy se jedná o atomickou operaci, kdy nemůže dojít k žádnému průniku změněných dat. Výsledkem je, že každý skript si takto atomicky zamkne data a až poté je začne zpracovávat.

Cron dokáže spustit příkaz každou minutu. V případě nutnosti odbavení velkého množství záznamů se může stát, že je potřeba spustit příkaz častěji. K tomu slouží příkaz sleep, který spouštěný příkaz uspí a po stanovené době jej znovu spustí. Příklad takového cronu může vypadat například takto:

```
* * * * * sleep 3; php /bin/console mail:send -s klerk2 1> /dev/null

* * * * * sleep 6; php /bin/console mail:send -s klerk3 1> /dev/null

* * * * * sleep 9; php /bin/console mail:send -s klerk4 1> /dev/null
```

4.2 Generování newsletterů do fronty

Diplomová práce, jak již bylo řečeno v úvodu, je zaměřena na rozesílání e-mailů pomocí fronty. Toto rozesílání je demonstrováno na případu rozesílání newsletterů, které jsou schopny (vzhledem k jejich velkému počtu) opravdu důkladně otestovat navrženou knihovnu.

V našem případě byl tlak ze strany společnosti na co největší rychlost. Provázanost systémů není tak důležitá, proto jsem zvolil možnost, že aplikace, ze které se e-maily generují, má přímý přístup do e-mailové databáze.

4.2.1 Kde se vezmou newslettery

Newslettery se načítají z hlavní databáze. Možností, jak omezit výběr e-mailových adres je více, například odeslání newsletterů podle poštovního směrovacího čísla, podle státu, domény, aj. Ve výsledku se však všechny e-maily generují ze dvou velkých tabulek. Jedná se o tabulky obsahující zákazníky (*customers*) a odběratele newsletterů (*newsletter_emails*). Obě tabulky mají miliony záznamů a generování proto zabere určitý čas.

Generování ve zkratce probíhá tak, že se v prvním kroku vytvoří view pro každou skupinu, která se posílá. Tyto view se poté pomocí UNION spojí a vznikne jeden velký pohled, ze kterého se tahají všechny newslettery k odeslání.

Ve smyčce se poté prochází jeden e-mail za druhým a plní se do databázové fronty pomocí MailSenderu, který implementuje IMailSender, jež jsem popisoval v kapitole 2.6.4. MailSender, který je použit pro testování, je jen velmi mírně upravený PHPMailSender z knihovny, která je součástí diplomové práce.

4.2.2 Rychlost generování a odesílání na SMTP server

Rychlost odesílání byl jeden z hlavních důvodů, proč jsem celý nový e-mailový systém navrhl. Od marketingového oddělení totiž přišel požadavek na zrychlení rozesílky newsletterů, která byla skutečně tristní. Velice omezující to bylo zejména tehdy, když bylo potřeba upozornit zákazníka, že v některém státě bude končit výprodej zboží. Rozesílka pro české zákazníky totiž trvala téměř 24 hodin. Skript pro rozesílání e-mailů (ne jenom newsletterů) se spouštěl jednou za 20 vteřin. Jednou se dokonce stalo, že e-mail zákazníkům na Slovensku dorazil až poté, co akce skončila.

Problém nebyl na straně providera. Ten neměl potíže s odesláním emailů na základě toho, jak jsme je plnili do jeho fronty. Problém byl zejména v tom, že jsme měli pouze jeden přístupový bod, jeden kanál, kterým jsme mu byli schopni poslat e-maily k odeslání. Prvním krokem tedy bylo zvýšení počtu kanálů, kterými bychom mohli e-maily odesílat. Provider pro nás tedy připravil aplikaci, shodou okolností založenou na RabbitMQ, se 3 přístupovými body. To pro nás znamenalo značné zrychlení rozesílky, kterou jsme byli schopni touto cestou snížit na cca 10 hodin. Skript pro rozesílku e-mailů se spouštěl pro každý přístupový bod po 20 vteřinách, takže 3x častěji než v předchozím případě.

Přetrvával nicméně druhý problém, což byla velká zátěž na výkon serveru. Rozesílka běžela z hlavní databáze a hlavního webového serveru. Vzhledem k této skutečnosti bylo interně zakázáno generovat

newslettery přes den a ustálilo se pravidlo, že veškeré newslettery se začínaly generovat o půl 1 ráno, kdy bylo nejmenší zatížení serveru.

Z tohoto důvodu, jak již bylo uvedeno v předchozích kapitolách, jsem napsal kompletně novou aplikaci pro rozesílání e-mailů spolu s databází, které jsem umístil na jiný server. Jádro dané aplikace je předmětem této diplomové práce. Díky ní se mohla rozesílka ještě téměř o polovinu zrychlit. Nová aplikace totiž přistupuje k méně vytížené databázi, využívá moderní a rychlejší PHP 7.1 a běží na méně vytíženém serveru. Díky tomu netrvala rozesílka 20 vteřin, ale výsledně pouze necelých vteřin 10. Takže jsme byli schopni nastavit v cronu spouštění pro každý skript 5x za minutu a dostali jsme se k rozesílce o objemu cca 40 mailů za vteřinu.

4.3 Instalace

Knihovna, která využívá cron k rozesílání e-mailů, vyžaduje ke svému běhu minimálně PHP 7.1, MySQL a composer. Pro vývoj na Windows je součástí knihovny Vagrantfile, který nainstaluje Ubuntu jako virtuální stroj. Technologie, které by měly být nainstalované, jsou uvedeny ve složce ansible, a pomocí stejnojmenné technologie jednoduše nainstalovatelné.

Vagrant slouží pro jednoduché nastavení vývojového prostředí. Umí pracovat s různými virtuálními stroji, nejlépe však s VirtualBoxem od Oraclu. Po instalaci VirtualBoxu a Vagrantu je třeba restartovat počítač a nejlépe se ještě v biosu ujistit, že je povolena podpora virtualizace. Spuštění samotného stroje se provede příkazem

```
vagrant up
```

První spuštění trvá poměrně dlouho, protože se musí stáhnout image operačního systému (v našem případě Ubuntu).

4.3.1 Ansible

Ansible je předpis pro instalaci programů. Je možné nastavit, jaké programy, aplikace a příkazy se mají provést po sobě. Ansible poté zkontroluje, jestli je program nainstalovaný a jestli je ve správné verzi. Pokud nainstalovaný není nebo se od sebe liší verze, tak se nainstaluje verze podle předpisu. Je to tedy ideální nástroj pro instalaci totožného prostředí na více strojů.

Další důležité příkazy jsou

```
vagrant halt // Zastaví virtuální stroj

vagrant destroy // smaže image virtuálního stroje
```

Vagrant, obzvláště ve spojení s Ansible, je velice mocný nástroj s mnoha možnostmi. Ve většině případů však příkazy pro základní spuštění a vypnutí aplikace postačí.

4.3.2 Composer

Po instalaci je třeba se přepnout do příkazové řádky nainstalovaného linuxu pomocí příkazu *vagrant ssh*. Přepnout se do složky /var/www/mail-server a spustit *composer update*. Composer je balíčkovací nástroj, který je schopný si udržovat stromovou strukturu závislostí. Tento příkaz si do paměti stáhne všechny závislosti, které jsou deklarované v souboru *composer.json*, a pokud nenastane konflikt, tak tyto závislosti stáhne. Další velice důležitá funkce composeru je vytvoření tzv. ClassLoader souboru. Tento soubor si udržuje mapování všech tříd a souborů. V PHP je totiž možné mít v jednom souboru více tříd a všechny soubory, které se používají, se musejí vyžádat ve spouštěcím souboru. A právě to dělá composer za programátora.

4.4 Implementace knihovny

Knihovna pro rozesílání e-mailů je v maximální míře striktně tipovaná a potřebuje tedy ke svému běhu PHP 7.1. Veškeré závislosti (kde je to možné) jsou vázány na rozhraní (interface), takže je velice flexibilní a rozšiřitelná o vlastní implementace.

Knihovna je hodně abstraktní právě pro to, aby byla rozšiřitelná a nevyžadovala žádnou konkrétní technologii. Jedná se zejména o připojení do databáze, ale jde i o samotné entity. Například interně ve firmě máme implementovanou entitu *MailType* se sloupci *facade* a *template*. Je to proto, že newslettery mají sice pokaždé stejnou strukturu dat, ale jsou vykreslovány jinou šablonou. Knihovna však musí jít použít i na jednoduché účely, klidně i jen pro jeden e-mail odesílaný napřímo. Proto jsou definované pouze minimální parametry a každý si může rozšířit kteroukoli část knihovny tak, jak potřebuje.

Knihovna složená pouze z rozhraní by vlastně neukázala nic o konkrétním použití. Proto vznikla ještě jedna knihovna: **mailsender/doctrine-mail-sender**. Tato knihovna využívá pro komunikaci s databází ORM (Object-relational mapping) vrstvu Doctrine. Obsahuje i 2 jednoduché skripty, které se starají o uložení dat do databáze a odeslání e-mailu. Na této konkrétní implementaci v této kapitole ukáži, jaká je má představa o použití knihovny.

4.4.1 Vytvoření e-mailu

E-mail může být vytvořen v podstatě kdekoli. V knihovně mailsender/doctrine-mail-sender je tím místem třída MailDemoService. Tato třída momentálně využívá 2 způsoby vytvoření e-mailu, je však nachystaná i na třetí způsob. Tato třída závisí na 4 třídách, přičemž ta poslední je volitelná.

```
public function __construct(

      IMailRepository $mailRepository,

      IMailTypeRepository $mailTypeRepository,

      IMailTypeFacadeFactory $mailTypeFacadeFactory,

      ?SerializerInterface $serializer = null

)
```

- $mailRepository – slouží k načítání e-mailů z databáze podle ID
- $mailTypeRepository – slouží k načítán typu e-mailu z databáze podle jména
- $mailTypeFacadeFactory – slouží k vytvoření instance fasády, která zpracuje data z databáze a vytvoří tělo e-mailu
- $serializer – je volitelná závislost třídy. Používá se pro vytvoření entity z jsonu.

E-mail se musí vytvořit na základě svého typu (například podle toho, jestli se jedná o potvrzení objednávky, newsletter, registrační e-mail apod.).

```
$mailType = \Mailsender\DoctrineSender\MailDemoFacade::class

$mailType = $this->mailTypeRepository

      ->fetchMailTypeByName($mailTypeName);

$mail = new Mail($mailType);
```

Takto vytvořený e-mail si převezme veškeré výchozí nastavení z typu e-mailu. Vyvstává tu jeden problém a tím je, že entita není validní ihned po vytvoření. E-mail nemá smysl bez příjemce a typ e-mailu o něm nemůže nic vědět. Je proto nutné pro vytvořenou entitu vždy zavolat metodu ->*setRecipient()*. Tento problém by šel vyřešit pomocí přepravky na data, která by měla stejné atributy jako entita e-mailu, a až ta by se předávala do konstruktoru e-mailu. Takto nastíněné řešení mě však přišlo zbytečně složité. Pokud s aktuálním postupem budou problémy, tak není problém implementovat výše nastíněné řešení.

Pro vytvořenou entitu se ještě musí předat příjemce a případně data pro e-mail.

```
$mail->setRecipient(

      new \Mailsender\Core\Entity\Contact('Petr Olisar',
'petr.olisar@gmail.com')

);

$mail->setData(json_encode([

     'dearCustomer' => 'Drahousek zakaznik',
```

```
]));
```

V tuto chvíli je e-mail validní a může se odeslat. Data jsou volitelná a předávají pouze (v jakékoli struktuře) informace pro odesílací část aplikace. Může se jednat o číslo objednávky, jméno zákazníka, nakoupené produkty apod.

4.4.2 Uložení e-mailu do fronty

O odeslání e-mailu se stará třída, která implementuje rozhraní IMailSender. Konkrétně QueueMailSender, který dostane na vstupu entitu a vloží ji do databáze. V Doctrine je možné uložit do databáze entitu, která je namapovaná na tabulku. Protože však s tímto postupem byly problémy s neakceptovatelným narůstáním potřebné paměti, tak jsem zvolil uložení napřímo pomocí PDO. Ihned po vytvoření e-mailu v databázi se vytvoří záznam pro frontu, kam se vloží id právě vytvořeného e-mailu.

```
$this->connection->prepare('INSERT INTO mails (`mail_type_id`,
`recipient`, `sender`, `subject`, `charset`, `data`, `hashcode`,
`date_created`) VALUES (?,?,?,?,?,?,?,NOW())')->execute($data);
```

```
$this->connection->prepare('INSERT INTO mail_queue (`mail_id`)
VALUES (?)')->execute([(int) $this->connection->lastInsertId()]);
```

QueueMailSender má tedy závislost pouze na připojení do databáze. V tuto chvíli je e-mail odeslán do databáze. Může se stát, že není potřeba „odkládat“ e-mail do fronty a je vhodno jej odeslat přímo. V takovém případě se nahradí třída QueueMailSender například za připravenou třídu PHPMailSender a e-mail se odešle rovnou.

4.4.3 Odeslání e-mailu

Odeslání e-mailu je v principu velice jednoduché. Načte se definovaný počet e-mailů z databáze, které nemají příznak job_id a zároveň u nich nebyl překročen definovaný počet pokusů o odeslání, a zkusí se jeden po druhém odeslat. Pokud nenastane chyba přímo v odesílání, tak se celá fronta zpracuje i kdyby byly v e-mailech vážné chyby bránící odeslání. Pouze vadné e-maily nebudou odeslány a zvýší se počítadlo pokusů. Toho je dosaženo obalením celého procesu odeslání e-mailu do try-catch bloku. Úspěšně odeslané e-maily se z fronty pro odesílání smažou, ale v databázi zůstávají, aby je bylo možné zobrazit například v online režimu.

Odesilatelem e-mailu může být jakýkoli objekt implementující rozhraní *Mailsender\Core\MailSenders\IMailSender*. V základu je implementováno odeslání pomocí nejpoužívanější knihovny PHPMailer. Objekt, který se má starat o odeslání, musí pouze implementovat metodu *send()*, která přijímá celý e-mail v objektu implementujícím rozhraní *IEmail*. To zajišťuje velice rozšiřitelný systém a kdokoli, kdo chce použít knihovnu, může implementovat své vlastní funkce; případně celé knihovny pro odeslání e-mailu.

Zjednodušeně probíhá proces odeslání e-mailu následovně:

```
$this->lockJobId($jobId);

$emailIds = $this->selectLockedMailIds();

foreach($emailIds as $emailId) {

      Try {

            $mail = $service->createById($emailId);

            $mailSender->send($mail);

            $success[] = $mail->getId();

      }

      Catch (Exception $e) {

            $error[] = $emailId;

      }

}

$this->deleteSuccessed($success);

$this->updateErrors($error);
```

4.5 Problémy při implementaci databázové fronty

Při nasazování se vyskytly dva hlavní problémy. Problém s RAM pamětí a problém s odpojením připojení do databáze

4.5.1 Problém s RAM pamětí

V testovacím režimu na testovacích datech bylo vše funkční, takže jsem byl schopen nasadit knihovnu do ostrého provozu. Jako první se posílaly menší newslettery o celkovém počtu do 200 tisíc e-mailů. Tyto newslettery byly odeslány relativně bez problémů. V grafu, který je generován nástrojem Grafana, bylo vidět, že generování spotřebovává čím dál více paměti. Vše se přesto odeslalo.

Problém přišel při překročení hranice 300 tisíc newsletterů. Okolo tohoto počtu se totiž vyčerpala paměť a plnění databázové fronty tak spadlo. Pomocí příkazu pro vypsání alokované paměti jsem určil místo daného problému a vyřešil ho.

Problémy byly ve skutečnosti dva.

4.5.1.1 Problém první

Problém tkvěl v tom, že rozesílka trvá velice dlouho (řádově jednotky hodin) a zpracovává statisíce řádků. Ve společnosti používáme MySQL databázi a ta pro PHP používá **ovladač mysqlnd**, který nahrazuje starý libmysql. Tento ovladač může pracovat **ve dvou režimech**:

1) **Buffered queries (dotazy přes vyrovnávací paměť)** – Všechny řádky z databáze se ukládají do vyrovnávací paměti. To má tu výhodu, že je potom možné pracovat s celou sadou výsledků. Nevýhodou je samozřejmě skutečnost, že celá sada výsledků je načtena v paměti a pokud je potřeba zpracovat velké množství řádků databáze, tak paměť může dojít.
2) **Unbuffered queries (dotazy, které se neukládají do vyrovnávací paměti)** – Naproti tomu dotazy, které se neukládají do vyrovnávací paměti, uvolní RAM paměť ve chvíli, kdy už nejsou potřeba. Programátor tím pádem nikdy nemá k dispozici celou sadu výsledků. Tento způsob je vhodný právě pro zpracovávání velkého množství dat, které se jen přečtou z databáze a zahodí. Navíc při nevyužití vyrovnávací paměti nejsou k dispozici některé vlastnosti PDO.

Některé databázové ovladače dovolují přepnout buffrovaný a nebuffrovaný přístup k datům i pouze pro jediný dotaz. U MySQL to není možné. Proto jsem musel vytvořit druhé připojení do databáze, které je přepnuto do unbuffred režimu, a přes toto připojení číst data. Veškerá ostatní komunikace probíhala pomocí jiného připojení.

Protože jsem potřeboval z jedné databáze pouze číst data a do druhé je zapisovat, tak řešením bylo nastavení unbuffered režimu pro databázi, ze které se pouze čte. Ve skutečnosti to však vytvořilo problém 4.5.2.

4.5.1.2 Problém druhý

Problém byl v naší interní knihovně, která obaluje práci s databází. V jednom místě vznikal objekt, který nikdy garbage collector neuvolnil. Takže skript měl neustále větší a větší nároky na paměť až do doby, kdy paměť nakonec došla.

Oprava potom byla poměrně jednoduchá. Stačilo nahradit naši obálku za přímý přístup k objektu PDO, který se v PHP stará o práci s databází:

```
$this->database->getPdo()->prepare('INSERT INTO mails
(`mail_type_id`, `recipient`, `sender`, `subject`, `charset`,
`data`, `hashcode`, `date_created`) VALUES (?,?,?,?,?,?,?,NOW())')-
>execute($data);
```

```
$this->database->getPdo()->prepare('INSERT INTO mail_queue
(`mail_id`) VALUES (?)')->execute([(int) $this->database-
>getLastInsertId()]);
```

Po této změně bylo využití RAM paměti konstantní. Nyní generování newsletterů spotřebovává konstantně okolo 8 MB RAM.

4.5.2 Problém s odpojením připojení do databáze

Vyřešení problému s RAM pamětí vytvořilo problém nový. Tímto problémem bylo odpojování se od databáze. Vše bylo způsobeno tím, že jsem vytvořil dvě připojení do databáze. Jedno připojení pouze pro čtení dat, druhé s možností zápisu. Připojení s možností zápisu po celou dobu generování newsletterů nepokládalo žádné dotazy do databáze. Když však poté mělo dotaz do databáze položit, již bylo připojení odpojeno. Následkem toho celá aplikace spadla. Na konci rozesílky bylo potřeba uložit statistická data a pokud to bylo nastaveno, tak začít odesílat další sadu newsletterů.

Vše vedlo k tomu, že se newslettery vygenerovali a správně odeslali, ale statistická data jsme nedokázali uložit. Mírnou komplikací bylo také to, že PHP ne všude vyhazuje výjimky a pokud se odpojí databáze, tak příkaz excute vrátí false, ale zároveň i warning. Byl jsem tedy nucen potlačit warning pomocí konstrukce se zavináčem a číst pouze true/false hodnotu.

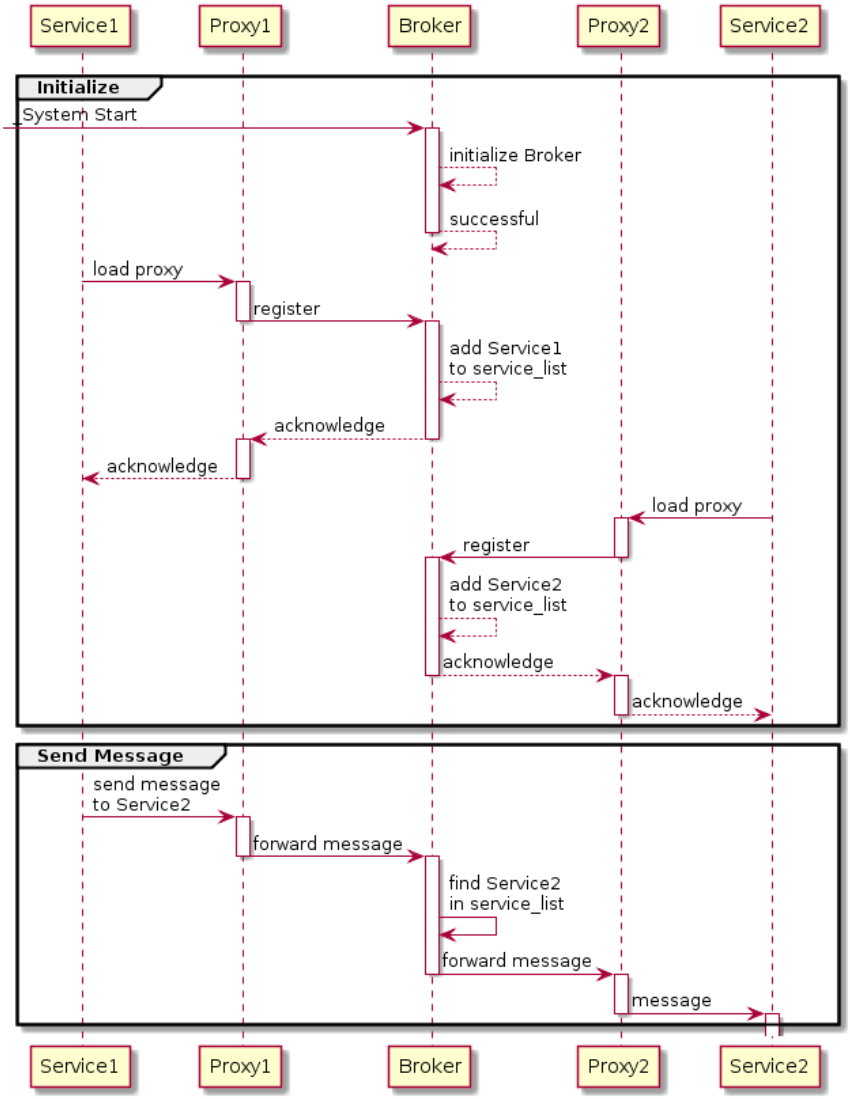
```
public function execute(): Statement {

	// …

	if(@$pdoStatement->execute($this->params) === false) {

		if(--$this->reconnectCounter <= 0) {

			$e = new ConnectionException($this->sql, 0);

			$e->setReconnectCount(self::MAX_RECONNECT_COUNT -
		$this->reconnectCounter);

			throw $e;

		}

		$this->connection->reconnect();

		return $this->execute();

	}

	// …

}
```

5 Technologie RabbitMQ

RabbitMQ je software, označovaný jako message broker (někdy také jako Message-oriented middleware). Původními vývojáři byla společnost Rabbit Technologies Ltd., což byl společný podnik společností LShift a CohesiveFT. Byla založena roku 2007. V roce 2013 byla založena společnost Pivotal a ve stejném roce se projekt RabbitMQ stal také její součástí [3].

RabbitMQ je napsaný v jazyce Erlang a původně podporoval pouze protokol AMQP (Advanced Message Queuing Protocol). Nyní podporuje AMQP verze 0.9.1 i 1.0, MQTT (Message Queuing Telemetry Transport) a STOMP (Streaming Text Oriented Messaging Protocol).

5.1 Jak funguje message broker

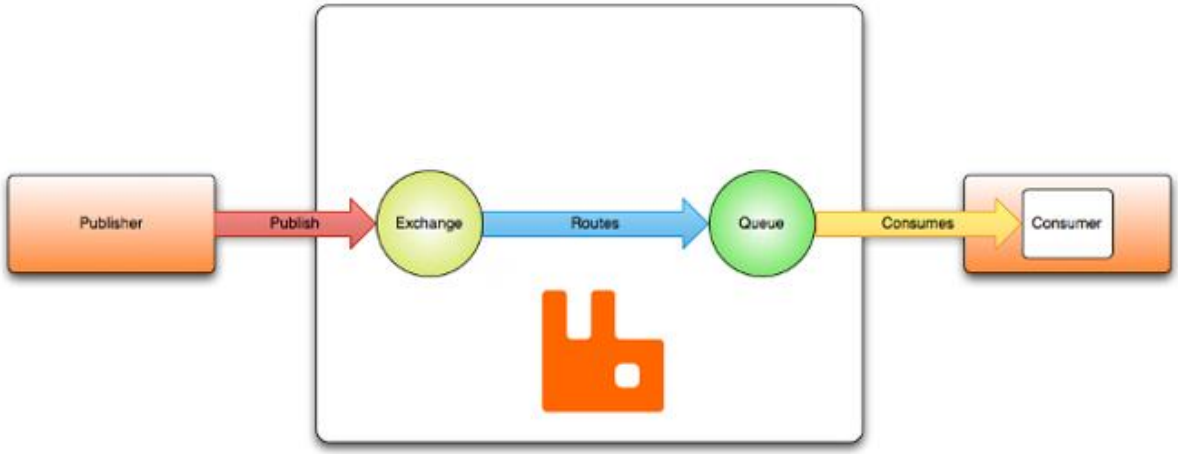


Obrázek 8 Sekvenční diagram vzoru message brooker [4]

Obrázek číslo 8 popisuje základní chování všech brokerů pro správu front. Konkrétní implementace se může lišit, nicméně tento základ je pokaždé stejný.

Prvním krokem je připojení k brokeru a nastartování spojení. To musí udělat jak Publisher (Service1), tak Consumer (spotřebitel, Service2). Pokud se nepřipojí Consumer, tak fronta bude neustále narůstat, protože nebude nikdo, kdo by ji zpracovával. Poté je možné posílat zprávy do fronty brokera, který je poté pošle dál do Consumera.

5.2 Jak funguje RabbitMQ



Obrázek 9 Jak funguje RabbitMQ [5]

Jedna část aplikace (Publisher) chce poslat zprávu jiné části aplikace. Může se jednat i o jinou aplikaci nebo dokonce i o úplně jinou technologii. Pokud k tomuto úkolu zvolí RabbitMQ proces, bude to vypadat následovně:

Publisher se připojí k Exchange a pošle jí zprávu. Exchange na základě typu zprávy a routovacích pravidel rozhodne, do které fronty zprávu pošle, a pošle ji tam. Ve frontě je zpráva tak dlouho, dokud se k ní nepřipojí consumer, který bude danou frontu zpracovávat.

Pokud by se stalo, že se zprávy budou tvořit rychleji než zpracovávat, tak to ničemu nevadí, protože se bude plnit fronta a consumer ji nakonec celou zpracuje. Rychlost zpracování fronty se může upravit přidáním dalších consumerů, kteří odbavování fronty zrychlí.

Zpráva, která se posílá, se skládají z metadat a z těla zprávy. Tělo zprávy se předává consumeru. Metadata slouží brokeru k rozhodnutí, co se s danou zprávou má udělat. Tělo zprávy je pro broker de facto neviditelné a používá pouze metadata, které zná. Metadata, které nezná, mohou být použity pro příjemce zprávy jako další pomocná data. [6]

5.2.1 Potvrzování zpráv

Protože AMQP je internetový protokol, je tudíž poměrně nespolehlivý. Zpráva se nemusí dostat do fronty kvůli mnoha různým příčinám. Stejně tak se nemusí z fronty řádně odbavit. Proto má AMQP

nástroj, jak řešit nespolehlivost internetového protokolu. Tím je potvrzování zpráv (message acknowledgements).

Pokud je zpráva odeslána do fronty, tak si Publisher může vyžádat potvrzení. O něco to zpomalí zpracování zprávy ve frontě, protože se čeká na zápis do fronty a odpověď. Na druhou stranu to dává programátorovi 100% jistotu, že se zpráva řádně uložila do fronty.

Úplně stejně to funguje i naopak. Zpráva je poslána consumerovi, který ji zpracuje, a broker může být nastaven tak, že dokud nedostane potvrzení o zpracování zprávy, tak ji nesmaže. Jediné, co v tomto případě může nastat je, že zpráva bude zpracována vícekrát. To může být ve chvíli, kdy z nějakého důvodu selže potvrzení, avšak zpráva se zpracuje v pořádku. Pokud se nepovede zprávu řádně zpracovat, tak může Consumer poslat brokeru tzv. negativní potvrzení (Negative Acknowledgement) [6].

5.2.2 Publisher

Jak již bylo řečeno výše, Publisher posílá data (zprávu) do RabbitMQ fronty. Existují dvě možnosti, jak zprávu do fronty doručit. Buď napřímo nebo přes Exchange. Napřímo to znamená, že Publisher komunikuje rovnou s frontou a Exchange obejde. Druhou možností je doručení zprávy do fronty přes Exchange, který ji poté distribuuje dál (do správné fronty či více front).

Doporučovaným řešením je druhý přístup. Hlavním důvodem je fakt, že v budoucnu se nemusí nic měnit na aktuálním chování, pokud se poté rozhodne, že se má se zprávou udělat něco dalšího nebo něco jiného. Exchange totiž může podle nastavení poslat stejnou zprávu do více front nebo na základě různých příznaků rozhodnout, do které fronty se má doručit. Nastavení Exchange je popsáno dále.

Výhodou odeslání přímo do konkrétní fronty je jednoduchost implementace a teoreticky vyšší rychlost, jelikož se nikdo nemusí na základě ničeho rozhodovat, jak zprávu routovat [6].

5.2.3 Exchange

Nastavení Exchange se definuje především na straně aplikace. Je to z toho důvodu, aby měl vše plně v režii programátor aplikace a on sám si mohl definovat, co se má s jeho zprávou stát. (Do jaké služby či jiné aplikace ji potřebuje poslat ke zpracování.) Při definici Exchange je vhodné také nastavit předpis pro vytvoření fronty. Pokud se spustí jako první Publisher, tak se vytvoří Exchange i fronta, a zpráva se do fronty rovnou uloží.

To, že si může programátor aplikace nastavit chování Exchange, mu dává velkou volnost. Není totiž odkázán na správce serveru (případně správce RabbitMQ) pokud nemá přístup do jeho administrace. Existuje možnost, že nastavení budou chybná a Exchange nebude správně fungovat. Tyto chyby v nastavení nejsou časté a velice často naznačují, že je chyba v konfiguraci Exchange [6].

Exchange má poměrně rozsáhlé možnosti nastavení, proto mu detailně věnuji celou jednu následující podkapitolu.

5.2.4 Queue

Fronta je to místo, kde se hromadí zprávy, které se nestihnou odeslat přímo consumeru. Pokud se zprávy generují pomaleji, než se zpracovávají, tak se fronta vlastně nikdy nevyužije a vždy jsou zprávy delegovány přímo na consumery.

5.2.5 Consumer

Consumer je skript, který zpracovává frontu. Jedna fronta může mít více consumerů, ale jeden consumer může zpracovávat pouze jednu frontu (alespoň v PHP). Může být napsaný v jakémkoli jazyce, který umí komunikovat pomocí protokolu, který podporuje RabbitMQ. To zabezpečí propojení dvou aplikací (například ERP - java systém s e-shopem - php).

Consumer by měl definovat předpis pro vytvoření fronty stejně jako Publisher. Důležité je, aby byl předpis stejný. Jestliže by byla fronta nakonfigurována jinak, nepodařilo by se k RabbitMQ vůbec připojit. Potom je možné spustit consumer jako první. On si vytvoří frontu, ke které se připojí a „poslouchá“ na ní. Až se spustí Publisher, tak si vytvoří Exchange (fronta už je vytvořena) a odešle zprávu, která se dostane rovnou do consumera.

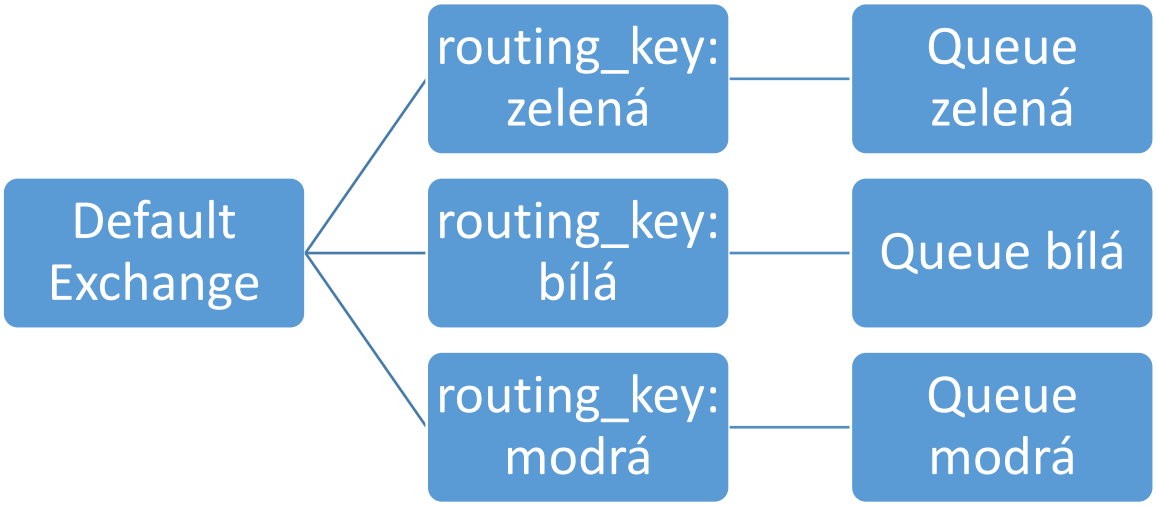
5.3 Exchange

Exchange je v podstatě takový router pro RabbitMQ. Zprávy, které na něj přijdou, mohou být směrovány do několika front, avšak i do žádné fronty. Takové zprávy jsou nenávratně ztraceny. Na tuto situaci si musí každý programátor dát velký pozor. Je to také důvod, proč je dobré v Publisheru dobré definovat jak Exchange, tak frontu. Pak totiž nehrozí, že je opomenuto propojit Exchange s frontou. V jazyce RabbitMQ se tomuto propojení říká bindování [6].

Existuje 5 typů Exchange, které mají **3 základní nastavení**.

1) **Jméno** Exchange – jedná se zároveň o přístupový bod, ke kterému se Publisher připojuje. Pro každý Exchange musí být tedy jméno unikátní.
2) **Trvanlivost** – Exchange je označen jako trvanlivý, dokud má nastaven příznak durable. Znamená to, že pokud z jakéhokoli důvodu „spadne“ RabbitMQ (nebo celý server), tak po opětovném nastartování bude stále Exchange existovat. Opakem durable je transient, který existuje pouze po dobu života RabbitMQ. Jakmile se ukončí instance RabbitMQ, tak transient Exchange zaniká.
3) **Automatické mazání** – Jde o příznak, že pokud je odpojena poslední fronta, tak se Exchange smaže.

5.3.1 Výchozí (default) Exchange

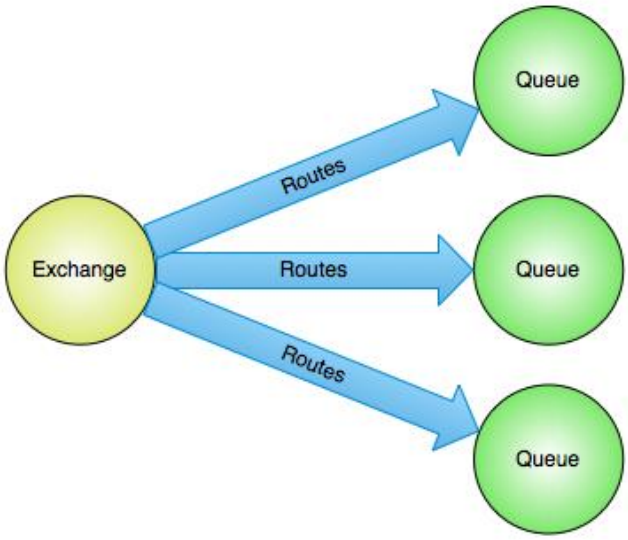


Obrázek 10 Výchozí (default) Exchange

Výchozí Exchange je taková, která nemá žádný název. Slouží de facto k přímému posílání zpráv z Publishera do fronty (i když to tak ve skutečnosti není).

Při definici Exchange se nechává prázdný název, vyplní se ale routovací klíč. RabbitMQ automaticky vytvoří (pokud ještě neexistuje) frontu a připojí ji na Exchange. Takto vytvořená fronta má stejný název jako je routovací klíč Exchange.

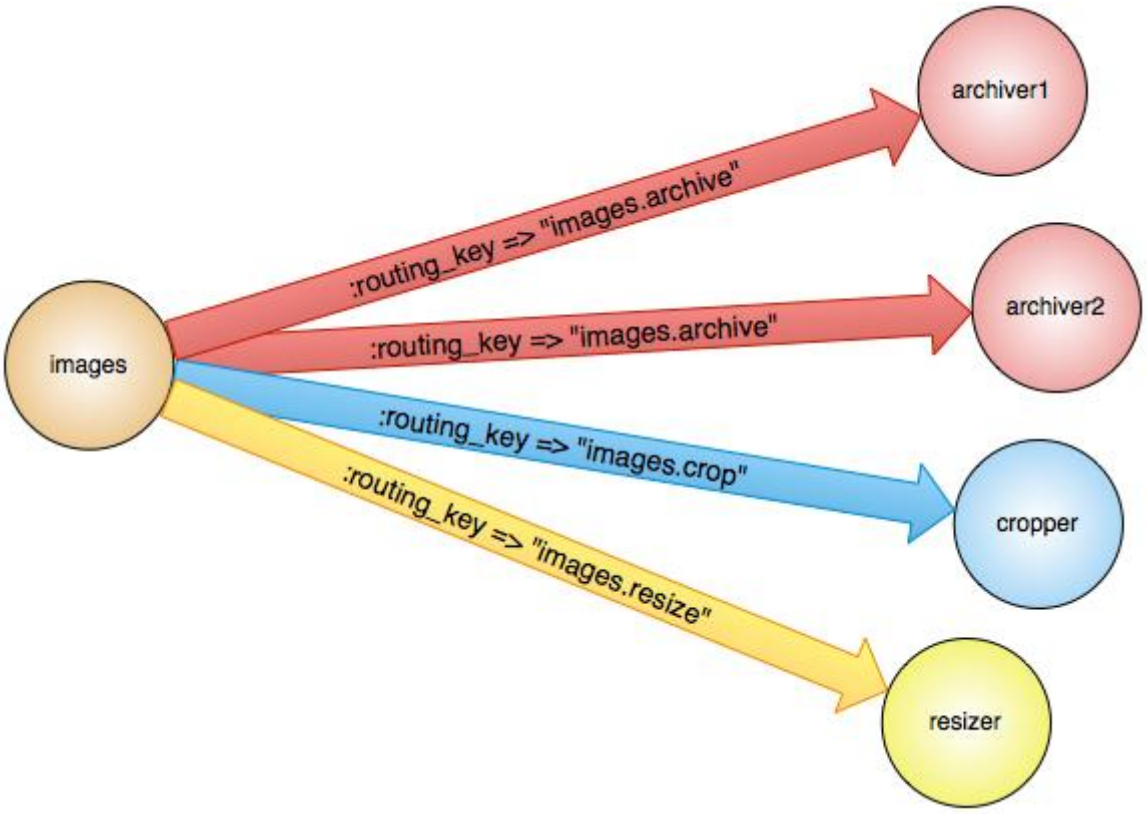
5.3.2 Fanout Exchange



Obrázek 11 Fanout exchange [6]

Fanout je nejjednodušší ze všech typů Exchange. Říká vlastně jen to, že zpráva, která na něj přijde, bude rozdistribuována do všech front. Zpráva je uložena v paměti jen jednou a ve všech frontách existuje pouze odkaz na toto místo. Využití paměti se reálně zvedne, protože metadata okolo zprávy se nesdílí, avšak zpráva samotná je v paměti pouze jednou.

5.3.3 Adresovaný (direct) Exchange



Obrázek 12 Adresovaný (direct) Exchange [6]

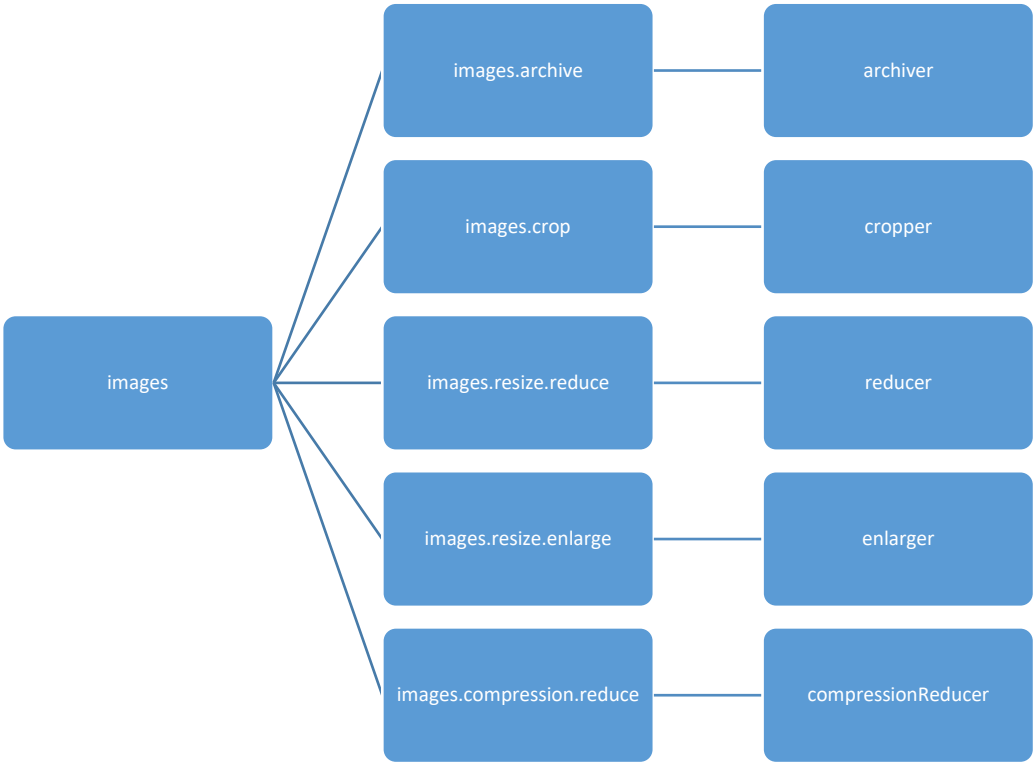
Toto je pravděpodobně nejčastější nastavení Exchange. Dovoluje přesně cílit, do jaké fronty se má zpráva dostat. Vše se řídí výhradně routovacími klíči.

Existuje mnoho možností, jak řídit adresovaný Exchange. Níže jsou pro ukázku uvedeny **4 nejčastější možnosti**:

1) **Jeden routovací klíč směřuje na jednu frontu**
    a. images.resize → resizer
2) **Jeden routovací klíč směřuje na více front** (takový malý fanout)
    a. images.archive → arcive1, archive2
3) **Dva routovací klíče směřují na jednu položku.** Tento případ se v praxi využívá, pokud existuje routovací klíč, který ukazuje na frontu, a z nějakého důvodu je potřeba jej přejmenovat. Vytvoří se proto druhý routovací klíč, který ukazuje na totožnou frontu.
    a. images.resize →resizer
    b. Není vyobrazen na obrázku images.reduction → resizer

4) **Jeden routovací klíč ukazuje na jednu frontu a druhý ukazuje na stejnou frontu jako první, plus navíc ještě na jednu další.**
   a. images.resize → resizer
   b. iamges.thumbnail → resizer
   c. images.thumbnail → cropper

5.3.4 Tématický (topic) Exchange



Obrázek 13 Tématický (topic) Exchange

Tématický Exchange je velice podobný adresovanému. Může se chovat jako fanout Exchange, jako adresovaný Exchange i jako cokoli mezi tím.

Oproti adresovanému typu přidává **2 speciální znaky**. Těmi znaky jsou **mřížka** (#) a **hvězdička** (*).

- # - znamená 0 až ∞ slov
- * - znamená přesně jedno slovo

Jinými slovy, název routovacího klíče může být oddělen tečkami. Daných teček je možno použít neomezeně mnoho. Například je možné zacílit veškeré změny velikosti použitím hvězdičky. Zápis by poté vypadal takto: *images.resize.**.

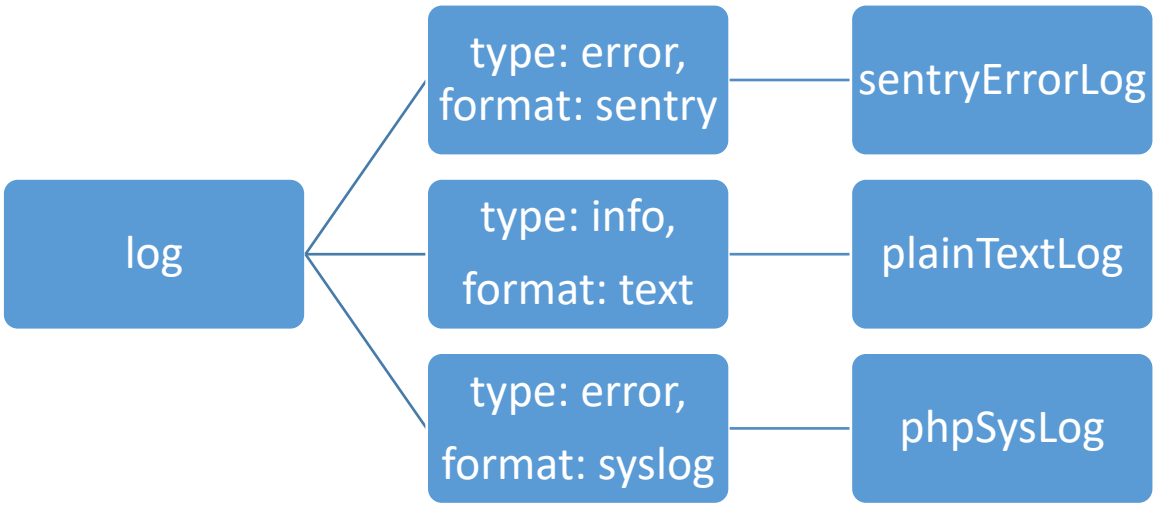
Hvězdička nahrazuje právě jedno slovo. Proto pokud by bylo zapsáno *images.**, tak to znamená, že se vybere z obrázku výše fronta na základě routovacích klíčů: *images.archive*, *images.crop*. Pokud by bylo potřeba obrázek zmenšit jak po stránce množství pixelů, tak po stránce komprese, vybere se routovací klíč, který zajišťuje zmenšení nehledě na „subkategorii“: *images.*.reduce*.

Je-li potřeba předat zprávu do všech front, které pracují s obrázkem, použije se speciální znak mřížky. Zápis poté vypadá takto: *images.#* a zpráva se tak předá všem výše uvedeným frontám.

Z výše uvedeného vyplývá, že zápis pouze samotné mřížky se chová naprosto stejně jako fanout Exchange.

5.3.5 Headers Exchange

Jedná se o velice podobný způsob směrování jako tematický Exchange. Headers Exchange routuje zprávy **na základě hlaviček**. Speciální hlavičkou *x-match* se nastavuje, jestli se mají shodovat všechny hlavičky (*all*), což je výchozí hodnota, nebo alespoň jedna (*any*).



Obrázek 14 HeadersExchange

Při vytvoření zprávy se nastaví hlavičky, kterými je možné přesně určit, kam bude zpráva směřovat. Mějme 3 zprávy:

1) Zpráva: type: error, format: sentry
    a. Předá se pouze frontě, kterou zparcovává sentryErrorLog.
2) Zpráva: type:info, format: syslog
    a. Zpráva se nesměruje nikam, protože jako základní hodnota x-match je all a neexistuje žádný předpis pro takovou frontu.
3) Zpráva: type: error, format: text, x-match: any
    a. Zpráva bude směrována do všech třech front, protože jedna z jejich podmínek projde každou z nich.

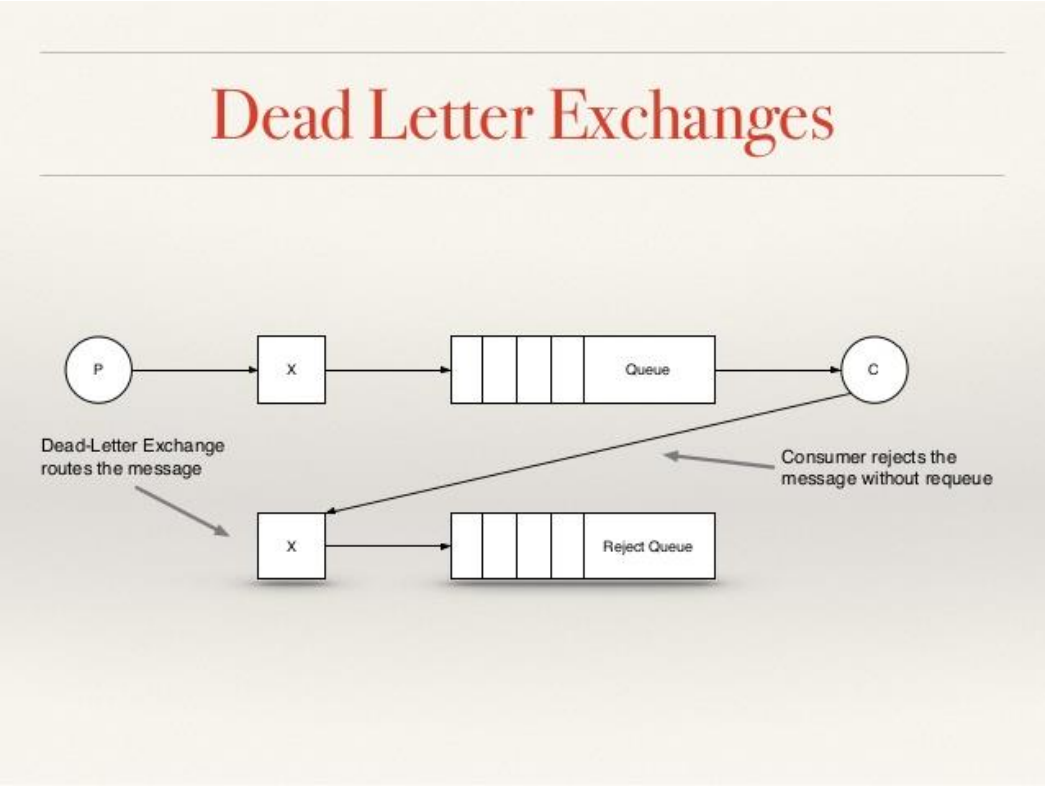
Tento způsob směrování je nejlepší použít v případě, kdy je možné rozdělit zpracovávané zprávy dle více kategorií. Například logování. Je možné logovat do souboru, do specializovaných programů,

systémových logů atp. Zároveň je možné rozlišit logy dle jejich úrovně na chyby, upozornění, informační nebo debugovací logy.

5.3.6 Dead letter Exchange

Poslední type Exchange je dead letter Exchange. Pokud se zprávu nepodaří doručit kvůli jednomu z následujících **tří důvodů**, tak může být označena jako mrtvá zpráva.

1) Zpravá byla zamítnua
2) Zprávě vypršela platnost (TTL – time to live)
3) Fronta byla naplněna a není možné do ní vkládat další zprávy



Obrázek 15 Dead letter Exchange [7]

Mrtvé zprávy jsou směrovány na speciální Exchange a jsou jim předány doplňující hlavičky. Na základě těchto hlaviček Exchange rozhodne, která fronta zpracuje mrtvou zprávu. Typickým příkladem může být zařazení zpět do fronty. Pokud se ani napodruhé nepovede zprávu doručit, tak lze například odeslat e-mail (nebo jiná notifikace programátora) s chybnou zprávou, aby nastalou situaci mohl programátor vyřešit [7].

5.3.7 Exchanges

Jak je vidět, Exchange je velice mocný nástroj RabbitMQ jak zpracovávat zprávy a odesílat je do správných front. Neumím si ani představit situaci, která by s poskytnutými možnosti nešla vyřešit. Zároveň také existuje mechanismus, jak se postarat o nedoručitelné zprávy tak, aby nezanikly, a vždy

se s nimi stalo to, co se stát má, nebo aby byl alespoň informován zodpovědný člověk, který je schopen nastalý problém vyřešit.

5.4 Queue (Fronta)

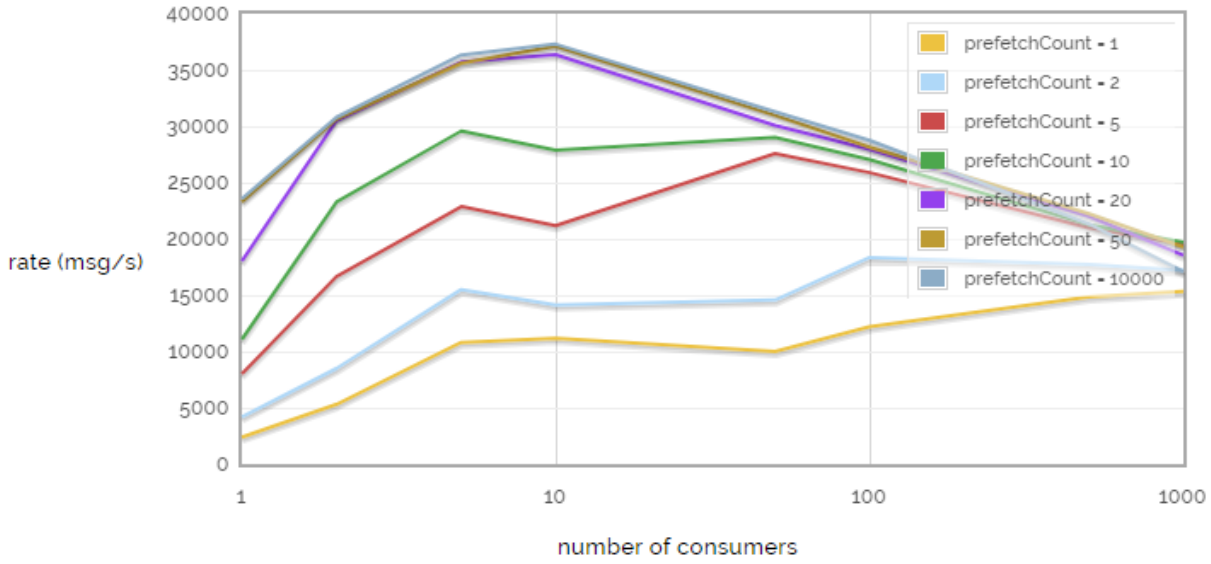
Fronta je místo, kde se hromadí a čekají zprávy, než je odbaví consumer. Exchange směruje zprávy na fronty, proto má mnoho nastavení, jak zprávu správně nasměrovat. Fronta se o zprávu musí řádně postarat, proto má zase mnoho nastavení, co se zprávou udělat v závislosti na různých okolnostech.

Základní nastavení [8]**:**

1) **Jméno fronty** – jedná se zároveň o přístupový bod, ke kterému se consumer připojuje. Musí být tedy jméno pro každou frontu unikátní.
2) **Trvanlivost** – durability funguje stejně jako u Exchange. Pokud se RabbitMQ restartuje, tak fronta „přežije“ tento restart. Pokud je nastaven příznak durable na false, tak fronta zaniká spolu s vypnutím RabbitMQ serveru.
3) **Automatické mazání** – Pokud se odpojí poslední consumer, tak se fronta smaže.
4) **Arguments** – různé nastavení. Může být specifické pro různé rozšíření RabbitMQ. Jde o nastavení, jako je např. délka života zprávy ve frontě, priorita consumerů nebo maximální délka fronty.

5.4.1 Prefetch

Jedno z nejdůležitějších nastavení fronty je hodnota prefetch. Znamená, kolik zpráv se má vyhradit pro konkrétního consumera. Tyto zprávy se přiřadí jednomu danému consumeru. Jiný consumer je již nikdy nedostane, a to i kdyby zrovna neměl žádnou zprávu ke zpracování.



Obrázek 16 Závislost počet consumerů a prefetch count [9]

Graf závislosti počtu consumerů a prefetch caunt ukazuje, že zvedání počtu consumerů je efektivní jen do určité chvíle. Obecně se asi nechá říct, že optimální počet consumerů je 10 a hodnotu prefetch count nemá smysl nastavovat vyšší než 20. Jakmile se vytvoří více než 10 consumerů, začne klesat počet zpracovávaných zpráv za sekundu, jelikož nároky na zprávu, kterému consumeru poslat kterou zprávu, začnou zpomalovat proces přidělování.

Prefetch count 1 má tu vlastnost, že každému consumeru přiřadí vždy právě jednu zprávu. Zprávy jsou tedy zpracovávány rovnoměrně. To může být rychlejší pro déle trvající malé množství zpráv.

Mějme 21 zpráv, které je třeba rozdělit 3 consumerům. Zpracování každé zprávy trvá 5 sekund a hodnota prefetch je 10. Znamená to, že consumer 1 i 2 budou zpracovávat zprávy po dobu 50 sekund. Consumer dostane jednu zprávu. Doba zpracování je tedy celkem 50 sekund, protože zpracování zpráv probíhá paralelně a všechny zprávy jsou zpracovány ve chvíli, kdy se zpracuje poslední zpráva.

Pokud bychom nastavili prefetch na 1, tak každý consumer by dostal 7 zpráv a zpracování by tak trvalo 35 sekund, což je zrychlení o téměř jednu čtvrtinu.

5.4.2 Negativní potvrzení

Negativní potvrzení řekne RabbitMQ, že se zpráva nepodařila zpracovat. Parametrem negativního potvrzení je, jestli se má zpráva znovu vložit do fronty, nebo jestli se má označit za mrtvou (a postará se o ni Dead letter exchange).

Pokud je nastaven prefetch víc než na 1, tak se může stát, že zprávy jsou již pro consumer vyhrazeny, ale on náhle skončí. Takové zprávy jsou vráceny zpět do fronty a přiřazeny jinému consumeru.

5.5 K čemu se používá

RabbitMQ je technologie, která se stará o zpracování fronty. Frontu lze využít k mnoha činnostem software. Jestliže zobecníme důvody použití fronty (minimálně ve světě PHP), vyplývají z toho **2 hlavní impulsy k použití**:

5.5.1 Zrychlit fungování aplikace

Jedním z důvodů je zrychlení zpracování aplikace. To se týká například mé aplikace na rozesílání newsletterů. Newsletterů je ohromné množství a kvůli marketingovým akcím (i jiným důvodům) je nutné dostat e-mail k zákazníkovi co nejdříve. Právě s tímto úkolem mi pomohl RabbitMQ. Mohl jsem paralelizovat zpracování rozesílky a místo jednoho skriptu se odesílal e-mail z 6 skriptů zároveň. Oproti předchozímu řešení je to téměř desetinásobné zrychlení. Podrobné statistiky budou uvedeny v závěru.

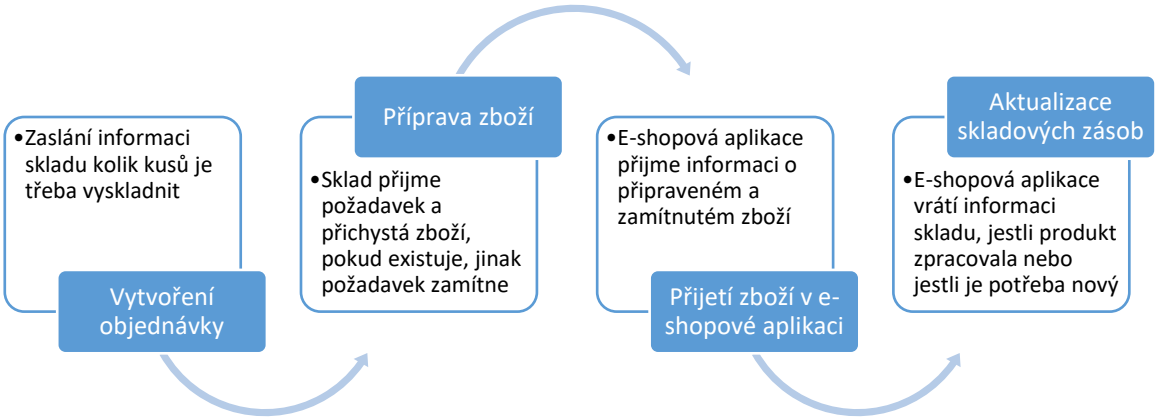
5.5.2 Docílit zpracování v reálném čase

Druhý hlavní okruh využití je zpracování v reálném čase. Do fronty se v tomto případě zpravidla posílají zprávy, které není možné zpracovat ihned. To má typicky dva důvody:

1) Stávající aplikace neobsahuje funkcionalitu, která je ke zpracování potřeba a je třeba zprávu předat na jinou aplikaci/systém.
2) Zpracování zprávy trvá neúnosně dlouho a není kritické ji zpracovat v reálném čase. Proto se zpráva odsune do fronty a zpracuje se později

5.5.2.1 Příklad využití v komunikaci mezi skladovou a e-shopovou aplikocí

Oba tyto problémy jsou řešitelné pomocí RabbitMQ. Mějme například komunikaci mezi dvěma systémy: skladový systém a e-shopová aplikace. E-shopová aplikace se stará o vychystání produktů pro expedici zákazníkovi. Sklad se stará o skladovou zásobu zboží a vyskladňuje produkty pro e-shopovou aplikaci. Celý proces může zjednodušeně vypadat jako na obrázku níže.



Obrázek 17 Proces komunikace mezi skladovou a e-shopovou aplikací

Zákazník vytvoří objednávku. Nyní se musí zaslat příkaz do skladové aplikace, aby mohli skladníci vyskladnit produkt. Komunikace mezi skladovou a e-shopovou aplikací může probíhat jakkoli. Data se můžou přenášet pomocí XML dokumentu, jedna aplikace může mít přístup přímo do databáze druhé, řešení pomocí RPC (Remote procedure call) atp. Všechny tyto způsoby mají společnou jednu vlastnost. Není stoprocentně zaručeno, že vzdálený server bude odpovídat. Čekat na odpověď se může i poměrně dlouhou dobu a podstatný je fakt, že není nutné zpracování v reálném čase. Čím dříve se sklad dozví, že má produkt nachystat, tím lépe, ale není důvod blokovat zákazníka s tímto problémem. Informace o vyskladnění se tedy odloží do fronty.

Po vyskladnění musí skladový systém říct tomu e-shopovému, jestli mu nachystal zboží pro zákazníka nebo jestli zboží není skaldem (pak je ale na to většinou nutné zákazníka upozornit, protože je předpokládáno, že zboží skladem je a zákazník s tím tedy počítá).

Posledním krokem je potvrzení přijetí zboží. Protože objednávek může být skutečně hodně a může dojít k jistým chybám, může se stát, že zboží bylo označeno jako vyskladněné, ale na straně e-shopové

aplikace nedojde k žádnému zpracování = zboží fyzicky vyskladněno nebylo. I tato informace se musí jako korekce dostat zpět do skladového systému.

Pokud je objednávek málo, tak nastíněná situace řeší problém číslo 1 – aplikace neobsahuje potřebnou funkionalitu.

Pokud objednávek začne přibývat a proces se díky tomu začne zpomalovat, může se spustit více consumerů, kteří zpracování úkolu paralelizují a zkrátí tak čas mezi vytvořením objednávky a vytvořením požadavku na skladový systém pro vyskladnění zboží. Toto zrychlení, pokud byl předchozí problém řešen pomocí RabbitMQ fronty, je pouhým přidáním dalšího consumera a nevyžaduje to žádný zásah programátorů.

5.6 Integrace RabbitMQ a PHP

RabbitMQ má na oficiálních stránkách návod pro 11 programovacích jazyků nebo frameworků. Klientskou knihovnu nebo vývojový nástroj jde však použít snad ve všech jazycích. Jen na stránkách, na kterých společnost vyvíjející RabbitMQ zmiňuje různé alternativy, jich vyjmenovává cca 70 [10].

Pro PHP se na daných stránkách uvádí šest knihoven. Knihoven je ale ve skutečnosti mnohem víc. Na GitHubu lze nalézt 615 knihoven (k 22. 4. 2018), které se nějakým způsobem týkají RabbitMQ [11].

Z výše uvedeného je patrné, že RabbitMQ je velice rozšířený a není problém nalézt pro jeho implementaci dostatek hotových knihoven. Já jsem pro čistotu kódu zvolil oficiální knihovnu php-amqplib.

Ukázku použití knihovny uvádím v kapitole 6.

5.7 Nevýhody

Ačkoli je RabbitMQ perfektní nástroj pro řešení určitých, výše nastíněných, problémů, tak má zajisté i své nevýhody/nástrahy. A je dobré o nich vědět.

5.7.1 Protokol TCP

Internetový protokol TCP (Transmission Control Protocol) může být nespolehlivý. Protokol jako takový sice garantuje „spolehlivé doručení ve správném pořadí“, ale server může postihnout výpadek, nebo může dojít k jiným problémům při přenosu po síti. Právě kvůli takovým účelům je v AMQP protokolu možnost schválit, respektive neschválit přijetí a zpracování zprávy. Pro důležité zprávy je nutné zajistit manuální schválení/neschválení v komunikaci směrem k Publisheru i consumeru.

5.7.2 Zátěž systému

RabbitMQ není výkonnostně nikterak přehnaně náročný. Avšak zpracovatelé zpráv mohou představovat problém. Jeden consumer nepředstavuje velkou zátěž. Pokud je ale potřeba zpracování

zprávy paralelizovat a zapne se víc consumerů, poté je nutno hlídat využití systémových zdrojů. Toto není zcela problém RabbitMQ, přesto však musí mít programátor na paměti, že k takovéto situaci může dojít.

5.7.3 Složitost implementace

Ve chvíli, kdy už je RabbitMQ server spuštěný a aplikace se k němu může připojit, je implementace velice jednoduchá. Problém tkví ve fázi, „než se aplikace připojí k serveru". Jestliže vše porovnám vůči frontě, která je řešená pomocí databáze, tak práce s RabbitMQ frontou je rozhodně mnohem náročnější. Vytvoření databázové fronty totiž představuje pouze vytvoření tabulky a zpracování oné tabulky v kódu.

Vytvoření RabbitMQ fronty představuje: Nainstalování RabbitMQ serveru, nainstalování php-amqplib knihovny (která zajišťuje připojení k serveru) a až nyní zpracování zprávy. V dnešní době by neměla instalace php-amqplib balíčku představovat problém, protože existuje balíčkovací nástroj Composer. Pro legacy (staré) aplikace, které Composer nepoužívají, to určitý problém představovat může.

6 Odesílání e-mailů pomocí RabbitMQ

Ve chvíli, kdy už je aplikace pro rozesílku pomocí databázové fronty napsána, je úprava, aby bylo možné rozesílat e-maily pomocí RabbitMQ, poměrně snadná. Ze závislostí na jiné knihovny stačí přidat v composeru jedinou - olicek/rabbitmq. Tato knihovna zajišťuje připojení k RabbitMQ. Jedná se tedy jen o lehkou vrstvu nad php-amqplib knihovnou.

Celá logika knihovny zůstává stejná a mění se pouze 2 věci. Vložení dat do fronty a čtení dat z fronty.

6.1 Vložení dat do fronty

Protože se do fronty vkládá pouze ID entity Mail, tak proces odeslání do fronty je velice podobný. E-mail se nejprve uloží do databáze a poté do RabbitMQ fronty. Je pro to připravena třída *QueueRabbitMqMailSender*, která má o jednu závislost více než třída QueueMailSender, která používá databázovou frontu. Tato závislost spočívá ve třídě *ConnectionFactory*, jež vrací připojení k RabbitMQ. Celé vložení do RabbitMQ fronty se provede následujícím kódem:

Nejprve se vytvoří připojení a z toho připojení se vytáhne kanál, jehož prostřednictvím se připojuje aplikace k RabbitMQ serveru.

```
$connection = $this->connectionRabbitMQFactory

      ->getConnection('default');

if($this->channel === null) {

      $this->channel = $connection->getChannel();

}
```

Jak bylo popsáno v předchozí kapitole, Publisher (což je v tomto případě třída *QueueRabbitMqMailSender*) by měl definovat Exchange a frontu, pro kterou bude zprávu posílat.

Toto vytvoří (pokud již neexistuje) Exchange s názvem Mails a nastaví ho na typ přímý (direct).

```
$this->channel->exchange_declare('mails', 'direct');
```

Další řádek vytvoří frontu (pokud již neexistuje) jménem newsletters a nastaví ji tak, aby přežila restart RabbitMQ.

```
$this->channel->queue_declare('newsletters', false, true, false,
false);
```

Posledním nastavením je propojení exchange a fronty. Exchange je deklarované jako direct, takže očekává routovací klíč, podle kterého bude rozdělovat zprávy do front. Pro toto propojení exchange-fronta je routovacím klíčem slovo **newsletter**.

```
$this->channel->queue_bind('newsletters', 'mails', 'newsletter');
```

Nyní již existuje připojení k RabbitMQ a existuje vytvořený exchange a fronta, do které se zpráva pošle. Chybí poslední krok, a sice, vytvoření a odeslání zprávy. Zpráva je uložena jako první argument třídy AMQPMessage. Druhým parametrem je pole nastavení. Jedná se o takzvaná metadata zprávy. Mohly by zde být například routovací hlavičky pro exchange. Já zde mám uveden vedle content_type pouze příznak, že zpráva se má uložit na disk, a tudíž přežít restart RabbitMQ serveru.

```
$msg = new AMQPMessage((int) $this->connection->lastInsertId(), [

      'delivery_mode' => AMQPMessage::DELIVERY_MODE_PERSISTENT,

      'content_type' => 'text/plain',

]);
```

Závěrem je odeslání. Na předchozích řádcích bylo vytvořeno připojení do RabbitMQ, vytvoření exchange, fronty a zprávy. Nyní se pouze vloží zpráva do exchange **mails** s routovacím klíčem **newsletter**.

```
$this->channel->basic_publish($msg, 'mails', 'newsletter');
```

6.2 Čtení z fronty

Čtení z fronty přináší poměrně hodně věcí. Tou nejpřednější a pro programátora příjemnou je fakt, že se nemusí starat o získávání dat z fronty a o jejich mazání. O to se postará RabbitMQ na základě odpovědi, kterou consumer pošle.

Připojení a získání kanálu je stejné jako jo u publishera. Consumer se připojuje pouze ke své frontě, proto se zde nedeklaruje exchange ani propojení exchange a fronty. Vytváří se zde pouze fronta (pokud již neexistuje). Pokud by existovala, ale měla jiné nastavení, tak se nepovede připojit k RabbitMQ serveru a připojení vyhodí výjimku. Deklarace fronty tedy vždy musí být stejná, jako již existující.

```
$channel = $rabbitConnection->getChannel();

$channel->queue_declare('newsletters', false, true, false, false);
```

Čtení zpráv z fronty je realizováno pomocí callbacku. Callback je v PHP nejčastěji anonymní funkce, která může dostat na vstupu data. Tato data jí jsou předána v místě volání funkce. Pro RabbitMQ jsou vstupní data callbacku AMQPMessage. Je to tedy stejný objekt, který byl vytvořen v publiseru. Celý kód callbacku může vypadat následovně:

```
$callback = function(\PhpAmqpLib\Message\AMQPMessage $msg) use
($mailRepository, $mailSender, $em) {

      Try {

            $mail = $mailRepository

                  ->fetchMailById((int) $msg->getBody());

            $mailSender->send($mail);

            $msg->delivery_info['channel']

                  ->basic_ack($msg->delivery_info['delivery_tag']);

      }

      catch(\Throwable $e)  {

            $msg->delivery_info['channel']

                  ->basic_nack($msg->delivery_info['delivery_tag']);

      }

};
```

V kódu výše se nejdříve z databáze načte entita Mail na základě ID, které bylo doručeno zprávou. Poté se takto vytvořená entita předá odesilateli, což může být například PHPMailSender, který byl představen v kapitole 5. Posledním bodem procesu zpracování zprávy je potvrzení, respektive její zamítnutí. Na základě toho je zpráva smazána z fronty nebo označena za mrtvou, může být zahozena nebo zpracována Death letter exchange.

Takto vytvořený callback se potom musí zavolat při konzumaci fronty. Ještě před tím je možné nastavit, kolik zpráv se má alokovat pro tohoto konkrétního consumera. V mém případě to je 20 zpráv.

```
$channel->basic_qos(null, 20, null);

$channel->basic_consume('newsletters', 'newsletter' . getmypid(),
false, false, false, false, $callback);
```

Skripty, které zpracovávají frontu RabbitMQ, jsou z pravidla dlouho běžící. Může se stát, že si budou pro sebe neustále alokovat paměť. Může i dojít k jinému problému a consumer přestane plnit funkci, kterou plnit má. V takovém případě je dobré mít nástroj, kterým se consumer restartuje. Pro tento případ jsem použil velikost RAM paměti a to 15MB. Jakmile si skript pro sebe alokuje 15MB paměti, ukončí se. Je dobré, pokud na pozadí běží nějaký daemon (jako supervisord nebo systemd), který je schopen ukončené procesy opět nastartovat. Tím se dosáhne nekonečného běžícího procesu.

```
while(\count($channel->callbacks) && memory_get_usage() < 15000000)
{

      $channel->wait();

}
```

Kód výše říká, že skript má počkat, jestliže je vytvořen nějaký callback a zároveň velikost alokované paměti není větší než 15MB.

A to je vše. Toto jsou jediné zásahy, které je nutno provést, aby se namísto databázové fronty začala používat RabbitMQ fronta.

7 Dopad na výkon

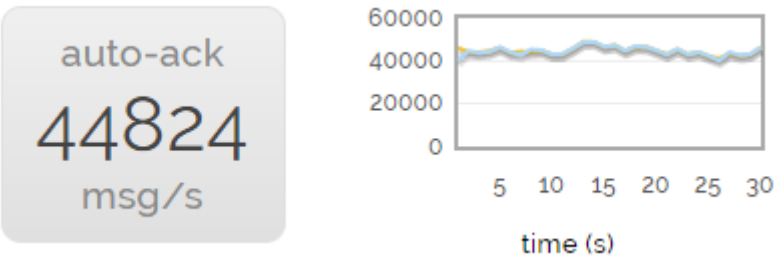
Jedním z hlavních cílů práce je změřit dopad na výkon serveru a rychlost odesílání e-mailů. V této kapitole se tedy budu věnovat rychlosti RabbitMQ obecně a v praxi vyzkoušené a změřené rychlosti rozesílky newsletterů a zátěže pro server. Testování rychlosti rozesílky a zatížení hardware probíhalo po dobu jednoho měsíce a celkem bylo odesláno a změřeno přes 10 milionů e-mailů.

7.1 Rychlost RabbitMQ serveru

Rychlost samotného RabbitMQ je měřena pomocí jednoho Publishera a jednoho consumera. Publisher publikuje pouze pár bitů velkou zprávu a consumer nemá kolem zpracování zprávy postavenou žádnou logiku. Jde čistě jen o propustnost RabbitMQ serveru.

7.1.1 Publikování a konzumování naráz

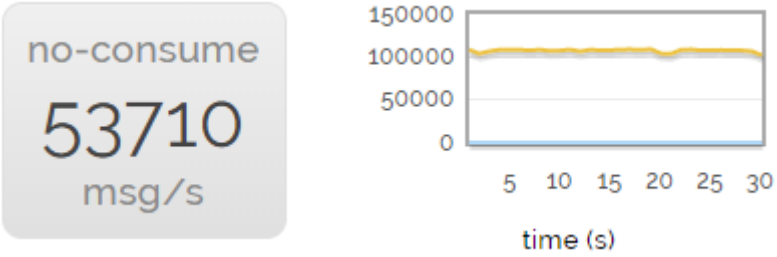
První graf uvádí, kolik je možné poslat zpráv přes RabbitMQ za sekundu. Publisher publikuje, consumer ihned zpracovává.



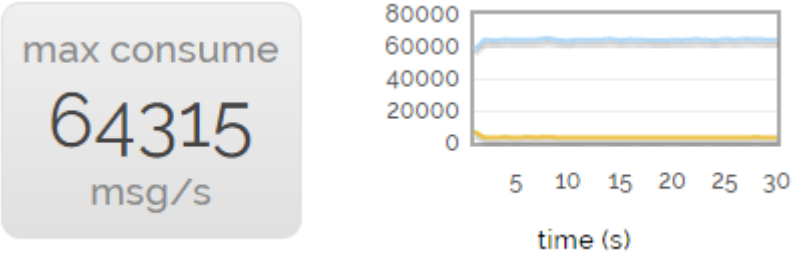
Obrázek 18 Rychlost - publikování a konzumování naráz [9]

7.1.2 Publikování a konzumování zvlášť

Pokud by existoval pouze Publisher, tak se zpracování zrychlí o cca 16%. Pokud má server dostatečně velkou paměť anebo zpráv není velké množství a je nutné je co nejrychleji vygenerovat nebo odbavit, tak se může použít právě tento princip: že se data nejdříve vygenerují a v momentě plné fronty se spustí consumer, který je rychleji odbaví.



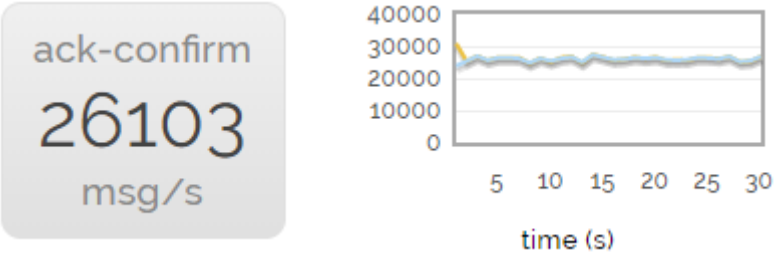
Obrázek 19 Rychlost - pouze publikování



Obrázek 20 Rychlost - pouze konzumování [9]

7.1.3 Publikování a konzumování naráz + potvrzení přijetí a zpracování

Jedná se o nejnáročnější operaci, která neukládá zprávu na disk. Zahrnuje potvrzení vložení zprávy do fronty a potvrzení zpracování consumerem. Rychlost byla snížena o více než 40% oproti situaci, kdy se žádná zpráva nepotvrzovala.

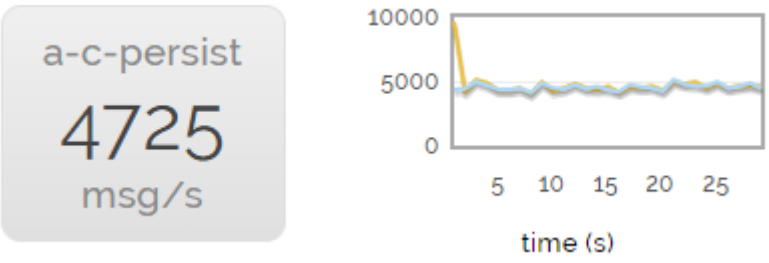


Obrázek 21 Rychlost - publikování a konzumování naráz + potvrzování [9]

7.1.4 Publikování a konzumování naráz + potvrzování + persistence

Poslední a nejnáročnější operace je totožná jako předchozí s tím rozdílem, že se nyní každá zpráva uloží na disk. Pokud dojde k pádu RabbitMQ serveru, tak zpráva zůstane ve frontě uložena a jakmile se RabbitMQ opět nastartuje a připojí se alespoň jeden consumer, tak mohou být zprávy opět odbavovány.

Tento nejnáročnější úkol pro RabbitMQ je o 90% pomalejší než bez potvrzování a ukládání na disk.



Obrázek 22 Rychlost - Publikování a konzumování naráz + potvrzování + persistence [9]

7.2 Rychlost rozesílky

Porovnávány jsou 4 způsoby rozesílky. A sice jeden SMTP server, tři SMTP servery, nová knihovna přes 3 SMTP servery, RabbitMQ přes 3 SMTP servery.

7.2.1 Jeden SMTP server

Jedná se o původní řešení, které bylo nahrazeno v roce 2017. Rozesílka byla velice pomalá a kvůli vytížení hlavní databáze bylo dovoleno generovat newslettery pouze v nočních hodinách. Kvůli pomalosti rozesílky se objevila potřeba řešit rozesílání e-mailů jiným způsobem. I navzdory nočnímu generování e-mailů bylo rozesílání velice pomalé. Hlavním důvodem byl pouze jeden SMTP server, na který jsme mohli e-maily posílat, a doba běhu jednoho skriptu.

Jeden skript se spouštěl každých 20 sekund a odbavil 200 e-mailů. Do tohoto počtu se počítaly i transakční e-maily (objednávky, rezervace, atp.). Výsledkem bylo odeslání pouhých 9 e-mailů na SMTP server za sekundu.

Tabulka 4 Rychlost rozesílky newslettrů - Jeden SMTP server

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
256 038	97,17	101,27	9,22	9,22	9,35

7.2.2 Tři SMTP servery

Někdy ke konci listopadu jsme ve společnosti dostali od poskytovatele možnost nahrávat na 3 SMTP servery naráz. Databáze, infrastruktura, vše zůstalo stejné, jen namísto jednoho jsme mohli užívat 3 SMTP servery. Na servery se posílalo pro každý SMTP po 20 sekundách, takže se během minuty skript spustil 9x (místo předchozích 3x). Podle očekávání se rozesílka zrychlila přibližně 3x.

Tabulka 5 Rychlost rozesílky newsletterů - Tři SMTP servery

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
2 811 764	118,16	142,98	30,02	29,96	31,03

7.2.3 Nová knihovna pomocí databázové fronty

Pro tuto rozesílku se změnilo téměř všechno. Dříve se newslettery rozesílali výhradně z hlavního serveru, na kterém běží i nejdůležitější aplikace. Přistupovalo se do jedné databáze, ze které se generovaly e-maily, jež se tam také ukládali. Nyní je e-mailový server (generování + odesílání) umístěn na jiném serveru. Přistupuje k jiné databázi (na stejném serveru) a newslettery generuje z replikační databáze hlavního serveru (takže neblokuje ani hlavní databázi). Díky tomu se mohlo začít rozesílat newslettery i přes den.

Výsledky rozesílky newsletterů řešených pomocí databázové fronty jsem rozdělil na dvě skupiny.

7.2.3.1 Problém s pamětí

První skupina ukazuje generování e-mailů, když ještě nebyl vyřešen problém s pamětí, který je popsán v kapitole 4.5.1.

Tabulka 6 Rychlost rozesílky newsletterů - Nová knihovna pomocí databázové fronty (problém s RAM)

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
605 663	41,52	46,66	17,27	20,00	20,03

Jak je vidět z tabulky výše, tak neuvolňování si paměti vedlo k velkému zpomalení celé rozesílky. Je to proto, že čas je brán z průměru a na začátku bylo generování i odesílání rychlé. Jak však pomalu docházela paměť, server musel začít paměť uvolňovat, čímž rozsáhle přistupoval na disk a odkládal si tam záznamy, které zrovna nepotřeboval.

7.2.3.2 Vyřešený problém s pamětí

Tabulka 7 Rychlost rozesílky newsletterů - Nová knihovna pomocí databázové fronty

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
1 581 622	154,52	274,39	57,98	60,25	61,57

Tabulka demonstruje poměrně velké **navýšení rychlosti** oproti verzi 7.2.2. To je způsobeno několika **faktory**.

1) Čte se z jedné databáze, ke které téměř nikdo nepřistupuje (replikační databáze)
2) Zapisuje se do jiné databáze, do které je mnohonásobně menší přístup než do hlavní databáze
3) Spouštění skriptu mohlo být zrychleno a místo jednou za 20 sekund pro každý SMTP jsem mohl skript spouštět jednou za 10 sekund. Doba provedení jednoho skriptu se pohybovala okolo 8 sekund.

7.2.4 Nová knihovna pomocí RabbitMQ

Konečně poslední měřený způsob odesílání newsletterů byl pomocí RabbitMQ serveru. Server běžel na stejném serveru jako celý e-mailový systém a databáze.

Provedl jsem měření pro 3 různá nastavení s tím, že nejvíce newsletterů se odeslalo pomocí posledního nastavení.

7.2.4.1 12 consumerů (prefetch 1)

Tabulka 8 Rychlost rozesílky newsletterů - Nová knihovna pomocí RabbitMQ (12 consumerů, prefetch 1)

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
945 138	116,23	152,2987654	85,70	88,12	117,55

Jedná se o nastavení, které bylo zcela nejrychlejší ze všech co do odesílání newsletterů na SMTP servery. Protože však běžel skript 12x naráz, tak jsem zde narážel na výkonnostní limity serveru, proto se od tohoto nastavení upustilo.

7.2.4.2 6 consumerů (prefetch 1)

Toto nastavení bylo „přechodové“. Chtěl jsem změřit rozdíl mezi prefetch 1 a prefetch 20, které je podle dokumentace nejefektivnější. Zatížení serveru kleslo téměř o polovinu, ale rychlost rozesílání se taktéž snížila.

Tabulka 9 Rychlost rozesílky newsletterů - Nová knihovna pomocí RabbitMQ (6 consumerů, prefetch 1)

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
877 310	113,68	117,7684211	73,66	73,66	77,40

7.2.4.3 6 consumerů (prefetch 20)

Toto je mé finální nastavení, na kterém nyní finálně funguje rozesílka newsletterů ve firmě. Je to kompromis mezi rychlostí rozesílky a zatížením serveru.

Tabulka 10 Rychlost rozesílky newsletterů - Nová knihovna pomocí RabbitMQ (6 consumerů, prefetch 20)

Počet e-mailů	Průměr generování mail/sec	Nejrychlejší generování mail/sec	Průměr odeslaných mail/sec	Medián odeslaných mail/sec	Nejrychlejší rozesílka mail/sec
3 119 755	128,20	176,52	86,13	87,36	105,91

7.3 Zatížení serveru

Protože první dva typy měření, bez nové knihovny, byly spouštěny na jiném serveru s jinou konfigurací, nemohu zde ukázat graf vytížení serveru (více spouštěných skriptů, větší zátěž a nejasnost, zda by se do grafu nepromítly i jiné služby běžící na pozadí).

Server, na kterém probíhal test, má následující parametry:

- CPU: Intel Xeon E312xx 2500 MHz (4 jádra)
- RAM: 8 GB (v průběhu testování i kvůli jiným službám došlo k navýšení na 12 GB)
- Disk: 20 GB SSD (Solid-state drive)

7.3.1 Legenda ke grafům

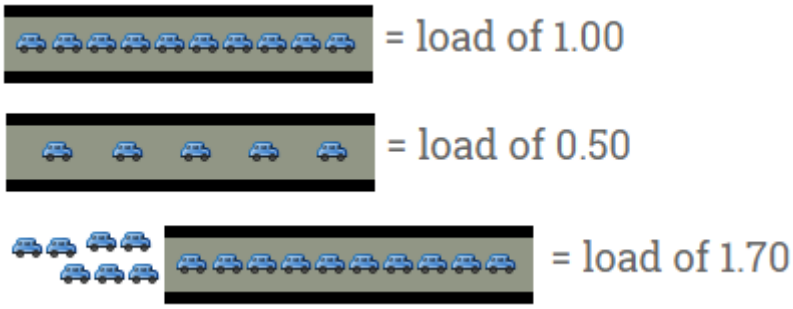
Na tomto místě bych chtěl objasnit, jak číst grafy, na kterých bude demonstrováno zatížení serveru. Některé grafy mají agregované hodnoty po půl hodině. Je to kvůli úspoře výpočetního výkonu. Grafy jsou generovány pomocí nástroje Grafana a tento nástroj dovoluje zobrazit grafy například za dobu

2 let. To by nebylo možné, pokud by si neustále držel veškerá data po 5 sekundách. I když budou některá data agregovaná, tak by měla mít dobrou vypovídací hodnotu o zatížení serveru.

7.3.1.1 CPU (Central processing unit) load

CPU udává vytížení jader procesoru. Podstatné je, že pokud je CPU load nižší nebo stejný než má procesor jader, tak zpracování probíhá plynule. Pokud CPU load stoupne nad počet procesorů, tak to znamená, že procesor brzdí zpracování. Je takzvaně přetížen. V grafu je tato hodnota prezentována vyplněnou zelenou plochou.

Pro jedno jádrový procesor to lze přehledně vysvětlit následujícím obrázkem.



Obrázek 23 Jak funguje CPU load? [12]

7.3.1.2 Swap

Jestliže dochází serveru paměť a může si nějakou uvolnit, tak si ji zapíše právě do odkládacího prostoru paměti – swapu. Pokud je do swapu zapisováno více, tak to značí problém s nedostatkem paměti a zpomaluje to běh serveru. Problém je řešitelný snadno - pořízením přídatné paměti.

7.3.1.3 Paměť (RAM)

Do paměti si PHP načte vše, co potřebuje ke zpracování skriptu. O uvolňování se stará garbage collector. V grafu se vyskytují **3 spojnice**:

- Mem_free: udává, kolik je volné paměti
- Mem_used: udává kolik je reálně využité paměti
- Mem_cached a mem_bufferd: jedná se o alokovanou paměť serverem. Ukládají se zde soubory, indexy, aj. pro případ potřeby. Děje se to kvůli zrychlení systému. A protože je to možné; jinak by paměť zůstala nevyužita. Jakmile je však vyžadována jakýmkoli programem, tak se dokáže velice rychle uvolnit.

7.3.1.4 SQL dotazy

Tento graf zobrazuje počet SQL dotazů do MySQL databáze. V databázi nejsou pouze e-maily, ale další schémata. Graf nicméně zobrazuje databázi jako celek, tedy všechna schémata dohromady. Opět jsou zde 3 spojnice. Nevím přesný důvod, ale Grafana zobrazuje pokaždé jinou barvou jednotlivé příkazy. Je tedy nutné se řídit legendou přímo v obrázku, kde je popsáno, co která spojnice znamená.

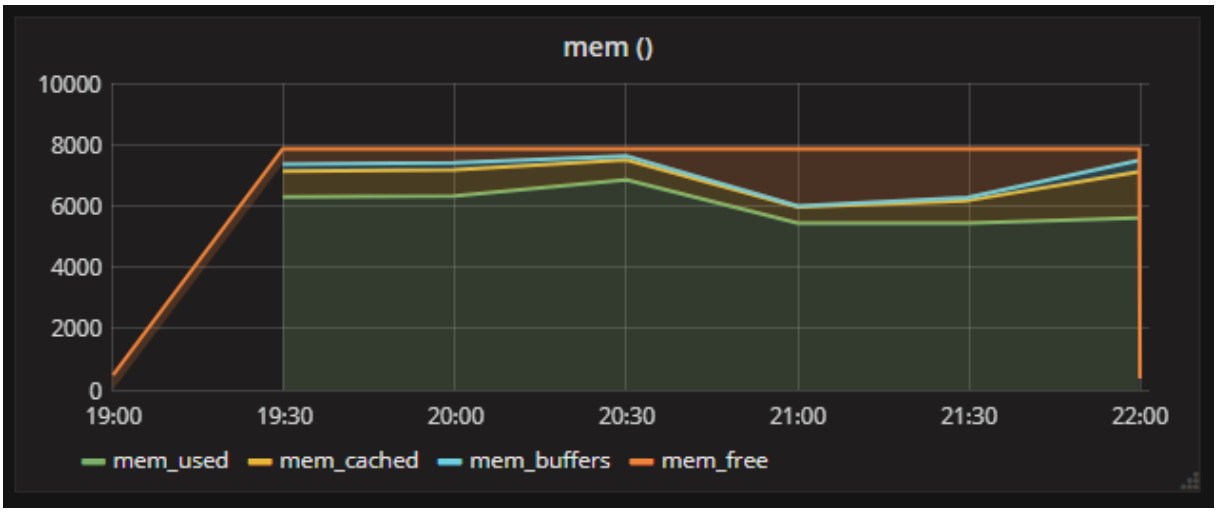
7.3.2 Databázová fronta

Databázová fronta bude opět rozdělena jako v předchozí kapitole na dvě části (problém s neuvolňováním paměti a bez problému).

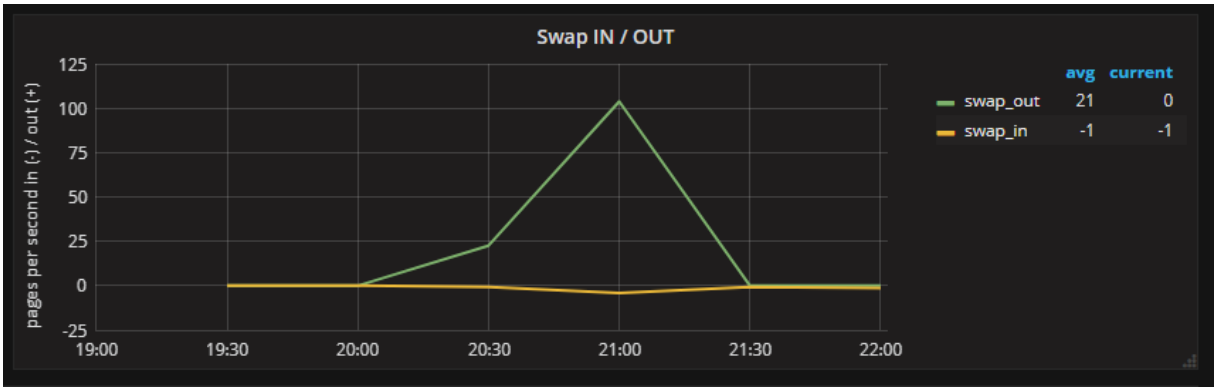
7.3.2.1 Problém s pamětí

Grafy jsou agregované po půl hodině. Proto tam není vidět přesně, jak došla paměť. Je to jen průměr za půl hodiny. Nicméně co vidět můžeme je, že ve 21:00 došlo ke značnému uvolnění paměti a to ve chvíli, kdy se vygenerovala fronta.

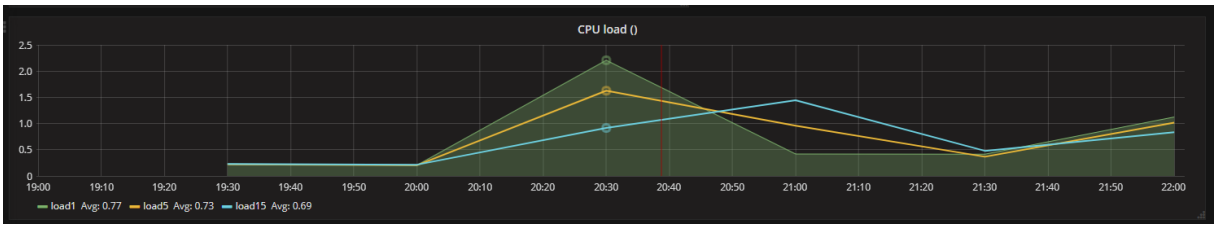
Dále je patrno, jak se paměť neustále uvolňovala, protože docházela a server tedy neustále odkládal data do swapu. Toto odkládání reflektuje také graf zápisu/čtení z disku. Z grafu CPU load lze vyčíst, že se po dobu generování e-mailů zatížila plně 2 jádra, takže ještě byla značná výkonnostní rezerva. Odesílání potom generovalo minimální zátěž ve formě 0,5CPU load.



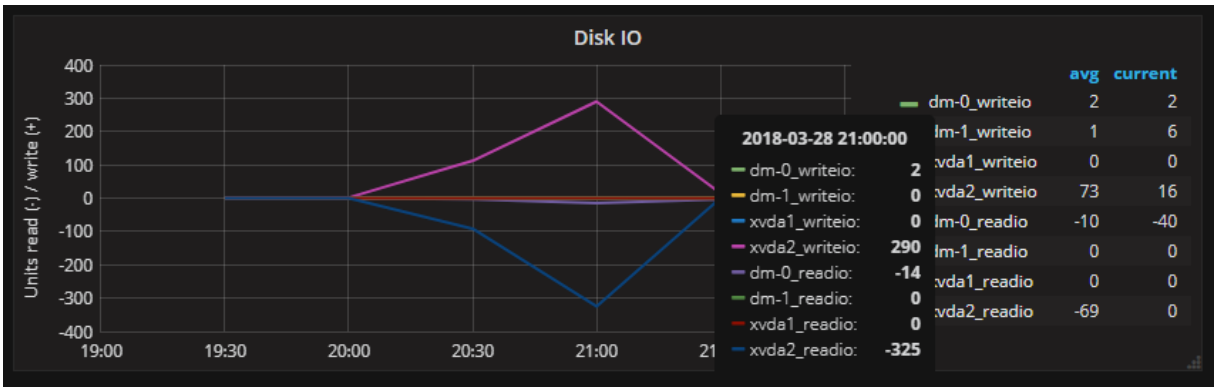
Obrázek 24 Problém s pamětí - RAM



Obrázek 25 Problém s pamětí - odkládací prostor RAM



Obrázek 26 Problém s pamětí - CPU load



Obrázek 27 Problém s pamětí - zápis/čtení disku

7.3.2.2 Vyřešený problém s pamětí

Poté, co se mi podařilo vyřešit problém s neuvolňováním paměti, začaly vypadat grafy zátěže mnohem lépe.

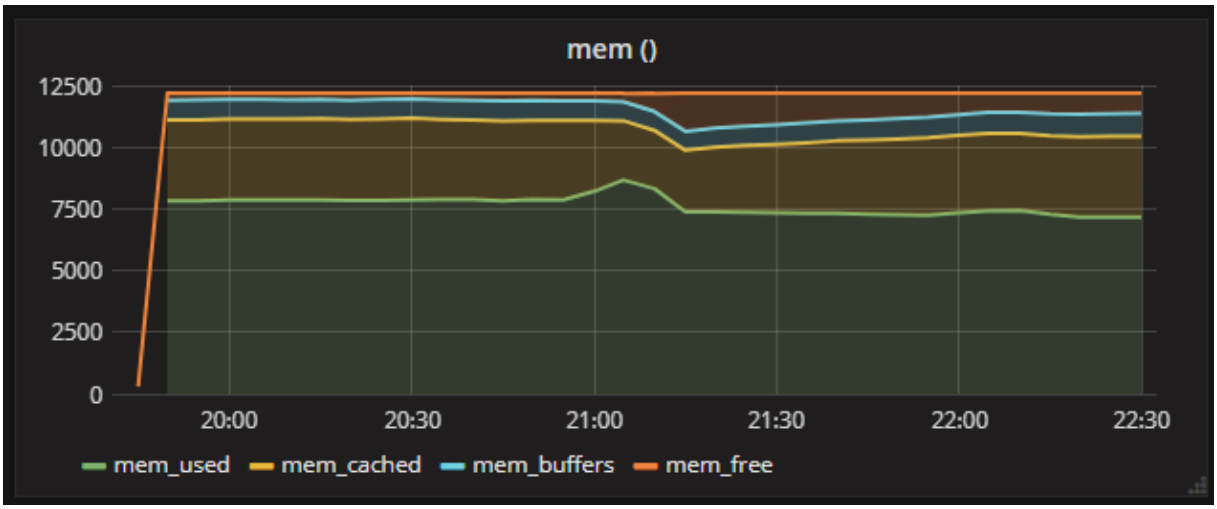
7.3.3 RabbitMQ

Opět je to rozděleno do 2 kategorií podle nastavení RabbitMQ. 12 consumerů (prefetch1) a finální řešení 6 consumerů (prefetch20).

7.3.3.1 12 consumerů (prefetch 1)

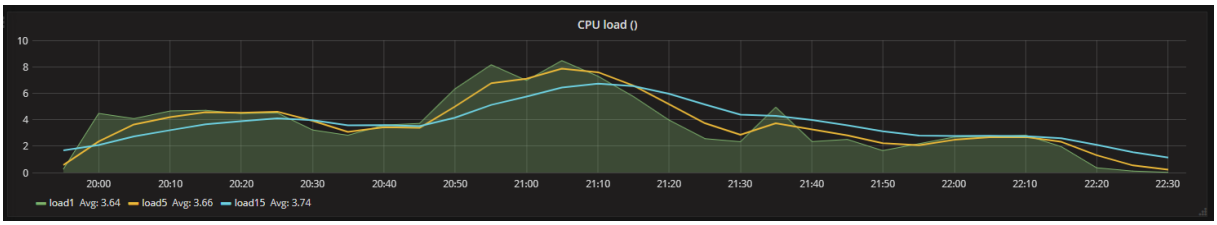
Jednalo se o největší zatížení CPU ve sledovaném období. Bohužel se do výsledných grafů dostal také jiný skript. Ten vytížil paměť a pokládal velké množství čtecích dotazů do databáze. Tento skript se spustil přesně ve 21:00 a trval 15 minut. Grafy níže zobrazují rozesílku přes 750 tisíc e-mailů.

Jak je vidět z grafu, paměť je alokována konstantně. Pouze po 21. hodině se mírně zvýšila v důsledku spuštění konkurenčního skriptu. Zpracování má konstantní alokování paměti díky tomu, že jakmile consumer dosáhne 15MB alokované paměti, tak se vypne a supervisord jej znovu okamžitě zapne.



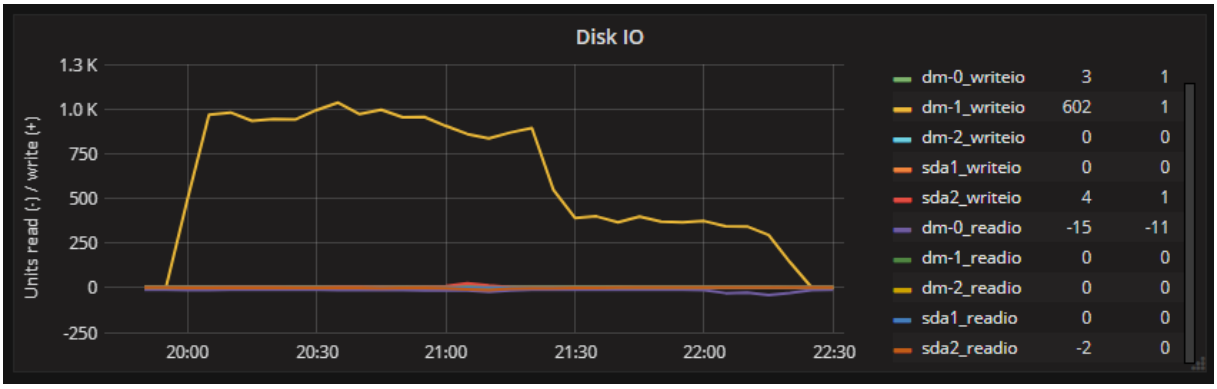
Obrázek 28 Dvanáct consumerů (prefetch 1) - paměť

Ještě před spuštěním konkurenčního skriptu je z grafu patrný nárůst zátěže na CPU. Je to kvůli režii se zpracováváním fronty. Fronta se generovala rychleji, než byla odbavována a server tak měl čím dál větší práci ji obsloužit. CPU load překročil i hranici 8, která by odpovídala 8 jádrům. My jsme měli na serveru k dispozici pouhá 4 jádra a celý server byl tedy značně přetížen.

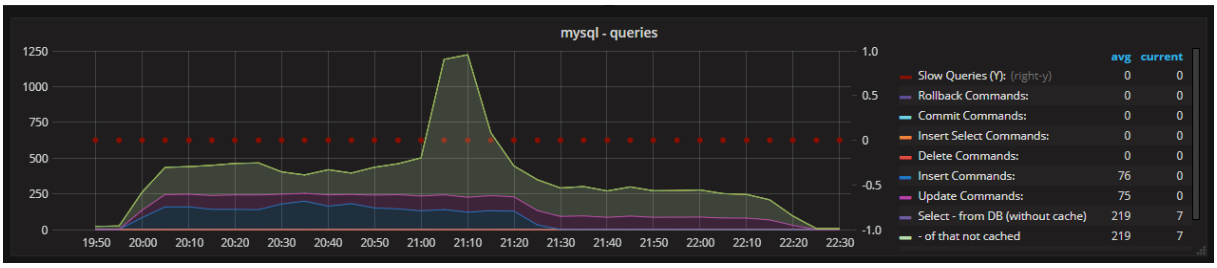


Obrázek 29 Dvanáct consumerů (prefetch 1) – CPU load

Na následujícím grafu je hezky vidět, že ve 21:25 byla kompletně vygenerována fronta a už se pouze odesílaly e-maily. V zápise na disk se to projeví proto, že zprávy jsou označeny jako durable, takže přežijí pád aplikace. Tuto odolnost zajišťuje právě zápis na pevný disk.



Obrázek 30 Dvanáct consumerů (prefetch 1) - zápis/čtení disku

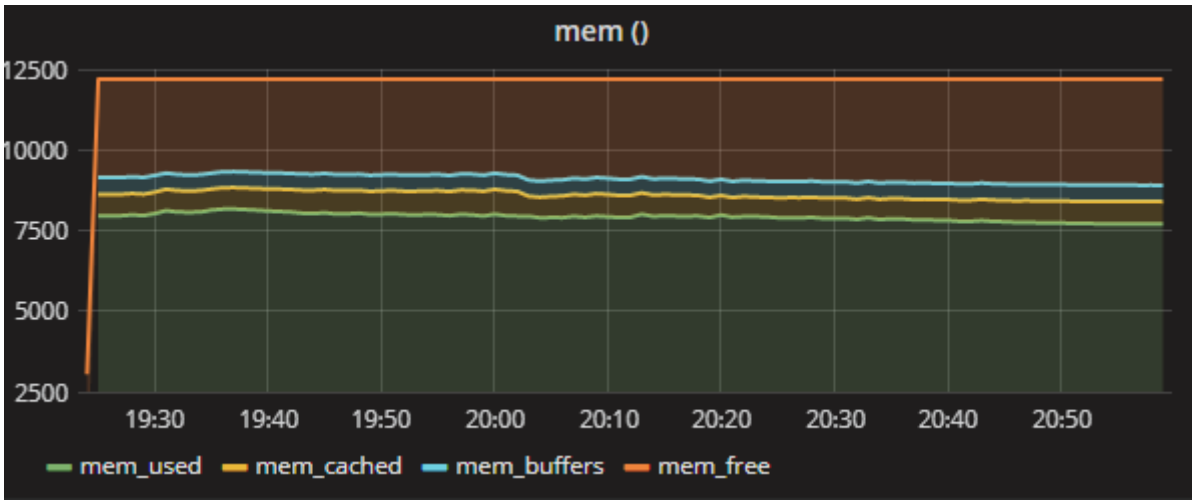


Obrázek 31 Dvanáct consumerů (prefetch 1) - MySQL

7.3.3.2 6 consumerů (prefetch 20)

Tato konfigurace je nejvýkonnější z hlediska rychlosti odeslání e-mailů do SMTP serveru za optimálního využití serveru. Grafy níže zobrazují rozesílku téměř 500 tisíců e-mailů.

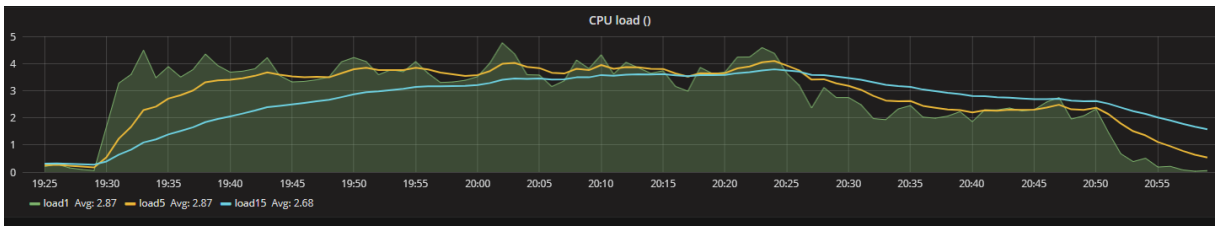
Paměť je alokována opět v podstatě konstantně. Na tomto grafu nejsou agregovaná data a je možno vidět, jak se paměť neustále uvolňuje, jakmile consumer dosáhne 15MB alokované paměti.



Obrázek 32 Šest consumerů (prefetch 20) - paměť

CPU load se drží okolo 4, což je na čtyř jádrovém procesoru 100% vytížení. Problém by nastal, pokud by bylo potřeba spouštět další (byť jen polovičně náročnou) službu ve stejnou dobu. Pak by byl server jednoznačně přetížen.

7.3.3.3



Obrázek 33 Šest consumerů (prefetch 20)

Graf čtení a zápisu na disk zde neuvádím záměrně; je téměř totožný jako předchozí.

8 Závěr

V úvodu bylo deklarováno několik cílů.

- Hlavním cílem práce bylo vytvořit knihovnu, která rozesílá e-maily. Tato knihovna je nyní napsána tak, aby byla do budoucnosti rozšiřitelná jakýmkoli způsobem. Tím je myšleno, že programátor, který knihovnu použije, může použít vlastní entity i vlastní systém pro ukládání a načítání e-mailů. V knihovně jsou připraveny nástroje jak ukládat a načítat data z databázové fronty. – Splněno a popsáno v kapitole 2 a 4.
- Druhým cílem bylo popsat fungování technologie pro správu front RabbitMQ a začlenit ji do již vytvořené knihovny pro rozesílání e-mailů. – Splněno a popsáno v kapitole 5 a 6.
- Dalším cílem bylo porovnat zátěž na server a rychlost rozesílky realizovanou pomocí databázové fronty a pomocí RabbitMQ. – Splněno a popsáno v kapitole 7.

Vytvořená knihovna je dostupná na portále GitHub, což je celosvětově nejrozšířenější úložiště veřejných zdrojových kódů. Konkrétně na adrese: https://github.com/mailsender/core. Verze distribuovaná pro potřeby diplomové práce je v0.8.

Konkrétní implementace za využití knihovny Doctrine je umístěna na stejném portále, na adrese: https://github.com/mailsender/doctrine-mail-sender. Taktéž ve verzi v0.8.

Knihovna jako taková je snadno rozšiřitelná. Dalším vývojem je snadno možné implementovat adaptéry do dalších knihoven jako je např. Nette Database, Nextras ORM, DIBI, … Implementace RabbitMQ a databázové fronty je součástí knihovny jako ukázka. Pro základní použití je to podle mého názoru naprosto dostatečné.

Pro lepší výkon by mohlo být zajímavé odesílání newsletterů při úplném odstřihnutí od databáze a předávání si celé entity e-mailu ve zprávě posílané do RabbitMQ. Musela by potom pravděpodobně existovat druhá fronta, která by e-mail uložila do databáze, aby byl dostupný on-line. Pokud by všechno prováděla jedna fronta, k žádné úspoře by nedošlo.

O fungování fronty pomocí databáze jsem měl poměrně dobré znalosti. RabbitMQ jsem znal před psaním diplomové práce jen velice povrchně. Tato technologie mě ve výsledku moc nepřekvapila; už od začátku jsem na ni měl velké nároky. Jediné, o co jsem měl obavu, byla ztráta dat (zprávy). To se ale nakonec ukázalo jako lichá obava, protože RabbitMQ má tuto problematiku vyřešenou velmi dobře. Ke ztrátě dat by tedy nemělo dojít.

Na základě mé zkušenosti s RabbitMQ máme nyní ve společnosti velké plány s jejím dalším využitím. Zejména pro náročné operace, které se doposud provádí přes noc s velkým počtem dat. Typickým příkladem je aktualizace produktů do ElastSearch.

V neposlední řadě bylo důležitým bodem mé diplomové práce měření zátěže serveru a rychlosti rozesílky. Zde se pro mě konala největší překvapení. Prvním překvapením bylo zjištění, jak velice pomalu fungovalo původní řešení. Nikdy předtím jsem ho totiž nezkoumal tak podrobně.

Druhým překvapením bylo, že oproti databázové frontě není RabbitMQ o tolik rychlejší. To ale může být dáno tím, že se s databází pracuje stejně. Jak jsem psal výše, je to jedna z možností, jak stávající knihovnu rozšířit, tj. aby nepracovala s databází vůbec (a do databáze se vkládalo případně pomocí jiné fronty).

Třetím překvapením byla skutečnost, že zpracování/generování e-mailů může být tak náročné. Nevím přesný důvod. Podle analytiky poskytovatele vše vytěžuje zejména PHP 7.1, proto se pravděpodobně jedná o běžný jev a zpracování tolika záznamů v tak krátkém čase server jednoduše vytíží.

Výsledkem přepsání rozesílání newsletterů pomocí RabbitMQ je konečné zrychlení rozesílky cca 10x. Další možností ještě většího zrychlení by momentálně představovalo rozdělení rozesílky a generování. A to jak na naší straně, tak na straně poskytovatele. Již nyní s poskytovatelem jednáme o možnosti nahrát do SMTP serveru e-maily, které by se odeslaly později. Podle odhadů by se tak rozesílka měla zrychlit ještě o cca jednu třetinu.

Použitá literatura

[1] J. Klensin, „Simple Mail Transfer Protocol,“ 2008. [Online]. Available: https://tools.ietf.org/html/rfc5321#section-4.5.3.2. [Přístup získán 20 4 2018].

[2] J. Tarvainen, „Symfony Finland,“ 28 12 2015. [Online]. Available: https://www.symfony.fi/entry/symfony-benchmarks-php-56-hhvm-and-php-7. [Přístup získán 20 4 2018].

[3] „Wikipedia,“ Wikimedia Foundation, Inc., 12 4 2018. [Online]. Available: https://en.wikipedia.org/wiki/RabbitMQ. [Přístup získán 20 4 2018].

[4] „Message broker,“ Wikimedia Foundation, Inc., 23 12 2017. [Online]. Available: https://en.wikipedia.org/wiki/Message_broker. [Přístup získán 20 4 2018].

[5] „rabbitmq,“ Pivotal, [Online]. Available: https://www.rabbitmq.com/img/tutorials/intro/hello-world-example-routing.png. [Přístup získán 20 4 2018].

[6] Pivotal, „AMQP 0-9-1 Model Explained,“ Pivotal, [Online]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts.html. [Přístup získán 20 4 2018].

[7] Pivotal, „Dead Letter Exchanges,“ Pivotal, [Online]. Available: https://www.rabbitmq.com/dlx.html. [Přístup získán 20 4 2018].

[8] Pivotal, „Queues,“ Pivotal, [Online]. Available: https://www.rabbitmq.com/queues.html. [Přístup získán 20 4 2018].

[9] Pivotal, „RabbitMQ Performance Measurements, part 2,“ Pivotal, 25 4 2012. [Online]. Available: https://www.rabbitmq.com/blog/2012/04/25/rabbitmq-performance-measurements-part-2/. [Přístup získán 20 4 2018].

[10] Pivotal, „Clients & Developer Tools,“ Pivotal, [Online]. Available: http://www.rabbitmq.com/devtools.html. [Přístup získán 20 4 2018].

[11] „GitHub,“ GitHub, Inc., [Online]. Available: https://github.com/search?l=PHP&q=rabbitmq&type=Repositories. [Přístup získán 22 4 2018].

[12] Andre, „Understanding Linux CPU Load - when should you be worried?,“ Scout, [Online]. Available: http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages. [Přístup získán 20 4 2018].

[13] „Simple Mail Transfer Protocol,“ Wikimedia Foundation, Inc., 9 4 2018. [Online]. Available: https://cs.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol. [Přístup získán 20 4 2018].

[14] Oracle, „Commercial License for OEMs, ISVs and VARs,“ Oracle, 7 2010. [Online]. Available: https://www.mysql.com/about/legal/licensing/oem/. [Přístup získán 20 4 2018].

[15] „Entita (informatika),“ Wikimedia Foundation, Inc., 17 2 2017. [Online]. Available: https://cs.wikipedia.org/wiki/Entita_(informatika). [Přístup získán 20 4 2018].

[16] „PHP,“ Wikimedia Foundation, Inc., 20 4 2018. [Online]. Available: https://en.wikipedia.org/wiki/PHP. [Přístup získán 20 4 2018].

[17] „API,“ Wikimedia Foundation, Inc., 4 10 2017. [Online]. Available: https://cs.wikipedia.org/wiki/API. [Přístup získán 20 4 2018].

[18] „Streaming Text Oriented Messaging Protocol,“ Wikimedia Foundation, Inc, 1 11 2017. [Online]. Available: https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol. [Přístup získán 20 4 2018].

[19] „MQTT,“ Wikimedia Foundation, Inc, 12 4 2018. [Online]. Available: https://en.wikipedia.org/wiki/MQTT. [Přístup získán 20 4 2018].

[20] „Remote procedure call,“ Wikimedia Foundation, Inc, 5 4 2017. [Online]. Available: https://cs.wikipedia.org/wiki/Remote_procedure_call. [Přístup získán 20 4 2018].

[21] „Transmission Control Protocol,“ Wikimedia Foundation, Inc, 4 10 2017. [Online]. Available: https://cs.wikipedia.org/wiki/Transmission_Control_Protocol. [Přístup získán 20 4 2018].

[22] „Elasticsearch,“ Wikimedia Foundation, Inc, 1 3 2018. [Online]. Available: https://cs.wikipedia.org/wiki/Elasticsearch. [Přístup získán 20 4 2018].

A Přílohy

A.1 Slovníček pojmů

A.1.1 View (pohled)

Databázová konstrukce view slouží ke spojení dat z více tabulek do jedné tabulky a jednoduchému načítání spojených dat.

A.1.2 SMTP

Jedná se o zkratku pro Simple Mail Transfer Protocol. Tato technologie slouží pro přenos elektronické pošty (e-mailu) mezi odesilatelem a adresátem. Přenos probíhá nad protokolem TCP. Původní norma RFC 821 byla vydána již v roce 1982 a v roce 2001 ji nahradila nová norma RFC 2821, která byla nahrazena normou RFC 5321 [13].

A.1.3 Fronta

Frontou se v programování myslí struktura, která přijme zprávu a zařadí ji na konec. Z fronty se zároveň čte první položka z fronty (FIFO - first in, first out) a přidává se na konec nová položka, která bude zpracována až jako poslední.

Fronta může být tvořena jakoukoli technologií, avšak některé technologie jsou pro frontu přímo navrhnuté (například RabbitMQ nebo Apache Kafka). Jiné technologie nemusí být příliš vhodné (například diskové systémy pro jejich pomalost).

A.1.4 Messaging broker

Jedná se o aplikaci, která převádí zprávu od odesílatele k příjemci. Odesílatel i příjemce navíc může komunikovat s messaging brokerem pomocí jiného protokolu. [4]

A.1.5 Databáze

V celé diplomové práci je databází myšleno jednak uložení dat a jednak systém řízení báze dat. Použitou databází je MySQL. Jedná se o databázi distribuovanou pod GPL licencí od společnosti Oracle [14].

A.1.6 Adapter

Je to návrhový vzor, který má za cíl řešit změnu rozhraní jedné třídy za rozhraní jiné třídy. Typicky přidává nějakou funkčnost, kterou původní třída nebo rozhraní neměly, a tím dovolí původní třídu (nebo rozhraní) připojit k systému, kam by jinak připojena být nemohla.

A.1.7 Rozhraní (interface)

Rozhraní nese informaci o metodách, které musí třída deklarovat, aby mohla rozhraní implementovat. Je to jeden z velice důležitých prvků SOLID principů objektově orientovaného programování. Pakliže knihovna vyžaduje jako svou závislost rozhraní, tak je možné konkrétní implementaci napsat jakkoli. Pokud však vyžaduje konkrétní třídu, tak tuto třídu musí programátor zdědit, jestliže chce upravit výchozí chování knihovny. Všechny knihovny by proto vnitřně měly co nejvíce záviset na rozhraních a až kolem nich vystavět nějakou logiku.

A.1.8 Entita

„Je libovolný objekt (osoba, zvíře, věc či jev) reálného světa, který je zachycen v datovém modelu. Entita musí být rozlišitelná od ostatních entit a existovat nezávisle na nich.“ [15]

A.1.9 PHP

PHP je nejrozšířenější skriptovací jazyk pro web. Název je rekurzivní zkratkou pro PHP: Hypertext Preprocessor. Aktuální nejnovější verze je PHP 7.2. [16]

A.1.10 Composer

Composer je nástroj pro verzování balíčků v jazyce PHP. Zajišťuje kompatibilitu mezi použitými balíčky tím, že nedovolí nainstalovat vzájemně nekompatibilní balíčky.

Tato technologie představuje obrovský průlom v programování v jazyce PHP. Dříve bylo velice obtížné začlenit cizí kód do vlastní aplikace/knihovny. Nyní se o stáhnutí a vložení do aplikace stará Composer.

A.1.11 PSR-4

PSR je zkratkou pro PHP Standards Recommendations (standardy a doporučení pro PHP), které vydává PHP-FIG. PSR-4 je standardem, který se zaměřuje na automatické načítání tříd do aplikace. Hlavní myšlenkou PSR-4 je, že po nadefinování výchozího adresáře odpovídá adresářová struktura struktuře jmenných prostorů.

A.1.12 Balíček

Balíčkem se rozumí ucelená funkcionalita, která zajišťuje konkrétní činnost. Příkladem může být ondrejnov/eet, což je balíček, který se stará o odesílání elektronické evidence tržeb finančnímu úřadu. Tato ucelená funkcionalita pro své fungování může využít jiné balíčky. Například výše uvedená knihovna potřebuje ke svému běhu knihovnu robrichards/wse-php a tato knihovna potřebuje zase další balíček. Takto mohou balíčky vytvořit nekonečný řetězec závislostí.

A.1.13 API (Application Programming Interface)

„Jde o sbírku procedur, funkcí, tříd či protokolů nějaké knihovny (ale třeba i jiného programu nebo jádra operačního systému), které může programátor využívat.“ [17]

A.1.14 Use case diagram

Českým překladem diagram případů užití. Jedná se o diagram, který se zvenku dívá na modelovanou problematiku a hledá, jakým způsobem má jít aplikaci používat. Jaké funkce aplikace má mít a jací aktéři s aplikací budou interagovat.

A.1.15 PDO

PDO je rozhraní pro práci s databázemi. Obsahuje několik ovladačů, například pro MySQL a PostgreSQL databáze. Jeho největší předností je ochrana před SQL injection útokem, což je závažný útok, který může vést ke stažení veškerých dat z databáze nebo dokonce ke kompletnímu smazání databáze.

A.1.16 Garbage collector

Slouží ke správě paměti. Speciální algoritmus vyhledává nepoužívané části paměti a ty uvolňuje.

A.1.17 Message-oriented middleware

Jedná se o softwarovou nebo hardwarovou infrastrukturu, která se stará o přenos zprávy z aplikace do message brokera a z message brokera do jiné aplikace.

A.1.18 AMQP (Advanced Message Queuing Protocol)

AMQP je protokol. Je to vrstva, která obaluje message-oriented middleware a stará se o přepravu zprávy, routování i řazení do fronty. Je to hlavní protokol, který využívá RabbitMQ.

A.1.19 (TTMP) Streaming Text Oriented Messaging Protocol

TTMP je jednoduchý, textově založený protokol designovaný pro message-oriented middleware. [18]

A.1.20 Message Queuing Telemetry Transport

Jedná se o ISO standard (ISO/IEC PRF 20922) pro zprávy založené na protokolu publish-subscribe. Je designovaný pro TCP/IP protokol s malým datovým tokem, takže je vhodný k použití v případě, kdy je k dispozici malá šířka pásma. [19]

A.1.21 TTL – time to live

Česky doba života. V této diplomové práci se používá výhradně ve spojení se zprávou pro RabbitMQ. Nastavuje, po jak dlouho dobu bude zpráva „žít“ ve frontě. Po uplynutí této doby bude z fronty odstraněna a předána Death Letter Exchange nebo zahozena.

A.1.22 RPC (Remote procedure call)

Česky vzdálené volání procedur. Jde o „technologii dovolující programu vykonat kód na jiném místě, než je umístěn volající program. Příkladem budiž provedení složitějšího matematického výpočtu výkonnějším počítačem, dostupným skrze síť.“ [20]

A.1.23 TCP (Transmission Control Protocol)

„Je nejpoužívanějším protokolem transportní vrstvy v sadě protokolů TCP/IP používaných v síti Internet. Použitím TCP mohou aplikace na počítačích propojených do sítě vytvořit mezi sebou spojení, přes které mohou obousměrně přenášet data. Protokol garantuje spolehlivé doručování a doručování ve správném pořadí. TCP také umožňuje rozlišovat a rozdělovat data pro více aplikací (například webový server a emailový server) běžících na stejném počítači.“ [21]

A.1.24 ORM (Object-relational mapping)

Česky Objektově-relační mapování. Je to mapování relační databáze na objekty. Dovoluje to vývojáři psát a myslet „v objektech“, ale data ukládat strukturovaně do relačně mapované databáze.

A.1.25 ElastSearch

Jedná se o velice rychlý fultextový vyhledávač. Je napsaný v jave a je vydáván zdarma pod licencí Apache. Hlavními rysy jsou jeho veliká rychlost (data jsou zobrazena téměř okamžitě) a vysoká dostupnost. [22]