DELTA – Střední škola informatiky a ekonomie, s.r.o.

Ke Kamenci 151, Pardubice

Implementace technologie Elasticsearch a RabbitMQ pro optimalizaci e-shopu

Příjmení, jméno: **Žďárský, Lukáš**

Třída: **4.A**

Studijní obor: **Informační technologie 18-20-M/01**

Školní rok: **2025/2026**

Zadání maturitního projektu

Jméno a příjmení: Lukáš Žďárský
Pro školní rok: 2025/2026
Třída: 4.A
Obor: Informační technologie 18-20-M/01
Téma práce: Implementace technologie Elasticsearch a RabbitMQ pro optimalizaci e--shopu
Vedoucí práce: Jan Čech

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cíl projektu:
Cílem tohoto projektu je implementace technologie Elasticsearch a RabbitMQ pro zrychlení a optimalizaci e-shopu u jednoho z klientů firmy. Projekt zahrnuje nasazení Elasticsearch na server klienta, integraci RabbitMQ pro distribuci dat do Elasticsearch a implementaci obou technologií do samotného e-shopu. Projekt se zaměřuje na optimalizaci načítání dat, zejména pro filtrování kategorií a vyhledávání v e-shopu.

Specifikace projektu:
- Provedení analýzy stávající struktury e-shopu a návrh architektury integrace Elasticsearch a RabbitMQ.
- Instalace a konfigurace Elasticsearch na serveru; navržení databázové struktury pro ukládání dat.
- Implementace integrace Elasticsearch do e-shopu pro filtrování kategorií a vyhledávání produktů.
- Instalace a konfigurace RabbitMQ pro distribuci dat mezi e-shopem a Elasticsearch.
- Testování výkonnosti a ladění systému.

Požadované výstupy:
- Funkční implementace Elasticsearch a RabbitMQ na serveru a v e-shopu klienta.
- Dokumentace popisující implementaci Elasticsearch, RabbitMQ a jejich integraci do e-shopu.
- Testování a vyhodnocení výkonu a efektivity implementace.

Stručný časový harmonogram:

Září: Analýza stavu e-shopu, návrh architektury.

Říjen: Instalace a konfigurace ES na serveru.

Listopad: Implementace RabbitMQ.

Prosinec: Implementace ES pro vyhledávání a filtrování.

Leden: Testování a ladění.

Únor–březen: Finalizace, dokumentace.

Prohlašuji, že jsem maturitní projekt vypracoval samostatně, výhradně s použitím uvedené literatury.

V Pardubicích dne ……

…….

podpis autora

Rád bych poděkoval Janu Čechovi za odborné vedení maturitního projektu a cenné připomínky v průběhu celé práce. Dále děkuji Dominiku Šlechtovi za pomoc při řešení projektu a přínosné diskuze ohledně implementace.

Resumé

Práce se zabývá návrhem a implementací třívrstvé architektury pro integraci technologií Elasticsearch a RabbitMQ do e-shopových projektů. Hlavním cílem je zrychlení full-textového vyhledávání a facetového filtrování produktů, které jsou při použití tradičních SQL dotazů na rozsáhlých datasetech výrazně pomalejší. Řešení sestává z centralizovaného Elasticsearch serveru, klientského Composer balíčku pro integraci do e-shopů a samotné implementace do cílového e-shopu. Server zajišťuje indexování dat prostřednictvím RabbitMQ a zpracování vyhledávacích dotazů. Klientský balíček poskytuje univerzální rozhraní pro komunikaci se serverem a mapování dat. Výsledný systém je nasazen v produkčním prostředí a podporuje současný provoz více e-shopových projektů. Měřením bylo dosaženo přibližně 90% zrychlení doby odezvy vyhledávacích dotazů oproti původním SQL dotazům.

Klíčová slova: Elasticsearch, RabbitMQ, full-text vyhledávání, e-shop, indexování, facetové filtrování, PHP, Nette Framework, Docker

Abstract

This thesis addresses the design and implementation of a three-layer architecture for integrating Elasticsearch and RabbitMQ technologies into e-commerce projects. The primary objective is to accelerate full-text search and faceted product filtering, which are significantly slower when using traditional SQL queries on large datasets. The solution consists of a centralized Elasticsearch server, a client Composer package for e-shop integration, and the actual implementation into the target e-shop. The server handles data indexing via RabbitMQ and processes search queries. The client package provides a universal interface for server communication and data mapping. The resulting system is deployed in a production environment and supports simultaneous operation of multiple e-shop projects. Measurements show an approximately 90% reduction in search query response times compared to the original SQL queries.

Keywords: Elasticsearch, RabbitMQ, full-text search, e-commerce, indexing, faceted filtering, PHP, Nette Framework, Docker

Obsah

1 Úvod

E-shopy s rozsáhlým katalogem produktů se pravidelně potýkají s problémem pomalého vyhledávání a filtrování. Při tisících až desítkách tisíc produktů s vazbami na varianty, vlastnosti, příslušenství a kategorie se SQL dotazy využívající klauzuli `LIKE` stávají výkonnostním úzkým hrdlem. Situace se dále zhoršuje při facetovém filtrování, kde je nutné pro každý filtr (cenový rozsah, materiál, barva, rozměry) spočítat počet odpovídajících produktů — což v SQL vyžaduje sérii agregačních dotazů nad celým datasetem.

Tato práce vznikla v kontextu firmy CzechGroup, která provozuje a vyvíjí desítky e-shopů postavených na frameworku Nette. Potřeba řešení přesahujícího možnosti SQL full-textu se projevila zejména u projektů s rozsáhlými katalogy, kde doba odezvy filtrovacích dotazů přesahovala jednotky sekund a negativně ovlivňovala uživatelský zážitek.

Cílem projektu je navrhnout a implementovat třívrstvou architekturu integrující technologie Elasticsearch a RabbitMQ do stávajících e-shopů. Řešení sestává ze tří částí:

- **Elasticsearch server** — centralizovaný server spravující indexy, zpracovávající vyhledávací dotazy a přijímající data k indexování prostřednictvím message brokeru RabbitMQ. Server je navržen jako multi-tenantní — podporuje současný provoz více e-shopových projektů s izolovanými indexy.
- **Elasticsearch client** — Composer balíček, který se instaluje do e-shopu a poskytuje univerzální rozhraní pro komunikaci se serverem. Zajišťuje mapování databázových entit na dokumenty Elasticsearch, správu indexovacích úloh a sestavování vyhledávacích dotazů.
- **Implementace do e-shopu** — úpravy v cílovém e-shopu, kde byl Elasticsearch integrován do vyhledávání produktů, filtrování v kategoriích a napovídání při psaní (typeahead). Při nedostupnosti Elasticsearch systém automaticky přepne na původní SQL dotazy.

Práce popisuje návrh architektury, implementaci jednotlivých vrstev, vzájemnou komunikaci komponent a nasazení systému do produkčního prostředí.

2 Teoretická část

2.1 Problém full-textového vyhledávání v SQL

Relační databáze jako MariaDB nabízejí dva základní přístupy k textovému vyhledávání. Klauzule `LIKE '%výraz%'` provádí sekvenční procházení všech záznamů v tabulce, což při tisících produktů vede k nepřijatelným dobám odezvy. Vestavěný `FULLTEXT` index tento problém částečně řeší vytvořením invertovaného indexu nad textovými sloupci, avšak jeho možnosti

jsou omezené — nepodporuje pokročilé jazykové analyzéry, skóring relevance je základní a konfigurace tokenizace minimální [1].

Zásadnějším problémem je facetové filtrování. Moderní e-shopy zobrazují u každé kategorie filtrovací panel s počty odpovídajících produktů — například „Materiál: dřevo (42), kov (18), plast (7)“. Pro výpočet těchto počtů je v SQL nutné provést agregační dotaz pro každý filtr zvlášť, přičemž každý dotaz musí respektovat aktuálně aktivní filtry. Při desítkách filtrů a tisících produktů se celková doba odezvy stává neúnosnou.

Řešením je použití dedikovaného vyhledávacího engine, který je pro tyto operace optimalizován. Elasticsearch ukládá data v invertovaném indexu, kde jsou dokumenty indexovány podle jednotlivých termů. Vyhledání dokumentů obsahujících konkrétní term je tak konstantní operací nezávislou na celkovém počtu dokumentů. Agregace (facety) jsou nativní součástí dotazovacího jazyka a jsou počítány paralelně v rámci jednoho dotazu [1].

2.2 Elasticsearch

Elasticsearch je distribuovaný vyhledávací a analytický engine postavený na knihovně Apache Lucene. Data jsou ukládána ve formě JSON dokumentů v indexech, přičemž každý dokument je automaticky indexován pro rychlé vyhledávání [1].

Klíčovým principem je **invertovaný index** — datová struktura, která pro každý unikátní term uchovává seznam dokumentů, ve kterých se term vyskytuje. Při vyhledání výrazu „dřevěná police“ engine rozloží dotaz na jednotlivé termy, vyhledá je v invertovaném indexu a vrátí průnik odpovídajících dokumentů.

Před uložením do indexu prochází text řetězcem **analyzérů**, které zajišťují tokenizaci (rozdělení na slova), normalizaci (převod na malá písmena, odstranění diakritiky) a případně stemming (převod na kořen slova). Díky tomu dotaz „dřevěné“ nalezne i dokumenty obsahující „dřevěná“ nebo „dřevo“.

Mapping definuje strukturu dokumentu v indexu — datové typy polí, použité analyzéry a způsob indexování. Mapping je analogií databázového schématu, avšak s tím rozdílem, že Elasticsearch podporuje vnořené objekty a pole přímo na úrovni dokumentu.

Query DSL (Domain Specific Language) je dotazovací jazyk ve formátu JSON, který umožňuje sestavovat složité dotazy kombinující full-textové vyhledávání, přesné shody, rozsahové filtry a vnořené dotazy. Dotazy typu `bool` umožňují logické kombinace podmínek pomocí klauzulí `must`, `should`, `must_not` a `filter`.

Agregace jsou mechanismem pro výpočet statistik nad výsledky dotazu. Pro e-shop jsou klíčové zejména `terms` agregace (počty hodnot pro filtry), `min`/`max` agregace (cenové rozsahy) a vnořené agregace pro hierarchická data. Agregace jsou počítány v rámci jednoho HTTP požadavku společně s vyhledáváním [1].

2.3 RabbitMQ

RabbitMQ je open-source message broker implementující protokol AMQP (Advanced Message Queuing Protocol). Slouží jako prostředník pro asynchronní komunikaci mezi aplikacemi — producent odesílá zprávy do fronty a konzument je z fronty postupně zpracovává [2].

V kontextu integrace Elasticsearch do e-shopu plní RabbitMQ roli prostředníka při indexování dat. Když je v e-shopu upraven produkt, klientský balíček odešle data na server, který je zařadí do RabbitMQ fronty. Konzument (dlouho běžící proces) zprávy postupně zpracovává a indexuje do Elasticsearch. Tento přístup přináší několik výhod:

- **Decoupling** — e-shop nemusí přímo komunikovat s Elasticsearch, což snižuje provázanost systémů.
- **Spolehlivost** — zprávy ve frontě jsou persistentní (přežijí restart brokeru). Konzument potvrzuje zpracování každé zprávy (acknowledgement); nepotvrzené zprávy jsou automaticky vráceny do fronty [2].
- **Škálovatelnost** — při zvýšené zátěži (například hromadná aktualizace katalogu) se zprávy hromadí ve frontě a jsou zpracovány postupně, aniž by došlo k přetížení Elasticsearch.
- **Retry mechanismus** — při selhání zpracování zprávy je možné ji vrátit do fronty s nastaveným počtem opakování.

Fronty v RabbitMQ jsou konfigurovány jako **durable** (trvalé), což zajišťuje jejich přežití při restartu serveru. Zprávy jsou označeny jako **persistent**, čímž je zajištěno jejich uložení na disk [3].

2.4 Použité technologie

Pro vývoj jednotlivých vrstev systému byly zvoleny následující technologie:

- **PHP 8.1+** — programovací jazyk pro klientský balíček a cílový e-shop. Server využívá PHP 8.5 s nejnovějšími jazykovými konstrukty [6].
- **Nette Framework 3.1/3.2** — PHP framework použitý ve všech vrstvách projektu. Poskytuje dependency injection kontejner, routování, šablonovací systém a konfiguraci prostřednictvím souborů NEON [4].

- **Nextras ORM/DBAL** — objektově-relační mapovač a databázová abstraktní vrstva používaná v klientském balíčku a na serveru pro správu entit (joby, fronta, produkty) [5].
- **Elasticsearch 9.2** — vyhledávací engine s oficiální PHP klientskou knihovnou (verze ^8.0) [1].
- **RabbitMQ** — message broker s PHP knihovnou php-amqplib pro komunikaci přes AMQP protokol [2].
- **Docker a Docker Compose** — kontejnerizační platforma pro nasazení serveru (Elasticsearch, RabbitMQ, Apache, MariaDB) [7].
- **PHPStan** — nástroj pro statickou analýzu PHP kódu, který odhaluje typové chyby bez nutnosti spuštění kódu [8].
- **PHP CS Fixer** — nástroj pro automatickou úpravu formátování PHP kódu podle definovaných pravidel.

3 Metodika a vlastní řešení

3.1 Architektura systému

Systém je navržen jako třívrstvá architektura, kde každá vrstva plní specifickou roli a komunikuje s ostatními prostřednictvím HTTP API. Vrstvy jsou na sobě nezávislé a mohou být nasazeny na různých serverech.

Elasticsearch server (oranžová vrstva v diagramech) je centralizovaná služba spravující Elasticsearch indexy a RabbitMQ frontu. Přijímá data k indexování od klientských balíčků a zpracovává vyhledávací dotazy. Server je navržen jako multi-tenantní — jeden server obsluhuje více e-shopových projektů s izolovanými indexy.

Elasticsearch client (modrá vrstva) je Composer balíček instalovaný do e-shopu. Poskytuje fasádu pro komunikaci se serverem, mapování databázových entit na ES dokumenty, správu indexovacích úloh a sestavování vyhledávacích dotazů. Balíček je univerzální — projektově specifické rozdíly se řeší prostřednictvím konfigurace a DataMapper tříd.

E-shop (červená vrstva) je cílová aplikace, do které se Elasticsearch integruje. Využívá klientský balíček pro vyhledávání a indexování, přičemž zachovává SQL fallback pro případ nedostupnosti Elasticsearch.

Komunikační toky mezi vrstvami jsou dva:

Indexování dat: E-shop → Client (vytvoření jobu, příprava dat) → HTTP POST → Server (validace, zařazení do RabbitMQ) → Consumer (zpracování zpráv, indexování do ES).
Vyhledávání: E-shop → Client (sestavení strukturovaného požadavku) → HTTP POST → Server (validace požadavku, sestavení ES dotazu, provedení) → Response s výsledky a agregacemi zpět do e-shopu.
Přehled všech interakcí mezi vrstvami a aktéry systému je znázorněn na Obrázek 1.

Obrázek 1: Use Case diagram systému — červená: e-shop, modrá: client, oranžová: server
Diagram identifikuje tři aktéry a následující případy užití:

3.1.1 UC1: Vyhledávání produktů (Search)

- **Aktéři:** Uživatel e-shopu
- **Podmínky spouštění:** Uživatel zadá vyhledávací dotaz nebo aktivuje filtr v kategorii.
- **Základní tok:**
  1. E-shop ověří dostupnost Elasticsearch serveru prostřednictvím klientského balíčku (health check).
  2. Klientský balíček sestaví strukturovaný vyhledávací požadavek (mapování parametrů).
  3. Požadavek je odeslán na server, který provede dotaz nad Elasticsearch indexem.
  4. Server vrátí výsledky (identifikátory produktů, agregace) zpět klientskému balíčku.
  5. E-shop načte produkty z databáze podle vrácených identifikátorů a zobrazí je uživateli.

- **Alternativní tok:** Pokud je Elasticsearch nedostupný nebo dotaz selže, systém automaticky přepne na původní SQL vyhledávání (Fallback to SQL search).
- **Podmínky pro dokončení:** Uživateli jsou zobrazeny výsledky vyhledávání.

3.1.2 UC2: Editace produktu (Edit product)

- **Aktéři:** Administrátor e-shopu
- **Podmínky spouštění:** Administrátor upraví produkt v administračním rozhraní.
- **Základní tok:**
  1. E-shop zavolá klientský balíček s identifikátorem upraveného produktu.
  2. Klientský balíček vytvoří indexovací job a zařadí produkt a jeho doplňkové entity (varianty, vlastnosti, příslušenství) do lokální fronty.
  3. Při dalším zpracování fronty (cron) jsou data odeslána na server.
- **Podmínky pro dokončení:** Entita je zařazena do lokální fronty k odeslání.

3.1.3 UC3: Editace menu položky (Edit menuitem)

- **Aktéři:** Administrátor e-shopu
- **Podmínky spouštění:** Administrátor upraví kategorii (menu položku) v administračním rozhraní.
- **Základní tok:** Analogický s UC2, avšak zahrnuje entitu typu menuitem místo produktu.
- **Podmínky pro dokončení:** Entita je zařazena do lokální fronty k odeslání.

3.1.4 UC4: Kompletní reindexace (Trigger full update)

- **Aktéři:** Cron (automatický plánovač, interval 7 dní)
- **Podmínky spouštění:** Uplynutí nastaveného intervalu.
- **Základní tok:**
  1. Cron zavolá API endpoint e-shopu pro spuštění kompletní aktualizace.
  2. Klientský balíček vytvoří full-update job a zařadí všechny entity do lokální fronty s rozdělením do částí (partitioning).
  3. Při následném zpracování fronty jsou data postupně odesílána na server.
  4. Server zařadí data do RabbitMQ fronty, konzument je zpracuje a naindexuje do Elasticsearch.
  5. Po dokončení všech částí server atomicky přepne produkční alias na nové indexy.
- **Podmínky pro dokončení:** Všechny entity jsou naindexovány a produkční alias je přepnut.

3.1.5 UC5: Zpracování fronty (Trigger queue update)

- **Aktéři:** Cron (automatický plánovač, interval 5 minut)
- **Podmínky spouštění:** Uplynutí nastaveného intervalu a existence připravených částí v lokální frontě.
- **Základní tok:**
  1. Cron zavolá API endpoint e-shopu pro zpracování fronty.
  2. Klientský balíček načte jednu připravenou část fronty, transformuje entity pomocí DataMapperu a odešle na server.
  3. Server validuje data a publikuje je do RabbitMQ fronty.
  4. Konzument zprávy postupně zpracovává a indexuje do Elasticsearch.
  5. Konzument ověří, zda byly všechny zprávy úspěšně zpracovány; v případě selhání jsou zprávy vráceny do fronty.
- **Alternativní tok:** Pokud odeslání na server selže, klientský balíček dekrementuje počet zbývajících pokusů a část zůstane ve frontě pro další zpracování.
- **Podmínky pro dokončení:** Data jsou úspěšně naindexována v Elasticsearch, nebo je vyčerpán maximální počet pokusů.

3.2 Elasticsearch Server

Elasticsearch server je Nette Framework aplikace zajišťující správu indexů, zpracování indexovacích úloh a vyřizování vyhledávacích dotazů. Je nasazen jako sada Docker kontejnerů na produkčním serveru.

3.2.1 Struktura projektu

Projekt je organizován do následujících vrstev:

- `Api/Presenters/` — HTTP endpointy (ElasticPresenter pro vyhledávání a správu indexů, RabbitPresenter pro příjem dat k indexování).
- `Controller/` — nízkoúrovňové operace s Elasticsearch a RabbitMQ.
- `Service/` — aplikační logika (ElasticManagementService, SearchService, JobManagementService, ValidatorService).
- `Search/` — sestavování ES dotazů (ElasticQueryBuilder, AggregationBuilder, SearchRequestBuilder, DSL objekty).
- `Validator/` — validace dat dle projektového schématu (AbstractValidator, DrevojasValidator, ValidatorFactory).

- `DTO/` — datové přenosové objekty pro validaci, RabbitMQ zprávy a vyhledávací požadavky.
- `Model/` — Nextras ORM entity a repozitáře pro správu jobů.
- `Constants/` — enumerace (Project, JobType, JobStatus, QueueType).
- `Mapping/` — generování ES mappingů z validačních schémat.

3.2.2 Multi-tenantní podpora

Server podporuje současný provoz více e-shopových projektů. Každý projekt je registrován v enumeraci `Project` (Drevojas, Fencee, Dogtrace) a má vlastní:

- **Izolované indexy** — pojmenované podle konvence `{projekt}-{entita}-{jobId}` pro dočasné indexy a `{projekt}-{entita}-stable` pro produkční alias.
- **Validační schéma** — definované v příslušné Validator třídě (např. `DrevojasValidator`), která specifikuje strukturu produktu, variant, vlastností a dalších entit.
- **ES mapping** — generovaný automaticky z validačního schématu pomocí `MappingBuilder`, který převádí definice z `Nette\Schema` na datové typy Elasticsearch.

`ValidatorFactory` na základě názvu projektu v příchozím požadavku vrátí odpovídající Validator instanci. Přidání nového projektu tak vyžaduje pouze vytvoření nové Validator.

3.2.3 Indexování dat

Proces indexování začíná příjmem dat na endpointu `POST /api/rabbit/enqueue`. RabbitPresenter provede dvoustupňovou validaci — nejprve ověří strukturu hlaviček zprávy (typ jobu, číslo části, celkový počet částí, název projektu), poté validuje samotná data proti schématu odpovídajícího Validatoru. Dvoustupňový přístup je zvolen z důvodu paměťové efektivity — při nevalidních hlavičkách není nutné načítat celý dataset.

Po úspěšné validaci je vytvořen nebo aktualizován záznam jobu v databázi a zpráva je publikována do RabbitMQ fronty. Konzument (`bin/consumer.php`) běží jako dlouho běžící proces, který naslouchá na frontě a zpracovává zprávy postupně (prefetch_count=1). Pro každou zprávu:

1. Extrahuje a validuje AMQP hlavičky.
2. Validuje data proti projektovému schématu.
3. Indexuje dokumenty do Elasticsearch (bulk upsert).
4. Inkrementuje počítadlo zpracovaných částí jobu.
5. Potvrdí zpracování zprávy (acknowledgement).

Při selhání zpracování je zpráva vrácena do fronty s dekrementovaným počtem zbývajících pokusů. Po vyčerpání pokusů je zpráva zahozena.
Hlavní entity (produkty, menuitemy) jsou indexovány jako samostatné dokumenty s identifikátorem `{id}-{lang}`. Doplňkové entity (varianty, vlastnosti, příslušenství) jsou přidávány do polí rodičovského dokumentu prostřednictvím skriptového updatu, který pomocí `HashSet` zabraňuje duplicitám.

3.2.4 Správa indexů a jobů

Systém rozlišuje dva typy indexovacích úloh:
Full Update — kompletní reindexace všech dat projektu. Vytvoří nové dočasné indexy, do kterých se postupně indexují všechna data. Po dokončení se atomicky přepne produkční alias na nové indexy, čímž je zajištěn zero-downtime přechod.
Edit Product / Edit Menuitem — aktualizace konkrétní entity. Vytvoří dočasný index, do kterého se naindexuje aktualizovaná entita, a poté provede reindex do stabilního produkčního indexu.
Přehled možných stavů jobu je uveden v Tabulka 1. Životní cyklus jobu je sledován v databázové tabulce `jobs` s atributy `expected_parts`, `received_parts` a `finished_parts`. Endpoint `POST /api/elastic/check-jobs` (volaný periodicky) kontroluje stav běžících jobů — dokončené joby finalizuje (přepnutí aliasu nebo reindex), nedokončené joby po překročení timeoutu (počet očekávaných částí × 10 minut) označí jako neúspěšné.
Endpoint `POST /api/elastic/cleanup` zajišťuje mazání osiřelých indexů — indexů, které nejsou napojeny na žádný aktivní job ani produkční alias.

Stav jobu	**Popis**
Running	Aktivní job, přijímá části z RabbitMQ
Finished	Všechny části úspěšně zpracovány, index finalizován
Failed	Detekována nekonzistence dat
Timeouted	Překročen časový limit pro dokončení

Tabulka 1: Stavy indexovacích jobů

3.2.5 Vyhledávání

Vyhledávací dotazy jsou přijímány na endpointu `POST /api/elastic/search`. `SearchService` validuje příchozí požadavek, identifikuje projekt a předá dotaz `ElasticQueryBuilder`, který sestaví Elasticsearch Query DSL.
Podporované typy filtrování:

- **Full-textové vyhledávání** — využití `match` dotazů napříč konfiguratelnými poli (název, popis, kód, objednací kód).
- **Filtry vlastností (property filters)** — vnořené dotazy na pole `property` s podporou hodnotových filtrů (barva, materiál) a rozsahových filtrů (šířka, hloubka).
- **Filtry příslušenství (supplement filters)** — vnořené dotazy na pole `product_supplement` pro binární filtry (novinka, sleva, s dřezem).
- **Filtry kategorií (menuitem filters)** — hierarchické filtrování s podporou podkategorií. Server rozbalí hierarchii na základě vazby rodič–potomek.
- **Řazení** — standardní řazení podle ceny (vzestupně/sestupně) a prioritní řazení pomocí Painless skriptu, který počítá skóre na základě vnořených polí příslušenství.

Pro facetové filtrování je implementována strategie **post-filter** — filtry jsou aplikovány až po výpočtu agregací, čímž uživatel vidí počty odpovídajících produktů pro všechny dostupné hodnoty filtrů, nikoli jen pro aktuálně vybranou kombinaci.

Endpoint `GET /api/elastic/healthcheck?projectName={project}` ověřuje existenci všech hlavních indexů projektu a slouží pro kontrolu dostupnosti před odesláním dotazu.

3.2.6 Validator pattern

Každý projekt definuje svou Validator třídu rozšiřující `AbstractValidator`. Validator specifikuje:

- **Projektový název** — návratová hodnota metody `defineProjectName()`, odpovídající hodnotě v enumeraci `Project`.
- **Schémata entit** — metody `defineProductSchema()`, `defineProductVariantSchema()`, `defineMenuitemSchema()` a další definují strukturu dat pomocí `Nette\Schema\Expect`. Každé schéma určuje povinná a volitelná pole s datovými typy.
- **Mapování filtrů** — metody `definePropertyFilterMapping()`, `defineSupplementFilterMapping()` specifikují názvy polí pro vyhledávací filtry.

Z validačních schémat `MappingBuilder` automaticky generuje Elasticsearch mapping — převod typů je znázorněn v Tabulka 2.

Nette Schema typ	ES mapping typ	Poznámka
`Expect::string()`	`text` + `.keyword`	Full-text + přesná shoda
`Expect::int()`	`integer`	Číselné hodnoty
`Expect::float()`	`float`	Desetinná čísla

Nette Schema typ	ES mapping typ	Poznámka
`expectDateTime()`	`date`	ISO 8601 formát
`Expect::list()`	`nested`	Vnořené objekty

Tabulka 2: Převod typů z Nette Schema na Elasticsearch mapping

3.3 Elasticsearch Client

Elasticsearch client je Composer balíček (`czechgroup/elasticsearch-client`), který se instaluje do e-shopu a poskytuje kompletní rozhraní pro komunikaci s Elasticsearch serverem.

3.3.1 Architektura balíčku

Balíček je navržen podle principu fasády — třída `Client` poskytuje jednoduché veřejné API (`updateProduct()`, `updateMenuitem()`, `updateAll()`, `search()`, `isHealthy()`), zatímco příkazy interně deleguje na specializované služby:

- `JobManagementService` — vytváření a správa indexovacích jobů.
- `QueueManagementService` — rozdělování dat do částí a příprava fronty.
- `SearchService` — sestavování a odesílání vyhledávacích dotazů.
- `ServerCommunicationService` — HTTP komunikace se serverem (cURL).

Datová vrstva využívá Nextras ORM pro správu dvou tabulek: `entity_update_job` (sledování jobů) a `entity_update_queue` (fronta entit k odeslání).

3.3.2 Nette DI Extension

Integrace do e-shopu probíhá prostřednictvím Nette DI extension `ElasticsearchClientExtension`, která se registruje v `config.neon` e-shopu. Extension přijímá konfiguraci zahrnující:

- **Připojení k serveru** — host, port, HTTPS, API klíče (klientský pro autentizaci požadavků, serverový pro ověření callbacků).
- **Projektový název** — identifikátor projektu odpovídající enumeraci na serveru.
- **Konfigurace vyhledávání** — defaultní filtry, vyhledávací pole, řazení, filtrování podle vlastností, příslušenství a kategorií.
- **Limity partitioningu** — maximální počet entit na jednu část fronty (výchozí: 8 000 produktů, 100 000 variant na část).

Extension automaticky zaregistruje všechny potřebné služby, repozitáře, entity a ApiPresenter do DI kontejneru.

3.3.3 Správa indexovacích jobů

Při zavolání metody `Client::updateProduct($id)` proběhne následující proces:

1. `JobManagementService` vytvoří záznam jobu v lokální databázi e-shopu (typ: edit-product, entita: $id, stav: pending).
2. `QueueItemFactory` vloží do fronty všechny související entity — produkt, jeho varianty, vlastnosti, příslušenství a vazby na kategorie. Každý záznam fronty uchovává identifikátory entity, podentity, jazyk a typ.
3. Záznamy fronty jsou rozděleny do **částí (parts)** podle konfigurovatelných limitů. Při překročení limitu (např. 100 000 variant) jsou záznamy rozděleny do více částí, každé s přiřazeným `part_number`.

Zpracování fronty probíhá voláním endpointu `POST /api/elastic/processQueue` (typicky z cronu). ApiPresenter provádí následující kroky:

1. Vezme jednu připravenou část fronty.
2. Načte odpovídající entity z databáze prostřednictvím Nextras ORM.
3. Transformuje entity na ES dokumenty pomocí DataMapperu.
4. Odešle data na server (`POST /app-api/rabbit/enqueue`) s hlavičkami obsahujícími ID jobu, číslo části, celkový počet částí a typ entity.
5. Podle HTTP odpovědi aktualizuje stav části (enqueued / enqueue-failed s dekrementací pokusů).

Server po dokončení zpracování zavolá zpět endpoint `POST /api/elastic/updateJobStatus` s výsledkem, čímž se aktualizuje stav jobu v lokální databázi.

3.3.4 DataMapper pattern

DataMapper zajišťuje transformaci Nextras ORM entit na pole (asociativní array) vhodné pro indexování do Elasticsearch. Architektura využívá dědičnost:

`AbstractDataMapper` definuje pomocné metody pro běžné transformace — `PropertyValueSplitter` rozděluje hodnoty vlastností (např. „červená,modrá“ → pole dvou hodnot), `WebalizeValueConvertor` převádí hodnoty na URL-safe tvar (pro facetové filtrování), `NumericValueConvertor` převádí řetězcové číselné hodnoty na typ `float` a `DateTimeMapper` převádí `DateTime` na ISO 8601 formát.

`DefaultDataMapper` poskytuje výchozí mapování pro všechny typy entit (produkt, varianta, vlastnost, příslušenství, menuitem). `DrevojasDataMapper` rozšiřuje výchozí mapper o projektově specifické úpravy — například přidání pole `order_code` u variant.
`DataMapperFactory` dynamicky vybírá mapper na základě konfigurace — buď projektově specifický mapper (název odvozený z názvu projektu), nebo výchozí `DefaultDataMapper`.

3.3.5 Vyhledávání

Vyhledávací dotaz je sestavován třídou `SearchRequestBuilder`, která transformuje parametry z e-shopu (aktivní filtry, řazení, stránkování, hledaný text) na strukturovaný JSON požadavek kompatibilní se serverovým API. `SearchService` odešle požadavek prostřednictvím `ServerCommunicationService` a vrátí typovaný `SearchResponse` obsahující nalezené dokumenty, celkový počet výsledků a agregace pro filtrování. Přehled API endpointů klientského balíčku je uveden v Tabulka 3.

Endpoint	**Metoda**	**Popis**
fullUpdate	POST	Spustí kompletní reindexaci všech entit
processQueue	POST	Zpracuje jednu připravenou část fronty
updateJobStatus	POST	Callback ze serveru o dokončení jobu

Tabulka 3: API endpointy klientského balíčku

3.4 Implementace do e-shopu

Integrace Elasticsearch do cílového e-shopu (Drevojas) vyžaduje úpravy v několika vrstvách aplikace. Cílem je nahradit SQL dotazy pro vyhledávání a filtrování voláním Elasticsearch, přičemž při jakékoliv chybě systém automaticky přepne na původní SQL logiku.

3.4.1 Konfigurace

V souboru `config.neon` e-shopu je registrována DI extension klientského balíčku s kompletní konfigurací:

- **Defaultní filtrovací podmínky** — odpovídající podmínkám z SQL dotazu (produkt není skrytý, má skladovou dostupnost nebo je označen jako důležitý, není expirovaný).
- **Vyhledávací pole** — název, popis, kód produktu a objednací kód s operátory `match`.
- **Rozsahové filtry** — cenový rozsah (`final_pricevat`), šířka a hloubka (identifikátory vlastností).
- **Hodnotové filtry vlastností** — tvar, značka, provedení, typ madla, typ dřezu a další (identifikátory vlastností).

- **Filtry příslušenství** — novinka, sleva, s dřezem/bez dřezu, strana, lesk (identifikátory supplementů).
- **Filtry kategorií** — kolekce (menuitemy).
- **Řazení** — nejlevnější, nejdražší, doporučené (prioritní řazení na základě supplementů).

Hodnoty identifikátorů vlastností a příslušenství jsou převzaty z existující třídy `Filter.php` e-shopu, kde jsou definovány pro SQL filtrování.

3.4.2 ElasticProductSearch

Wrapper třída `ElasticProductSearch` zapouzdřuje dva typy vyhledávání:

- `searchWithFilters()` — plnohodnotné vyhledávání s filtry, řazením, stránkováním a agregacemi. Využíváno při procházení kategorií a na stránce vyhledávání.
- `searchByText()` — jednoduché full-textové vyhledávání bez agregací s limitem výsledků. Využíváno pro typeahead (napovídání při psaní).

Obě metody nejprve ověří dostupnost serveru voláním health checku. Při jakékoliv chybě (nedostupnost serveru, chyba dotazu, timeout) vrátí `null`, čímž signalizují nutnost SQL fallbacku.

3.4.3 Úpravy filtrovací komponenty

Nejvýznamnější úpravou v e-shopu je rozšíření třídy `Filter.php` o metodu `prepareQueryFromElastic()`. Tato metoda:

1. Zavolá `ElasticProductSearch::searchWithFilters()` s aktuálními parametry filtru.
2. Zpracuje agregace z odpovědi serveru — extrahuje minimální a maximální hodnoty pro cenové a rozsahové slidery, počty produktů pro hodnotové filtry vlastností, příslušenství a kategorií.
3. Nastaví výchozí hodnoty formulářových prvků (slidery, checkboxy) na základě agregací.
4. Uloží identifikátory nalezených produktů pro následné načtení z databáze.

Metody `setCategory()` a `setAllProducts()` nejprve zavolají `prepareQueryFromElastic()`. Pokud metoda uspěje, je nastaven příznak `$elasticSearchUsed` na `true`. V opačném případě se pokračuje původní SQL logikou beze změny.

3.4.4 Úpravy presenterů

V `HomepagePresenter` (renderování kategorie) a v traitu `Search` (stránka vyhledávání) je implementována podmínka na příznak `isElasticSearchUsed()`:

- **Elasticsearch cesta** — z odpovědi serveru jsou získány identifikátory produktů a celkový počet výsledků. Produkty jsou načteny z databáze podle identifikátorů s řazením odpovídajícím ES pořadí (`ORDER BY FIELD(id, ...)`). Celkový počet je předán paginátoru.
- **SQL cesta** — původní logika zůstává beze změny jako záložní varianta.

V `TypeaheadPresenter` je analogická úprava — pokud ES vrátí výsledky, jsou produkty načteny z databáze podle vrácených identifikátorů. Pokud ES selže, je použit původní SQL dotaz `productRepository->search()`.

3.4.5 Routování

Do `RouterFactory` jsou přidány tři routy mapující HTTP požadavky na endpointy klientského ApiPresenteru (viz Tabulka 4):

URL	Presenter
`/api/elastic/full-update`	ElasticsearchClient:Api:FullUpdate
`/api/elastic/process-queue`	ElasticsearchClient:Api:ProcessQueue
`/api/elastic/update-job-status`	ElasticsearchClient:Api:UpdateJobStatus

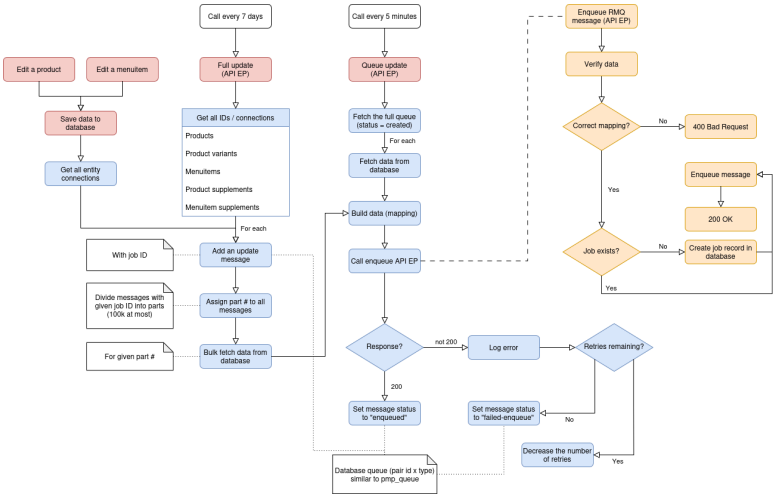
Tabulka 4: API routy pro Elasticsearch v e-shopu

3.4.6 Fallback strategie

Klíčovým principem implementace je **graceful degradace** — při jakékoliv chybě v komunikaci s Elasticsearch (nedostupnost serveru, chyba dotazu, timeout, nevalidní odpověď) systém automaticky přepne na původní SQL dotazy. Uživatel e-shopu tak nezaznamená výpadek funkčnosti, pouze potenciálně nižší rychlost odezvy. Tato strategie je implementována ve všech místech, kde se Elasticsearch používá — filtrování, vyhledávání i typeahead.

3.5 Workflow — zpracování dat

Přehled procesu indexace dat je znázorněn na Obrázek 2. Diagram zobrazuje interakce mezi vrstvami barevně odlišené podle architektury (červená = e-shop, modrá = client, oranžová = server). Detailní diagramy jednotlivých toků jsou v Příloze B.



Obrázek 2: Workflow — přehled procesu indexace dat

Systém obsahuje následující hlavní toky:

Vyhledávací tok (search flow) — uživatel zadá vyhledávací dotaz nebo aktivuje filtr v kategorii. E-shop prostřednictvím klientského balíčku odešle strukturovaný dotaz na server, který jej vykoná nad Elasticsearch indexem a vrátí výsledky s agregacemi. Pokud je Elasticsearch nedostupný, e-shop přepne na SQL dotazy.

Tok kompletní aktualizace (full update) — administrátor nebo cron spustí kompletní reindexaci. Klientský balíček vytvoří job, vloží všechny entity do fronty, rozdělí je do částí a postupně odesílá na server. Server zařadí data do RabbitMQ fronty, konzument je zpracuje a naindexuje. Po dokončení se atomicky přepne produkční alias.

Tok editace entity (edit product/menuitem) — při úpravě produktu v administraci e-shopu se spustí aktualizace konkrétní entity. Proces je analogický kompletní aktualizaci, avšak zahrnuje pouze jednu entitu a její doplňkové entity (varianty, vlastnosti, příslušenství).

Zpracování fronty (queue processing) — periodicky volaný endpoint zpracuje jednu připravenou část fronty — načte entity z databáze, transformuje je pomocí DataMapperu a odešle na server.

Monitoring jobů (job checking) — periodicky volaný endpoint kontroluje stav běžících jobů, finalizuje dokončené a označuje neúspěšné.

3.6 Nasazení

Elasticsearch server je nasazen na produkčním serveru nix01.vas-server.cz prostřednictvím Docker Compose. Konfigurace orchestruje pět služeb (viz Tabulka 5):

Služba	Popis	Port
Apache + PHP-FPM	Webový server s API endpointy	80/443
Elasticsearch 9.2	Vyhledávací engine (single-node, 7 GB heap)	9200
RabbitMQ	Message broker s administračním rozhraním	5672/15672
MariaDB	Databáze pro sledování jobů	3306
Consumer	Dlouho běžící RabbitMQ konzument	—

Tabulka 5: Služby v Docker Compose konfiguraci serveru

Elasticsearch běží v režimu single-node s přidělenou pamětí 7 GB pro JVM heap. Data jsou perzistentní díky Docker volume. RabbitMQ je konfigurován s trvalými frontami a persistentními zprávami, čímž je zajištěno přežití dat při restartu kontejneru.

Reverzní proxy Traefik zajišťuje HTTPS terminaci a routování požadavků na Apache kontejner. Health checky jednotlivých služeb zajišťují, že se závislé kontejnery spustí až po úspěšné inicializaci databáze a message brokeru.

Konfigurace prostředí (přihlašovací údaje k databázi, Elasticsearch a RabbitMQ) je řízena prostřednictvím `.env` souborů, přičemž lokální přepisy (`.env.local`) nejsou verzovány.

4 Výsledky

Výsledkem práce je funkční třívrstvý systém pro integraci Elasticsearch a RabbitMQ do e-shopových projektů, nasazený v produkčním prostředí.

Elasticsearch server poskytuje 5 API endpointů — vyhledávání, příjem dat k indexování, kontrolu stavu jobů, úklid osiřelých indexů a health check. Podporuje tři e-shopové projekty (Drevojas, Fencee, Dogtrace) s izolovanými indexy a projektově specifickými validačními schématy.

Elasticsearch client je Composer balíček s Nette DI integrací, který poskytuje kompletní rozhraní pro indexování a vyhledávání. Implementuje DataMapper pattern pro transformaci entit, partitioning velkých datasetů a automatický retry mechanismus při selhání odesílání.

Implementace do e-shopu Drevojas zahrnuje integraci do filtrování kategorií, full-textového vyhledávání a typeahead napovídání. Při nedostupnosti Elasticsearch systém automaticky přepne na SQL dotazy bez dopadu na funkčnost.

Systém je navržen pro snadnou rozšiřitelnost — implementace do nového e-shopu vyžaduje přibližně 12–22 hodin práce medior vývojáře (viz Příloha III), přičemž většina času je věnována konfiguraci filtrů a úpravě filtrovací komponenty e-shopu.

Měřením doby odezvy v produkčním prostředí bylo dosaženo následujících výsledků. Testy byly provedeny na produkčním serveru v testovacím režimu (mírně pomalejší kvůli Tracy logování a analýze).

Tabulka 6 zobrazuje celkovou dobu odezvy webové stránky — tedy čas od odeslání požadavku uživatelem po obdržení kompletní odpovědi.

Operace	**SQL**	**Elasticsearch**	**Zrychlení**
Full-text vyhledávání	7 800–7 900 ms	500–700 ms	92 %
Full-text vyhledávání s filtrováním	7 600–7 700 ms	400 ms	95 %
Typeahead (napovídání)	6 500–6 700 ms	400 ms	94 %

Tabulka 6: Celková doba odezvy webové stránky — SQL vs Elasticsearch

Tabulka 7 zobrazuje izolovaný čas potřebný pro získání dat z databáze/Elasticsearch. U Elasticsearch je rozlišen interní čas zpracování dotazu (parametr `took` v odpovědi ES) a celkový čas komunikace mezi e-shopem a ES serverem včetně síťové latence a režie.

Operace	**SQL dotaz**	**ES interní (took)**	**ES celkový (API)**
Full-text vyhledávání	7 500–7 600 ms	30 ms	120–150 ms
Full-text vyhledávání s filtrováním	7 400–7 600 ms	30 ms	140–170 ms
Typeahead (napovídání)	6 300–6 500 ms	10 ms	100–200 ms

Tabulka 7: Doba získání dat — SQL dotaz vs Elasticsearch

Uvedené hodnoty jsou průměrné doby odezvy při běžném provozu. Elasticsearch server aktuálně disponuje omezeným množstvím operační paměti oproti doporučené konfiguraci, tudíž je předpoklad dalšího zlepšení výkonu při navýšení prostředků.

5 Diskuse

5.1 Zdůvodnění technologických voleb

Elasticsearch vs Algolia / Meilisearch / Typesense — Elasticsearch byl zvolen pro svou flexibilitu při definování mappingů, pokročilý Query DSL umožňující složité vnořené dotazy a agregace, a otevřenost (self-hosted řešení bez závislosti na třetí straně). Cloudová řešení jako Algolia nabízejí jednodušší integraci, avšak za cenu měsíčního poplatku a omezených možností přizpůsobení dotazů. Meilisearch a Typesense jsou vhodné pro jednodušší vyhledávací scénáře, avšak neposkytují tak pokročilé agregace potřebné pro facetové filtrování v e-shopu [1].

RabbitMQ vs Redis queues — RabbitMQ byl zvolen pro svou nativní podporu AMQP protokolu, garantované doručení zpráv a robustní mechanismus potvrzení (acknowledgement). Redis-based fronty (např. Laravel Queues) jsou jednodušší na nastavení, avšak neposkytují stejnou úroveň spolehlivosti při vysoké zátěži a restartech serveru [2].

Nextras ORM vs Nette Database Explorer — pro klientský balíček a server byl zvolen Nextras ORM namísto Nette Database Explorer (používaného v e-shopu) z důvodu typové bezpečnosti entit, automatických validací a pohodlnější práce s vazbami [5].

5.2 Co se osvědčilo

Třívrstvá architektura umožňuje nezávislý vývoj a nasazení jednotlivých komponent. Server může být aktualizován bez dopadu na e-shopy a naopak.

Validator pattern na serveru zjednodušuje přidávání nových projektů — stačí vytvořit Validator třídu definující schéma dat a server automaticky generuje ES mapping a validuje příchozí data.

Fallback na SQL se ukázal jako klíčový pro produkční nasazení — eliminuje riziko výpadku e-shopu při problémech s Elasticsearch.

Partitioning velkých datasetů na části umožňuje spolehlivý přenos i při desítkách tisíc entit, aniž by hrozilo překročení paměťových limitů nebo timeoutů HTTP požadavků.

5.3 Omezení a prostor pro zlepšení

Webhook notifikace — při selhání konzumenta (zahozená zpráva po vyčerpání pokusů) systém pouze loguje chybu. Plánovaným rozšířením je implementace webhook notifikací pro okamžité upozornění administrátora.

Monitoring — systém nemá dedikovaný monitoring dashboard. Pro sledování stavu front a jobů je k dispozici pouze RabbitMQ administrační rozhraní a databázové dotazy.

Horizontální škálování konzumenta — aktuálně běží jeden konzument zpracovávající zprávy sekvenčně (prefetch_count=1). Při výrazném nárůstu objemu dat by bylo možné spustit více instancí konzumenta pro paralelní zpracování.

Real-time synchronizace — aktuální řešení vyžaduje explicitní spuštění aktualizace (cron nebo ruční volání). Alternativou by byla automatická synchronizace pomocí databázových triggerů nebo event systému.

6 Závěr

Cílem této práce bylo navrhnout a implementovat systém pro integraci technologií Elasticsearch a RabbitMQ do e-shopových projektů za účelem zrychlení full-textového vyhledávání a faceto-vého filtrování produktů. Tento cíl byl splněn — výsledkem je funkční třívrstvý systém nasazený v produkčním prostředí.

Systém sestává z centralizovaného Elasticsearch serveru s multi-tenantní podporou, univerzál-ního klientského Composer balíčku a implementace do cílového e-shopu. Server zajišťuje asynchronní indexování dat prostřednictvím RabbitMQ a zpracování vyhledávacích dotazů. Klientský balíček poskytuje Nette DI integraci, DataMapper pattern pro mapování entit a automatickou správu indexovacích jobů. Implementace do e-shopu zachovává SQL fallback pro případ nedostupnosti Elasticsearch.

Mezi hlavní přínosy patří zrychlení vyhledávání a filtrování o více než 90 % oproti původním SQL dotazům (viz Tabulka 6 a Tabulka 7), univerzálnost řešení (podpora více projektů z jednoho serveru) a snadná rozšiřitelnost do nových e-shopů. Architektura je navržena tak, aby minimalizovala dopad na stávající kód e-shopu a umožňovala postupné nasazování.

Mezi možná rozšíření do budoucna patří implementace webhook notifikací při selhání indexo-vání, monitoring dashboard pro sledování stavu systému, horizontální škálování konzumentu pro zvýšení propustnosti a automatická real-time synchronizace dat.

Literatura

1. Elastic N.V. *Elasticsearch Reference* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
2. Broadcom Inc. *RabbitMQ Documentation* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://www.rabbitmq.com/docs
3. RabbitMQ. *AMQP 0-9-1 Complete Reference Guide* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://www.rabbitmq.com/amqp-0-9-1-reference
4. Nette Foundation. *Nette Framework Documentation* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://doc.nette.org/cs/
5. Nextras. *Nextras ORM Documentation* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://nextras.org/orm/docs/5.0/
6. The PHP Group. *PHP Documentation* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://www.php.net/docs.php

7. Docker Inc. *Docker Documentation* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://docs.docker.com
8. MIRTES, Ondřej. *PHPStan Documentation* [online]. 2026 [cit. 2026-03-24]. Dostupné z: https://phpstan.org/user-guide/getting-started

Seznam obrázků

Seznam tabulek

Seznam příloh

Přílohy

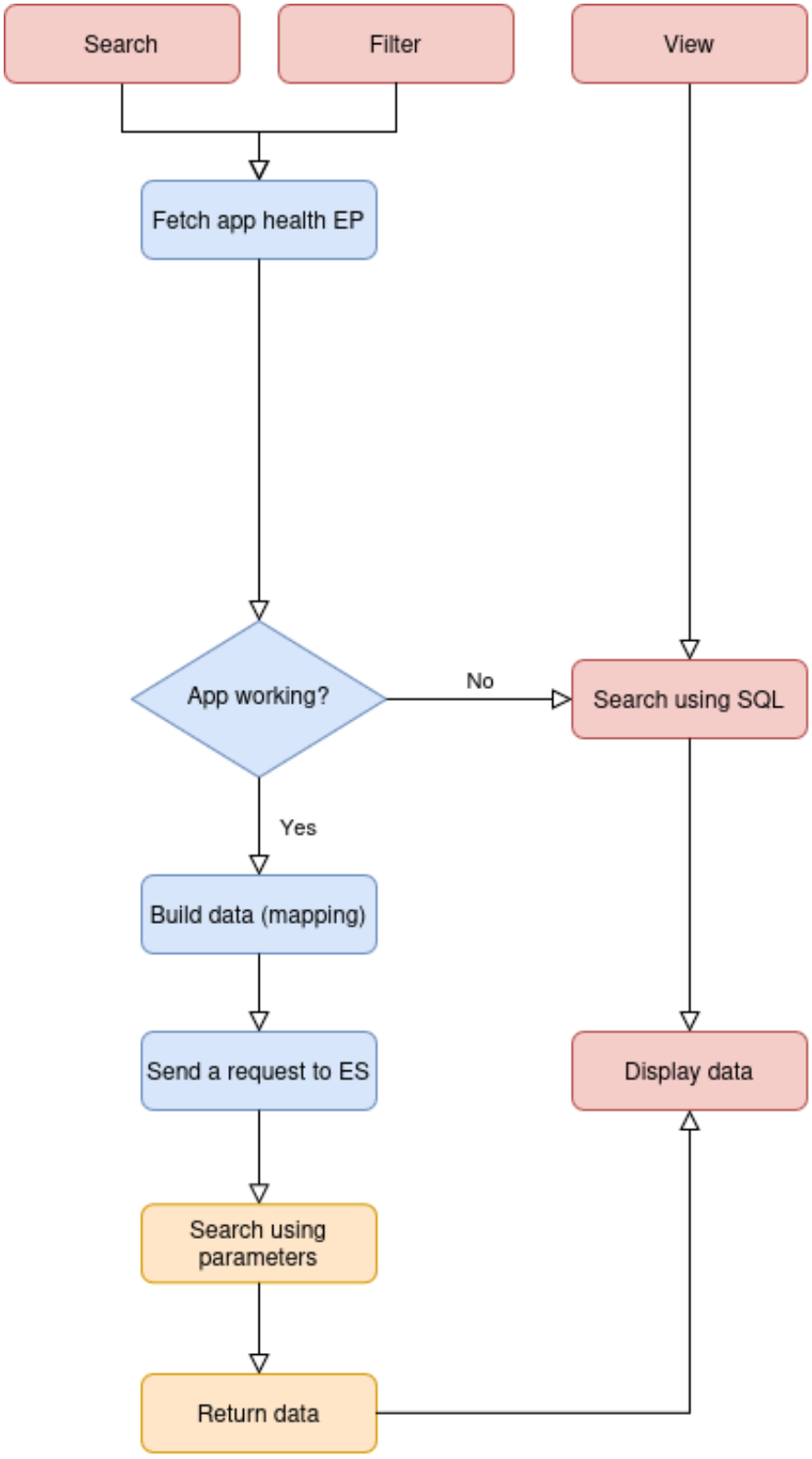
6.1 Příloha I: Use Case diagram systému

Use Case diagram znázorňuje interakce mezi aktéry (uživatel e-shopu, administrátor, cron) a jednotlivými vrstvami systému. Barevné odlišení odpovídá architektuře: červená = e-shop, modrá = client, oranžová = server.

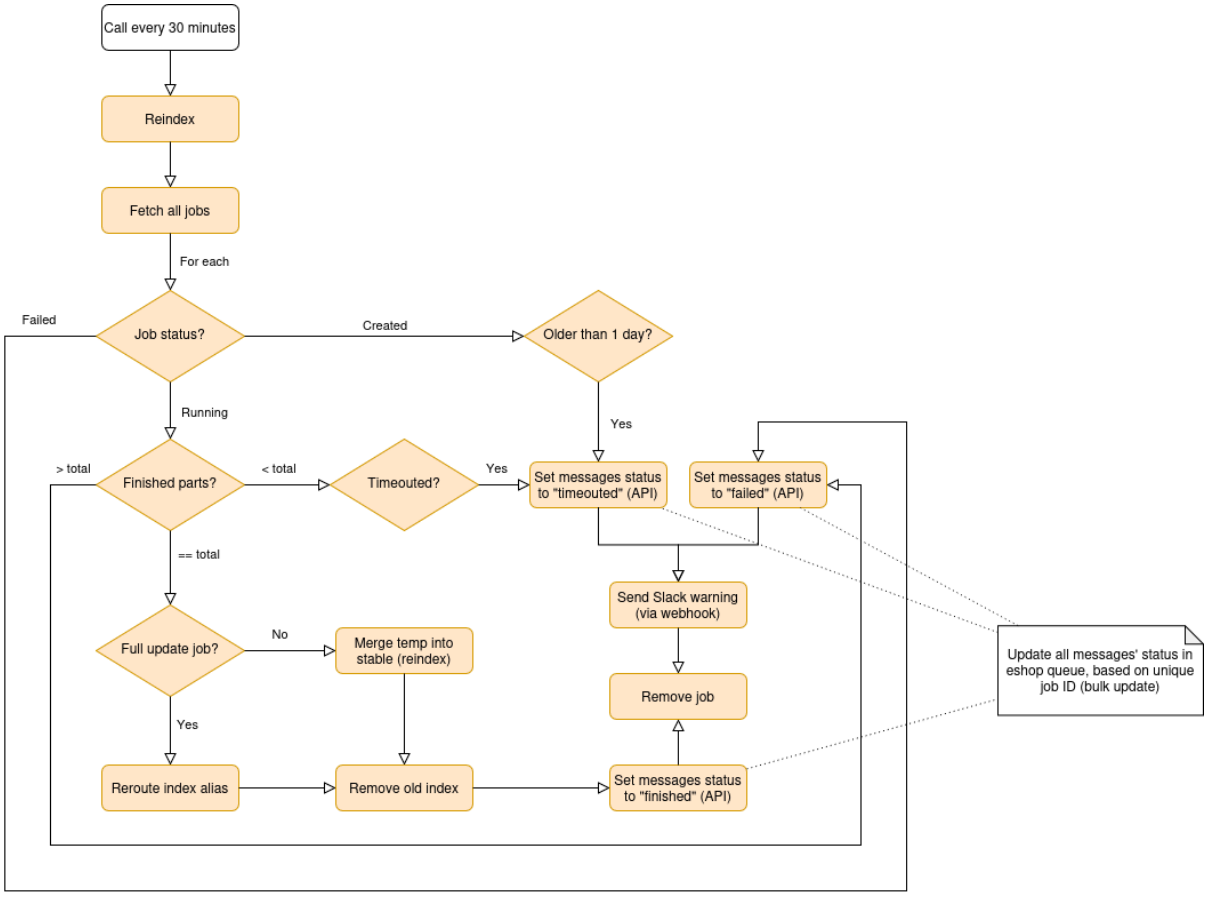
Obrázek 3: Use Case diagram — kompletní pohled

6.2 Příloha II: Workflow diagramy systému

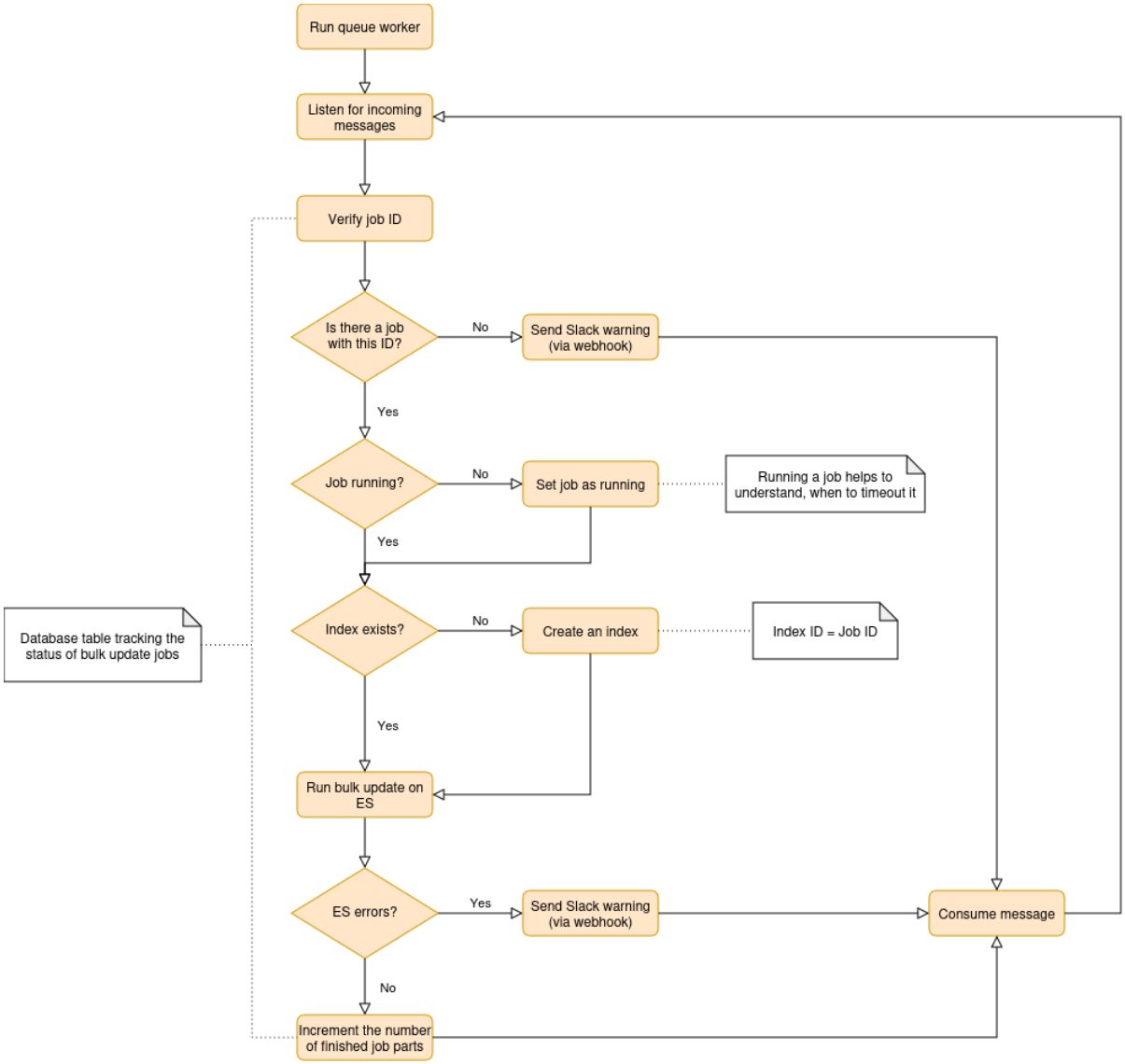
Workflow diagramy zobrazují detailní průběh zpracování dat v systému — vyhledávací tok, cron joby, zpracování fronty a přehled procesu indexace dat.



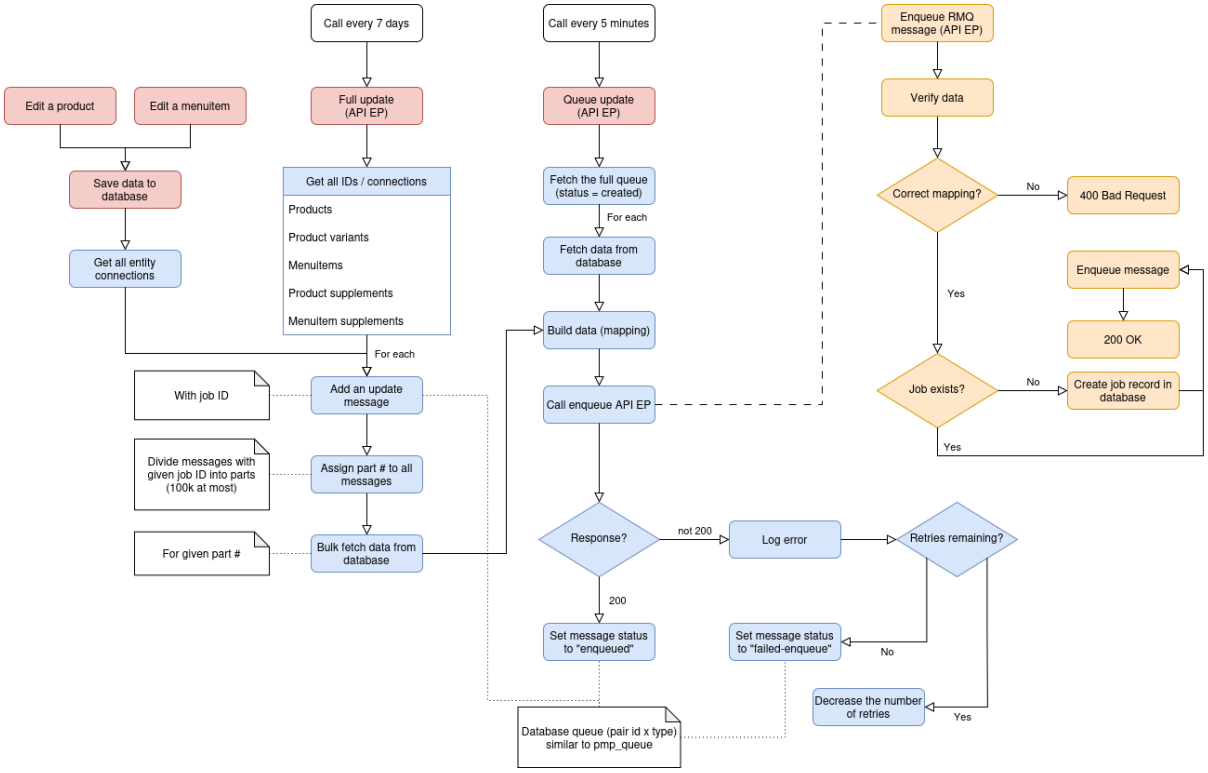
Obrázek 4: Workflow — vyhledávací tok (search / filter / view)



Obrázek 5: Workflow — cron joby a monitoring



Obrázek 6: Workflow — zpracování RabbitMQ fronty (queue consumer)



Obrázek 7: Workflow — přehled procesu indexace dat

6.3 Příloha III: Odhad implementace do nového e-shopu

Následující tabulka shrnuje časový odhad implementace Elasticsearch do nového e-shopového projektu (profil: medior vývojář se znalostí Nette, ES a struktury klientského/serverového projektu).

Úkol	**Odhad**
Server	
Přidání enum case projektu	10 min
Validator + DTO soubory	30–60 min
Migrace — enum v tabulce jobs	10 min
Client	
DataMapper + úpravy entit	1–1,5 h
E-shop	
Docker compose + Composer import	30 min–1 h
Migrace — tracking tabulky	15–30 min
Konfigurace config.neon	1–2 h
Routy v RouterFactory	10 min
TypeaheadPresenter	30 min–1 h

Úkol	**Odhad**
HomepagePresenter + Search trait	2–3 h
Filter.php — ES integrace	3–5 h
Průřezové	
Testování a debugging	2–4 h
Orientace v novém e-shopu	1–2 h
Celkem	**12,5–22 h (16 h)**

Tabulka 8: Časový odhad implementace do nového e-shopu