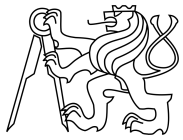
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

| | |
|---|---|
| **Název:** | InfoWeb - Nástroj získávání informací z webů |
| **Student:** | Jakub Tuček |
| **Vedoucí:** | Ing. Jiří Hunka |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2017/18 |

## Pokyny pro vypracování

V předmětech BI-SP1 a BI-SP2 byl realizován týmový projekt pro získávání informací z webů s primárním zaměřením na potřeby internetových eshopů.

- Obecně rozeberte problematiku získávání informací z webů s primárním zaměřením na potřeby internetových obchodů.
- Analyzujte a zhodnoťte současný stav projektu včetně korektnosti zvolených postupů a řešení s ohledem na požadavky internetových obchodů.
- Na základě analýzy navrhněte potřebná vylepšení stěžejních částí aktuální aplikace.
- Nejvíce potřebná vylepšení implementujte a funkčnost řádně otestujte.
- Zajistěte potřebnou infrastrukturu pro snadný a stabilní rozvoj projektu.
- Zhodnoťte finální stav celého projektu po Vašich úpravách.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 28. listopadu 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra ... (softwarového inženýrství)

Bakalářská práce

# InfoWeb - Nástroj získávání informací z webů

Vedoucí práce: Ing. Jiří Hunka

15. května 2017

# Poděkování

Rád bych poděkoval za trpělivost vedoucímu Ing. Jiřímu Hunkovi, rodině za podporu a svému týmu z předmětů BI-SP1 a BI-SP2 za obětavou práci na společném projektu.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 .....

### Odkaz na tuto práci

Tuček, Jakub. *InfoWeb - Nástroj získávání informací z webů* . Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

# Abstrakt

Tato práce rozebírá problematiku získávání informací z webů s důrazem na potřeby internetových obchodů jako například vývoj ceny produktu u konkurence. Jelikož této práci předcházel týmový projekt z předmětů BI-SP1 a BI-SP2 je popsaná i daná týmová realizace včetně zvolených postupů. Po zhodnocení týmového řešení jsou navrženy možné změny, které doplňují funkcionalitu, umožňují lepší rozšiřitelnost a opravují nežádoucí chování systému. Výsledkem práce je refaktoring týmového řešení spolu s rozšířením o nejdůležitější vylepšení. Vzniklý systém je na základě důkladného otestování znova zhodnocen. Finálně jsou navržena další vylepšení pro budoucnost projektu.

**Klíčová slova** informace z webu, internetové obchody, cena produktu, Git, Jenkins, průběžná integrace, modulární architektura

# Abstract

This thesis describes difficulties of data mining from web with emphasis on the needs of online shops. Such need is for example trend of product prices sold by competitors. Because this issue was already addressed in team project implemented in courses BI-SP1 and BI-SP2, thesis describes created system, including chosen work methods. After analysis of created team project,

possible changes are designed. These changes are extending existing functionality, improving expandability and fixing unwanted behaviour of created system. Goal of thesis is refactoring of team project along with implementing most important changes. Created system is evaluated based on thorough testing. Finally, additional improvements are designed for possible future system usage.

**Keywords** data mining, online shopping, product price, Git, Jenkins, Continuous Integration, Modular Architecture

# Obsah

**Úvod** **1**

**1 Popis problematiky získávání informací z webů** **3**
1.1 Problematika . . . . . 3
1.2 Výběr dat . . . . . 3
1.3 XML Path Language . . . . . 4
1.4 CSS Selector . . . . . 4
1.5 Současný stav řešení potřeb internetových obchodů . . . . . 5
1.6 Popis konkrétních existujících služeb . . . . . 6

**2 Analýza týmového projektu** **13**
2.1 Cíl týmového projektu . . . . . 13
2.2 Požadovaná funkcionalita . . . . . 13
2.3 Návrh . . . . . 14
2.4 Webové rozhraní . . . . . 14
2.5 Interní část . . . . . 14

**3 Vývoj a implementace týmového projektu** **17**
3.1 Vývoj . . . . . 17
3.2 Implementace . . . . . 18
3.3 Má role . . . . . 19

**4 Zhodnocení týmového projektu** **21**
4.1 Způsob hodnocení . . . . . 21
4.2 Stav . . . . . 22
4.3 Nedostatky . . . . . 22
4.4 Shrnutí . . . . . 26

**5 Návrhy na vylepšení** **31**
5.1 Refaktorování stávajícího řešení . . . . . 31

5.2 Plánování práce . . . 32
5.3 Oprava komunikace Manager - ProductProvider . . . 32
5.4 Spojení chyb analyzátoru a vylepšení rozhraní . . . 33
5.5 Monitorování . . . 33
5.6 Získání adres obchodů a příslušných detailů produktů . . . 34
5.7 Párování produktu . . . 34
5.8 Neimplementované návrhy . . . 35

**6 Realizace vylepšení** **37**
6.1 Refaktorování stávajícího řešení . . . 37
6.2 Plánování práce . . . 44
6.3 Oprava komunikace Manager - ProductProvider . . . 44
6.4 Spojení chyb analyzátoru a vylepšení rozhraní . . . 47
6.5 Monitorování . . . 48
6.6 Získání adres obchodů a příslušných detailů produktů . . . 48
6.7 Párování produktu . . . 50
6.8 Detekce neexistující stránky a nenalezeného produktu . . . 51
6.9 Více šablon detailů produktů . . . 51
6.10 Více stejných chyb . . . 51
6.11 Skladem . . . 53
6.12 Ostatní . . . 53

**7 Zhodnocení provedených vylepšení** **55**
7.1 Data . . . 55
7.2 Funkcionalita . . . 55
7.3 Dodatečné opravy . . . 57
7.4 Párování . . . 57
7.5 Webové rozhraní . . . 58
7.6 Návrh a testy . . . 58
7.7 Rozhraní administrátora . . . 59
7.8 Nemožnost vyhledávání na některých obchodech . . . 59
7.9 Shrnutí . . . 59

**8 Závěr** **61**

**Literatura** **63**

**A Seznam pojmů** **67**
A.1 Web API . . . 67
A.2 Verzovací systém Git . . . 67
A.3 Jednotkové a integrační testy . . . 67
A.4 Statická analýza kódu . . . 68
A.5 Průběžná integrace . . . 68
A.6 Sdílení dat pomocí front . . . 68

A.7 JSON . . . . . 69
A.8 Mock . . . . . 69
A.9 Refaktorování kódu . . . . . 69

**B Seznam použitých zkratek** **71**

**C Obsah přiloženého CD** **73**

# Seznam obrázků

1.1 Price checking . . . . . 8

3.1 MVC . . . . . 18

4.1 Diagram zobrazující vytvoření kampaně a nalezení nových dat v původním řešení týmového projektu . . . . . 23
4.2 Pokrytí testy vytvořeného řešení . . . . . 26

6.1 Diagram tříd analyzátoru . . . . . 43
6.2 Webové rozhraní pro vyřešení chyby analyzátoru po provedení vylepšeních . . . . . 49
6.3 Aktivity diagram ilustrující nalezení adres detailů produktů . . . . 50
6.4 Administrátorské rozhraní pro opravu detailu šablony . . . . . 52

7.1 Pokrytí testy po provedených vylepšení . . . . . 58

A.1 Větve v Git repozitáři . . . . . 68

# Seznam tabulek

4.1 Data použitá k otestování řešení týmového projektu. . . . . 22
4.2 První část přehledu nedostatků vytvořeného týmového řešení. . . . 28
4.3 Druhá část přehledu nedostatků vytvořeného týmového řešení. . . 29

7.1 Produkty použité v testovacích datech pro zhodnocení výsledného řešení. . . . . 56
7.2 Obchody použité v testovacích datech pro zhodnocení výsledného řešení. . . . . 56
7.3 První část přehledu nedostatků po implementaci navržených vylepšení. . . . . 60
7.4 Druhá část přehledu nedostatků po implementaci navržených vylepšení. . . . . 60

# Úvod

Práce uvádí čtenáře do problematiky získávání informací z webů s důrazem na požadavky internetových obchodů popřípadě distributorů zboží, konkrétně sledováním vývoje cen prodávaných produktů. Je popsán současný stav řešení potřeb internetových obchodů a to včetně existujících služeb.

V předmětech BI-SP1 a BI-SP2 v prostředí FIT ČVUT byl již výše popsaný projekt realizován. Týmový projekt je proto zanalyzován na základě popisu domény, kterou má projekt za úkol řešit a základního návrhu systému z hlediska interní a webové části. Dále jsou uvedeny postupy použité při vývoji, samotná implementace a finální zhodnocení vytvořeného řešení.

Cílem práce je provést analýzu týmového projektu, navrhnout vylepšení, nejdůležitější implementovat a výsledné řešení opět zhodnotit. Důraz je kladen především na budoucí rozšiřitelnost a opravu či doplnění stávající funkcionality pro nezbytné použití systému. Zhodnocení je provedeno na základě testování nad reálnými daty a odráží také snadnost rozšiřitelnosti, která byla ověřena v průběhu pozdější části implementace.

Kapitola 1

# Popis problematiky získávání informací z webů

V této kapitole se budu zabývat samotnou problematikou získávání informací z webů s důrazem na internetové obchody. Jelikož je tato problematika již řešena existujícími službami, existující služby zhodnotím.

## 1.1 Problematika

Získávání informací z webů je efektivní možnost jak získat databázi informací, které se na internetu vyskytují. Tato činnost však stojí na problematice data získávat a uchovávat v potřebné struktuře, protože jinak z dat nejsme schopni vyčíst potřebné informace. Vzhledem k specificitě dat, která jsou v kontextu činnosti zajímavá a kvůli unikátnosti webových stránek není možné jednoznačně určit jednotný a zcela automatizovaný postup získávání dat v požadovaném formátu.

## 1.2 Výběr dat

Možné řešení, jak získávat strukturované informace z webů je kombinace automatizace a prvku lidské inteligence. Což je obvykle dosaženo roboty, kteří data stahují a lidské práce určující jaké informace jsou ve stažených datech zajímavé.

Získávání informací ze stažených stránek lze zjednodušit na problematiku určení elementů v HTML. Element pak může obsahovat pouze požadovanou informaci, například cenu produktu. Lokaci lze jednoznačně určit mimo jiné pomocí těchto dvou možností:

1. XPath,
2. CSS Selector.

## 1.3 XML Path Language

XML Path Language[1] nazývaný zkráceně XPath je jazyk sloužící k výběru elementu v XML[2] dokumentu.

XML chápeme jako jazyk popisující strukturu strojově i lidsky čitelných dat. HTML lze vzhledem ke struktuře chápat jako formát podobný XML, ačkoliv se přímo o XML nejedná[3]. Popisuje způsob zobrazení dat ve formátu, které prohlížeče rozumí. Díky podobnosti s XML je však možné XPath použít pro definování cesty k prvku, který uchovává potřebnou informaci na webové stránce.

## 1.4 CSS Selector

Jazyk CSS je používán pro vizuální popis prezentace webové stránky v HTML. K určení prvků se kterými pracuje používá selektory, které označují konkrétní prvek v HTML, buď pomocí samotného názvu elementu, přiřazené třídy nebo nastaveného identifikátoru.[4]

Selektor nemusí vybírat pouze jeden element, nicméně zřetězením selektorů je možné jedinečného výběru snadno docílit.

## 1.5 Současný stav řešení potřeb internetových obchodů

I v kontextu malého trhu jako je Česká republika, se lze bavit o velké konkurenci na poli maloobchodů prodávající své zboží na internetu. Internetové obchody potřebují monitorovat nejen konkrétní konkurenci, ale i trh. Vzhledem k jejich zaměření je nejvíce zajímají obchody prodávající stejné zboží.

Potřebné informace o prodávaných produktech konkurencí se skládají z následujících atributů:

1. název,
2. model,
3. EAN,
4. cena,
5. inzerovaný název,
6. dostupnost.

S těmito daty je možné dále pracovat, například při analýze konkurenceschopnosti nebo za jaké ceny jsou produkty prodávané jednotlivými prodejci, což je informace zajímavá především pro distributory zboží.[5]

### 1.5.1 Srovnávače cen

Data lze získat pomocí srovnávačů cen jako jsou *zbozi.cz*[6] nebo *heureka.cz*[7]. Problém u těchto služeb spočívá v orientaci na koncové zákazníky, kterým umožňuje nalezení nejlepší ceny na trhu pro hledaný produkt. Bohužel tím narážím na skutečnost, že největší srovnávače cen neposkytují veřejně svá data, případně neexistuje možnost, jak je jednoduše získat.

V rámci výzkumu pro bakalářskou práci jsem měl možnost nahlédnout do dat, které *heuréka* poskytuje některým obchodům.[5] Data obsahují následující informace:

- informace o produktu (Segment, Kategorie, Jméno, ID, Výrobce, EAN, Item ID),
- URL na vlastním obchodu,
- URL na Heuréce,
- počet konkurence a popularita na trhu,
- vlastní cena a pozice dle ceny,

- deset nejvyšších a nejnižších cen.

První zásadní nedostatek zprávy z jmenovaného srovnávače se ukázal být logistický a to, že obchod musí být označen „Ověřeno zákazníky“, aby měl provozovatel obchodu k datům přístup. Další nedostatek jsou data neobsahující konkrétní označení konkurenčních obchodů.[8] Vzhledem k povaze struktury a splatnosti generovaných dat je nemožné ceny sledovat v časovém období. Ostatní srovnávače mají výstup velmi podobný nebo konkrétní data vůbec neposkytují. Díky tomu se srovnávače ukázaly jako nedostatečný zdroj dat.[5]

### 1.5.2 Existující služby

Problematiku sledování trhu s důrazem na firemní klientelu, řeší aktuálně několik existujících služeb.

Služby mají v zásadě velmi podobnou povahu poskytovaných možností. Rámcově se jedná o porovnávání cen včetně historie na různých internetových obchodech či na srovnávačích. Uživatel si zadá okruh či seznam produktů, buďto manuální formou či vstupem ze souboru. Některé služby umožňují přímé napojení na internetový obchod. Po různě dlouhé prodlevě je možné data zobrazit v grafech označující vývoj cen, trendů či náhlých změn. Všechny služby umožňují výstup sledování do souboru.

Největší rozdíl ve službách je, zda jsou data získávána přímo z obchodů nebo ze srovnávačů. Další odlišnost je schopnost sledovat i zahraniční trh.

Ceny služeb se obvykle odvíjí od počtu sledovaných produktů a četnosti aktualizací. Proto se měsíční platby mohou pohybovat od stovek korun po desítek tisíc korun.

## 1.6 Popis konkrétních existujících služeb

### 1.6.1 Price checking[9]

**Hlavní funkce**

- porovnává a vyhledává ceny zadaných výrobků v reálném čase,
- sleduje dostupnost produktů,
- automatické stahování dat v intervalech,
- statistické pohledy, nahlížení do historie,
- generování grafů,
- cenotvorba.

**Vstup**

- souhrn produktů určený pro sledování,
- libovolný formát, například xsl nebo xml,
- možný manuální vstup.

**Výstup**

- libovolný formát, například xsl nebo xml,
- webové rozhraní.

**Prostředí**

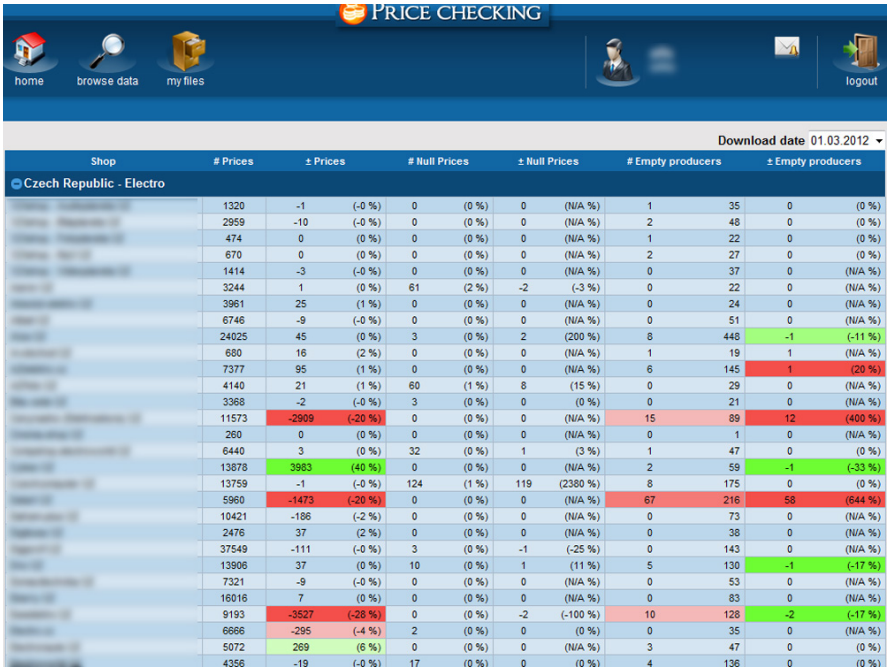
- webové rozhraní.

**Data**

- přes 250 výrobců, 300 obchodů a 1 200 000 výrobků,
- český, slovenský, polský, slovinský, německý a maďarský trh,
- aktualizace denně, maximálně 144 krát za den,
- počet sledovaných obchodů je fixní, lze však přidat na požádání,
- převážně elektronika, bílé zboží, pneumatiky a hračky.

**Cena**

- 6000 - 85 000 Kč (bez DPH) za licenci měsíčně,
- minimální doba smlouvy 12 měsíců.




Obrázek 1.1: Ukázka služby Price checking

### 1.6.2 Pricing intelligence[10]

**Hlavní funkce**

- monitorování a srovnávání cen konkurence, vývoj cen a trendů v čase,
- přehledné výpisy výsledků,
- u většiny cenových nabídek nutno definovat počet konkurentů,
- upozornění na změny cen v čase.

**Výstup**

- formát xsl nebo pdf.

**Prostředí**

- webové rozhraní.

**Data**

- nespecifikované data a zaměřený trh.

**Cena**

- 599 až 4999 Kč měsíčně,

- minimálně tři měsíce,
- neomezené sledování produktů a konkurentů je možné pouze s nejvyšším tarifem a po individuální ceně.

### 1.6.3 Sledování trhu[11]

**Hlavní funkce**

- sledování cen, pozic, dostupnosti a hodnocení na porovnávačích zboží i jednotlivých obchodech,
- uchování historie,
- možné napojení přímo na vlastní internetový obchod,
- notifikace změn,
- možnost více účtů s oddělenými přístupy,
- cenotvorba,
- detekce cenových spirál (kdo první zlevnil a následující dopady).

**Vstup**

- xml, xsl nebo manuálně.

**Výstup**

- xsl nebo webový.

**Prostředí**

- webové rozhraní.

**Data**

- srovnávače cen: heureka.cz, zbozi.cz, najnakup.sk, pricemania.sk, ceneo.pl, nokaut.pl, argep.hu, preisroboter.de,
- přímé sledování na obchodu,
- z toho plyne záběr na český, slovenský, německý a maďarský trh,
- aktualizace až několikrát denně.

**Cena**

- platba za každé vyhledání,
- individuální cena.

### 1.6.4 Pricebot[12]

Web je datován roku 2015, avšak popis funkcí není dokončený. Obsahuje výplňový text, proto je popis funkcí nekompletní.

**Hlavní funkce**

- denní monitoring cen na heureka.cz,
- možnost sledovat produkty konkurence,
- poskytuje pravidelný výsledek nalezených cen a vizualizaci změn,
- notifikace o změnách,
- notifikace o konkurentech prodávajících za nižší cenu,
- maximum lze sledovat 600 produktů ,
- maximum sledovaných konkurentů je 70.

**Vstup**

- produkty ke sledování.

**Výstup**

- pdf na email.

**Prostředí**

- webové rozhraní.

**Data**

- srovnávač cen Heureka.cz.

**Cena**

- dle počtů produktů,
- od 299 do 1299 Kč.

### 1.6.5 Zahraniční nástroje

Tyto nástroje jsou obecněji zaměřené a obvykle požadují od uživatele technické znalosti, jelikož je nutné přesně specifikovat kde, co a jak je požadováno sledovat. Vzhledem k tomuto omezení není možné použití přímo provozovateli e-shopů, jelikož těmito znalostmi z povahy práce obvykle nedisponují.

Příklad zahraničních nástrojů:

1. Screen scraper[13]
   - webová služba,
   - procházení web skrz odkazy,
   - potvrzování formulářů,
   - využití interního vyhledávání,
   - export do širokého množství formátu souborů,
   - cena: $549 - $2,799 za měsíc.
2. Web extractor[14]
   - Windows Aplikace,
   - procházení zadaných stránek,
   - hledání stránek pomocí klíčových slov,
   - export do csv formátu,
   - cena: $99 - $199 jednorázově.

Kapitola 2

# Analýza týmového projektu

V této kapitole se budu věnovat řešení vytvořeného v rámci předmětů BI-SP1 a BI-SP2 na ČVUT FIT v akademickém roce 2015/16. Popíšu cíl, který měl projekt za úkol řešit a jaká měla být výsledná funkcionalita řešení. Dále také vysvětlím základní strukturu navrženého systému.

## 2.1 Cíl týmového projektu

V předmětech BI-SP1 a BI-SP2 byl realizován týmový projekt. V souladu s osnovami byl BI-SP1 vytvořen návrh, který se v BI-SP2 následně implementoval.

Cílem tohoto projektu byla maximální možná míra automatizace získávání informací o produktech prodávaných konkurencí. Důraz byl především kladem na optimalizaci počtu nutných lidských úkonů. Navržený způsob spoléhal v nezbytných krocích na administrátora, u kterého se předpokládala technická zdatnost průměrného uživatele.

## 2.2 Požadovaná funkcionalita

Požadovaný stav projektu umožňuje uživateli vložit produkty do systému ve formátu *cvs* či *xlsx*, poté pomocí rozhraní definovat význam jednotlivých sloupců v tomto dokumentu a zvolit požadovanou frekvenci sledování dat.

Systém na základě dat vyhledá obchody, které prodávají vložené produkty. Z nich v definovaných intervalech získává data, ze kterých je vytvořen výstup pro uživatele obsahující především informace o cenách. Výstup lze vizualizovat i na grafech ve webovém rozhraní nebo stáhnout ve formátu *csv* či *xlsx*.

Proces samotného hledání byl navržen jako soubor více kroků, skládající se z procesů interních částí a interakcí administrátora, který zajišťuje řešení problémů, které systém nedokáže vyřešit.

## 2.3 Návrh

Řešení bylo rozděleno na **část webového rozhraní** a na část zpracovávající interní procesy, nazývanou v této práci jako **interní část**. Vzhledem k požadavkům na škálovatelnost aplikace se interní část skládá z více samostatných menších služeb - modulů komunikující spolu pomocí front. Díky tomu, že každý modul zajišťuje určitou funkcionalitu, je možné vytvářet více jejich instancí. Procesy lze zpracovávat paralelně a na více serverech, kde je jediné kritérium připojení na systém zajišťující komunikaci.

Uživatelská a interní část spolu sdílejí data pomocí relační databáze[15].

## 2.4 Webové rozhraní

### 2.4.1 Uživatelská část

Uživatelská část obsahuje množinu podstránek určených pro koncové uživatele služby, klienty.

Uživatelská část umožňuje vytvořit kampaň. Kampaň je proces trvající určitý časový úsek, který sleduje vložené produkty na konkurenčních obchodech. V rámci běžící kampaně má poté uživatel možnost vidět vizualizaci získaných dat, případně je umožněn export dat do formátu *csv* či *xlsx*. Ze zobrazených dat lze zjistit, na kterých webových stránkách je produkt prodáván a za jakou cenu.

### 2.4.2 Část pro administrátory

Pro přístup do části pro administrátory je nutné, aby měl uživatel speciální práva. Běžný uživatel, tak k této části nemá přístup. Slouží k monitorování kampaní uživatelů a řešení problémů, které systém není schopný automaticky vyřešit. Tím je myšleno definování selektorů pro výběr dat z webových stránek, párování produktu ke stránce nebo potvrzení zda jsou získaná data validní.

## 2.5 Interní část

Interní část je rozdělena do samostatných modulů, které spolu komunikují pomocí front. Moduly je možné spustit jako služby ve více instancích, kromě modulu Manager. Vzhledem k možnostem front, lze také práci distribuovat na více serverů, aniž by byla ohrožena bezpečnost databáze, protože k ní je možný umožnit pouze lokální přístup. Moduly jsou detailněji popsány v následujících podsekcích.

### 2.5.1 Manager

Manager je hlavní modul, který má jako jediný možnost přímého připojení do databáze. Jeho běžící instance může existovat pouze jednou. Manager má za úkol plánování práce pro ostatní části systému a samotnou správu komunikace s ostatními moduly. Práce je delegována pomocí *požadavků*, které jsou odeslány pomocí front jednotlivým modulům. *Odpovědi* a *chyby* reprezentují výsledky.

### 2.5.2 Finder

Modul Finder získává URL adresy internetových obchodů, které prodávají požadované produkty. Na nalezeném obchodě poté vyhledává adresy vedoucí na detaily produktů. K tomu je použito interní vyhledávání, které obchod poskytuje svým zákazníkům. Získané adresy detailů pak obsahují podrobné informace prodávaných produktů.

### 2.5.3 DataProvider

DataProvider je modul, který zpracovává adresy vedoucí na detaily produktů. Proces modulu reprezentuje následující seznam, kdy každý hlavní bod může skončit uvedenou chybou. Výsledek je odeslán ke zpracování Managerem.

1. Stažení stránky.
    - Stažení selhalo.
2. Vyparsování dat pomocí šablony.
    - Šablona neexistuje nebo je chybná.
3. Analýza dat vůči historickým datům (pokud existují).
    - Data jsou nevalidní.

Kapitola 3

# Vývoj a implementace týmového projektu

V této kapitole se věnuji průběhu vývoje týmového projektu a vytvořenému řešení. Popíši zvolené postupy při vývoji a jaké technologie byly vybrány.

Poslední část rozebírá mou roli v tomto projektu, protože téma bakalářské práce jsem měl již předběžně vybrané na začátku předmětu BI-SP2.

## 3.1 Vývoj

Vývoj byl rozdělen do 5 iterací, z nichž každá obsahovala 10 sprintů. V každé iteraci bylo definováno jakou musí obsahovat funkcionalitu, která bude na konci iterace prezentována vyučujícímu. Funkcionalita se skládala z jednotlivých úkolů rozložených do sprintů.

Úkoly byly přidělovány jednotlivým členům týmu. Samotné úkoly uchovával systém Redmine[16] a umožňoval sledovat jejich stav. Úkoly bylo možné v Redmine přiřadit k jednotlivým sprintům a iteracím, což umožňovalo přehled o plnění časového plánu.

Jako verzovací systém byl zvolen systém Git se vzdáleným repozitářem uložený na službě Gitlab[17]. Gitlab poskytuje webové rozhraní pro snadnou správu a možnost spouštění služeb na základě definovaných aktivit v repozitáři. Repozitář se skládal ze 4 částí (větví):

- Master - hlavní větev uchovávající verze určené k nasazení na produkční server.
- Develop - vývojová větev obsahující aktuální stav vývoje.
- Feature - vedlejší větev vytvořená pro konkrétní úkol přidávající novou funkcionalitu.
- Fix - vedlejší větev určená pro úkoly opravující chybu.

Protože práva k modifikaci větví Master a Develop měl pouze vedoucí projektu, musel být pro každou Feature a Fix větev vytvořen požadavek o zařazení (Merge request). Až po kontrole vedoucím byl požadavek zařazen nebo vrácen k opravě.

Na konci každé iterace byla poslední verze označená pomocí *tagu* a poté prezentována vedoucímu. Označení bylo zvoleno na základě pořadí iterace. První iterace je označena verzí „0.1“.

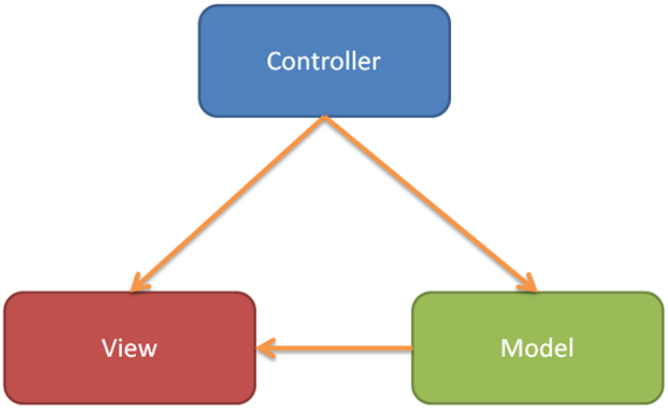
Pro vývoj se využil princip průběžné integrace. Každá verze byla zkompilována, otestována a zanalyzována na vzdáleném serveru. Tyto činnosti zajišťovaly systémy Jenkins[18] a Gitlab. Po změně v repozitáři byl spuštěn úkol v Jenkins. Ten aplikaci sestavil, spustil testy a statickou analýzu kódu zajištěnou systémem SonarQube[19]. Výsledky publikoval ve svém webovém rozhraní a zároveň v rozhraní Gitlab.

## 3.2 Implementace

### 3.2.1 Webové rozhraní

Webové rozhraní je implementováno v jazyce PHP verze 7. Základem aplikace je aplikační rámec Nette[20]. Nette obsahuje nástroje pro automatickou správu závislostí, komunikaci s databází, vytváření bezpečných formulářů, zabezpečení aplikace, šablonovací systém a rozhraní pro tvorbu testů.

Nette je navrženo s myšlenkou použití MVC architektury, která odděluje prezenční a logickou vrstvu. Zkratka MVC značí Model-View-Controller. V případě webového projektu v Nette představují *view* vrstvu šablony, definující vzhled webových stránek. *Controller* vrstva se skládá z presenter tříd obsluhující šablony. *Modelovou* vrstvu zajišťují třídy servisní, vykonávající logické části aplikace jako například práci s *repository* třídami nebo zpracování formulářů. *Repository* se starají o přímou komunikaci s databází.




Obrázek 3.1: Vizualizace návrhu MVC (Model-View-Controller)

Snadnou správu závislostí nad externími knihovnami zajišťuje balíčkovací systém Composer[21]. Na základě souboru definující potřebné knihovny a jejich verze jsou staženy pomocí jednoho příkazu z centrálního repozitáře. To zajišťuje jednotné verze a eliminaci nutnosti knihovny manuálně stahovat či přidávat přímo do repozitáře.

### 3.2.2 Interní část

Interní část je implementována v jazyce Java verze 8. Sestavení, spouštění testů a správu závislostí zajišťuje Gradle[22]. Umožňuje automatické stažení knihoven. Standardně je nastavený jako zdroj centrální maven repozitář.[23] Maven repozitář uchovává většinu volně přístupných knihoven, v tomto případě všechny, které jsou v rámci tohoto projektu použity.

V rámci sestavení lze pustit testy a další definované úkoly jako například tvorba dokumentace. Projekt používá doplněk Cobertura[24], který na základě spuštěné testovací sady vytváří zprávu obsahující pokrytí větví programu. Díky tomu je možné jednoduše zjistit jaké větve aplikace nejsou otestované.

Aplikace je rozdělena do nezávislých modulů běžící jako služby. Jednotlivé moduly spolu komunikují pomocí posílání zpráv v definovaných frontách. Komunikaci zajišťuje systém RabbitMQ Server[25] implementovaný v jazyce Erlang. Zprávy jsou serializovatelné objekty, jejichž definice je sdílena napříč všemi moduly.

*Serializace* představuje proces, kdy je objekt serializovaný do posloupnosti bitů, které jsou poslány jako zpráva. Vzhledem ke sdílené podobě objektu na obou stranách, lze zprávu jednoznačně deserializovat zpět do původního Java objektu se kterým je možné dále pracovat.[26]

Projekt využívá mnoho volně dostupných knihoven, nejpodstatnější jsou však následující:

- Google Guice - automatická správa závislostí.[27]
- Hibernate - objektově relační zobrazení databázových entit a práce s nimi.[28]
- Apache Commons - pomocné knihovny pro práci s řetězcemi a soubory.[29]
- RabbitMQ - rozhraní pro komunikaci s frontami.[25]

## 3.3 Má role

V druhé části týmového projektu, samotné implementaci, jsem byl vedoucí týmu. Jelikož jsem již měl téma své bakalářské práce vybrané, věnoval, jsem se projektu nad rámec předmětu. Kromě povinností vedoucího, které se skládaly z plánování práce a kontroly vytvořené implementace jsem se věnoval návrhu,

který bylo třeba v průběhu semestru pozměnit, jelikož návrh z předmětu BI-SP1 nebyl dostatečný. Jednalo například o navržení modulu Manager, který byl navržen pouze jako black-box.

Na začátku projektu jsem vytvořil celý ekosystém, tvořený z přidružených služeb použitých při vývoji. Zde se jedná především o propojení následujících služeb s Gitlabem:

- Redmine - možnost prokliku na úkol na základě čísla ve zprávě verzované jednotky (commit message).
- Jenkins - spouštění sestavení aplikace na základě nové verze, oddělené dle jednotlivých větví (hlavní, vývojová, vedlejší) a publikace výsledku.
- SonarQube - zobrazování interaktivního výsledku statické analýzy přímo v rozhraní Gitlab.

Samotný SonarQube bylo potřeba nastavit, aby se spouštěl při sestavení aplikace a výsledek se zobrazil v rozhraní Gitlabu. V rámci sestavení aplikace jsem nastavil spouštění nástroje Cobertura. Doplňky v Jenkins umožňovaly zobrazení přehledných výsledků, jak je kód pokryt testy viz ukázka 4.2. Přesné pokrytí testů mi poté umožňovalo jednoduše kontrolovat, jaké části kódu jsou otestované.

Kapitola 4

# Zhodnocení týmového projektu

Pro návaznost na kapitolu o provedených vylepšeních je nejprve nutné uvést v jakém kontextu jsou navrhovány. K tomu je třeba popsat způsob hodnocení, výsledný stav projektu a jeho funkcionalitu, čemuž se budu věnovat v této kapitole.

## 4.1 Způsob hodnocení

Jako metriky zhodnocení byly zvoleny následující kritéria seřazené od nejpodstatnějšího k nejméně důležitému:

- Kritické chyby.
- Úplnost požadované funkcionality.
- Rozšiřitelnost.
- Neefektivní chování.
- Uživatelská přívětivost.
- Škálovatelnost.

Rozšiřitelnost je hodnocena na základě obecného návrhu, pokrytí testy a počtu chyb statické analýzy kódu. Jednotlivým bodům se budu nejprve věnovat v rámci popisu konkrétních problémů. Následně budou shrnuty pomocí tabulky ze které bude vyvozen závěr analýzy.

Zhodnocení bylo provedeno na základě testování a prozkoumání kódu. Při testování jsem využil testovací data zobrazené v tabulce 4.1. Postup inicializace systému spočíval ve vytvoření kampaně, namapování produktů a manuálního přidání detailů produktů do databáze.

| Produkt | Obchody |
|---|---|
| REMINGTON S 8500 | hair-cosmetics.cz<br>terdom.cz<br>notino.cz<br>mameradivlasy.cz |
| 24"Samsung S24D300H | alza.cz |

Tabulka 4.1: Data použitá k otestování řešení týmového projektu.

## 4.2 Stav

Ačkoliv stav projektu odpovídal požadavkům na úspěšné odevzdání, nebyla dosažena implementace všech procesů. Tím bohužel nebylo možné reálné použití systému.

Odevzdávaný stav obsahoval funkční webové rozhraní, které se skládalo ze základní funkcionality pro uživatele a administrátory. Část pro administrátory obsahovala správu uživatelů, evidenci známých obchodů a řešení chyb vzniklých v interní části.

Uživatelská část umožňovala správu a vytvoření kampaní. Splňovala tak návrh z analytické částí 2.4.1. Na žádost jiného týmu bylo vytvořeno Web API rozhraní, poskytující získané ceny pro daný produktu ve formátu JSON.

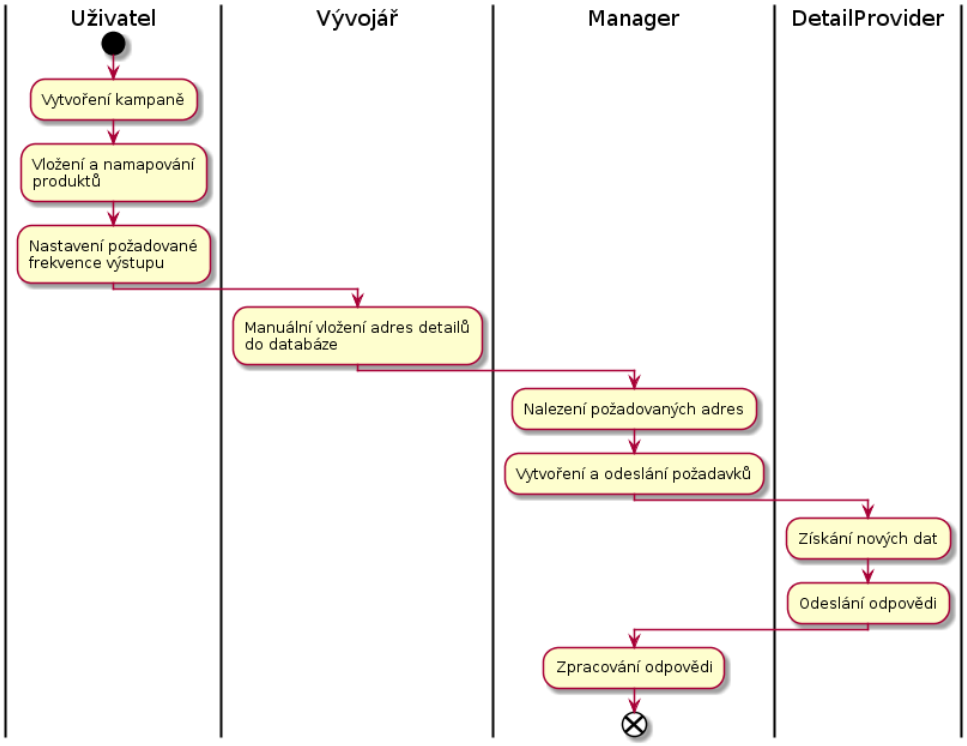
Interní část dokázala hledat nové data na již uložených adresách detailů. Manager vybral adresy detailů pro zpracování, vytvořil jednotlivé požadavky obsahující potřebná data a odeslal je k zpracování do DataProvideru. DataProvider obsah adresy stáhnul, vyparsoval a výsledek zanalyzoval vůči historickým datům. Analýza se skládala z kontrol velkých výkyvů cen a rozdílných identifikátorů produktu. Pokud všechny části proběhly bezchybně, byly data odeslány pro zpracování Managerem.

Systém se korektně zachoval i pokud při stažení, parsování nebo analyzování byla objevena chyba. Proces zastavil, výsledek označil jako chybný a odeslal chybnou odpověď. Zpracování Managerem zajistilo, že bylo možné chybu vyřešit administrátorem ve webovém rozhraní. Po vyřešení chyby systém zareagoval a vytvořil opětovný požadavek.

## 4.3 Nedostatky

### 4.3.1 Vytváření požadavků pro ProductProvider

Některé nedostatky byly úzce spjaty s tím, jak aplikace plánovala práci. Plánováním práce je myšlen proces nalezení adres detailů produktů pro které je požadováno získat aktualizované data. Z adres jsou vytvořeny a odeslány požadavky pro ProductProvider.




Obrázek 4.1: Diagram zobrazující vytvoření kampaně a nalezení nových dat v řešení týmového projektu.

Prvotním kritériem hledání jsou adresy detailů. Ty se mohou vyskytovat v různých stavech. Systém je vybral v následujícím pořadí:

- Adresy, které mají produkty v zaplacené kampani.
- Adresy bez produktů.
- Adresy, pro které neexistuje šablona pro parsování.
- Adresy, které mají vyřešenou chybu.

Z této množiny bylo vybráno prvních 10 adres, odebrány již odeslané a takové které mají nevyřešenou chybu. Opakované spuštění v předdefinovaném intervalu zajistilo vytvoření požadavků pro všechny požadované adresy.

Další nedostatek se skládal z ukládání stavu do databáze. Část nejdůležitějších atributů vytvářeného požadavku byla uložena do databáze s příznakem odesláno. Tento příznak bylo nutné uchovávat i u chyb šablon nebo analyzátoru, kde je potřeba označit zpracování Managerem.

Při neúspěchu odeslání požadavku už příznak u chyb nebyl změněn. To způsobilo, že chyba zůstala navždy s příznakem „zpracována“, ačkoliv se příslušný požadavek nepodařilo odeslat.

### 4.3.2 Neefektivní chování modulu Manager a ProductProvider

V rámci testování jsem zjistil, že v případě chyby při parsování stránky není použit uložený HTML dokument. I když dokument do databáze při zpracování chyby ukládán byl.

Manager vyhledal korektně adresy detailů, ale vytvoření objektu představující požadavek bylo pro všechny adresy stejné. Ztracení důvodu z jakého byla adres původně vybrána, způsobilo nemožnost jednoduchého uložení HTML dokumentu do požadavku. Každý požadavek znamenal opětovné stažení příslušné stránky.

### 4.3.3 Chyby analyzování

Poslední fáze procesu v modulu ProductProvider byla navržena jako analyzování získaných dat vůči již dříve uloženým. Implementace analyzátoru spouštěla jednotlivé validace, jejíchž logika se nacházela v oddělených třídách. Analýza kontrolovala zda se shoduje získaný EAN, název a modelové číslo, vůči uloženým identifikátorům. Pokud na jedné ze stran hodnota neexistovala, byly data označená, že jsou pravděpodobně chybná.

Dále probíhala kontrola získané ceny „s“ a „bez“ DPH oproti dříve získaným cenám na stejné adrese. Kontrola porovnala průměr historických hodnot s hodnotami získaných cen. V případě, že rozdíl byl větší jak 25%, byl výsledek označen jako chybný.

Po odeslání a přijetí chyby, Manager ji uložil do databáze pro administrátora k vyřešení. Administrátor pak mohl označit, zda je to opravdu chyba nebo se toto hlášení má do budoucna ignorovat.

#### 4.3.3.1 Problémová implementace

Pokud validační třída objevila nežádoucí data, vyhodila výjimku obsahující informace o chybě. Tento způsob řízení programu však způsoboval, že celá validace skončila při první chybě. Zároveň výjimka obsahovala pouze typ validace. Skončení po první validaci způsobovalo, že administrátor musel řešit chyby postupně. Pokud stránka obsahovala více chyb, byl po každém vyřešení vytvořen nový požadavek, který způsobil další chybu.

Samotné administrátorské rozhraní obsahovalo pouze informaci o typu chyby. Příčina byla nedostatečná práce s existující daty ve webovém rozhraní a zároveň nebyly ukládány detailnější informace o chybě.

### 4.3.4 Vytváření chyb šablon

Jako nedostatek se ukázalo plánování práce založené na kritériu, kdy jsou k vytvoření požadavku vybrány všechny adresy, které neobsahují šablonu. Myšlenka byla taková, že pro vytvoření samotné šablony, je nutné nejdříve stránku stáhnout, nechat vytvořit chybu parsování a následně ji vyřešit.

Systém však odeslal požadavek pro všechny uložené adresy na obchodě. Každá znamenala chybu šablony, které musel administrátor postupně vyřešit.

### 4.3.5 Modul Finder

Modul Finder nebyl zapojený do systému. Existovala pouze hlavní implementace interních procesů Finderu, jejíchž funkčnost byla pouze ověřena jednotkovými testy. Neexistující rozhraní pro práci s frontami a chybějící příslušné třídy Managera, které zajišťují vytváření příslušných požadavků neumožňovaly ověření celkové funkcionality této části. Z tohoto důvodu nebyl systém jako celek vhodný pro jakékoliv reálné použití, jelikož jediná možnost jak využít funkcionalitu interní částí, bylo vytvořit SQL insert skripty, obsahující adresy detailů produktů a ty spustit nad databází, kterou systém používal.

Další důsledek byla neexistence procesu párování produktů. Finder byl navržený tak, že po nalezení internetového obchodu pomocí interního vyhledávání nalezne detaily produktů, u kterých je velká pravděpodobnost, že patří hledanému produktu. Zda se jedná o správné adresy je třeba ověřit. Výsledkem hledání mohou být adresy, které nepatří hledanému produktu. Po získání hodnot ze stránky se musí nežádoucí adresy vyloučit a ostatní spárovat s uloženými produkty.

### 4.3.6 Plánování práce

Projekt neobsahoval plánování práce vůči požadovanému intervalu, kdy mají být nová data staženy. Aktuální stav hledal pouze adresy produktů, které se nachází v zaplacené kampani nebo měly vyřešenou chybu.

### 4.3.7 Škálovatelnost

Původní návrh počítal se škálovatelností aplikace na více serverech, kde je možné vytvořit neomezený počet instancí DataProvider a Finder. Reálný stav na konci projektu však tuto možnost neumožňoval. Interní část běžela jako jedna služba rozdělená do více modulů. Instance všech modulů probíhala v Managerovi, který musel mít přímou závislost na ostatních modulech.

### 4.3.8 Obecný návrh a testy

Implementace samotná byla velmi nepřehledná. Vykytovaly se prvky značící špatný návrh aplikace. Zde bych rád zmínil například dlouhé a nepřehledné

metody v DataProviderServiceImpl a AbstractFinderUrlListWorkerImpl, kde přestože jejich velikost nepřesahovala 60 řádků bylo velmi obtížné zjistit, co mají vykonávat. Problém byly také velké třídy jako například DataProvider-ServiceImpl zajišťující celý proces v modulu DataProvider.

Je nutné podotknout, že spousty špatných konstrukcí bylo eliminováno již v průběhu vývoje týmového projektu. První důvod byla statická analýza kódu detekující konstrukce, které jsou zdrojem častých chyb. Statická analýza však nebyla schopná najít všechny problémy. Druhý důvod byla moje kontrola při schvalování vytvořeného kódu pro zadaný úkol. Z důvodu časové tísně nebyl vždy prostor na to vrátit kód k přepsaní a opravení všech nedostatků. To způsobilo, že se vědomě dostaly do hlavních větví konstrukce, které nebyly považovány za ideální. Myšlenkou bylo, že budou přepracovány později, což se ne vždy povedlo.

Další problém návrhu byly procesy v DataProvider modulu řízené pomocí výjimek obsahující přídavné informace. A to i v případech, kdy byl takový výsledek očekávaný nebo dokonce chtěný. Toto použití je však v rozporu ideou použití výjimek, které mají signalizovat *neočekávaný* stav, kdy není možné dále pokračovat.[30] Pro uchování přídavných informací bylo nutné vytvářet výjimky vlastní, které obsahují údaje o chybě.

V kritických částech chyběly některé důležité testy, jelikož třídy snažící se dělat více věcí najednou, by bylo velmi složité otestovat. Chybějící testy se vykytovaly například u následujících částí: databázová vrstva, fasády, vytváření požadavků pro DataProvider, hlavní servisní třída DataProvideru nebo validace dat analyzátorem. Z tohoto důvodu jakákoliv oprava nebo implementace nových požadavků mohla narušit stávající funkcionalitu bez možnosti rychlého ověření. Tím by mohla být jakákoliv změna velmi časově náročná, s velmi nejistým konečným výsledkem.


Obrázek 4.2: Pokrytí testy vytvořeného řešení. Získáno pomocí nástroje Cobertura. Vizualizace výsledků byla vytvořena při sestavení na Jenkins s příslušným doplňkem.

## 4.4 Shrnutí

Na základě shrnutí nedostatků v tabulkách 4.2 a 4.3 je možné prohlásit, že pro základní použití systému by byla nutná oprava všech kritických chyb a doplnění o chybějící funkcionalitu. Pro zajištění jednoduchého zapracování

změn bude také nutný refaktoring, úprava návrhu a doplnění základních testů, které chybějí. Výsledek statická analýzy kódu ukázal, že počet nalezených chyb je minimální, což bylo způsobeno jejím průběžným použitím při vývoji. Při přezkoumání byly i přes výsledek analýzy kódu nalezeny nežádoucí konstrukce.

Problémy týkající se nedostatečné přívětivosti administrátorského rozhraní způsobují nepohodlné použití systému a bylo by vhodné je též zapracovat.

V případě provedení výše uvedených změn, předpokládám snadnou možnost oprav částí týkající se neefektivního chování. Neefektivita se může negativně odrazit při velké zátěži a proto by měla být odstraněna. Škálovatelnost má vzhledem k ostatním problémům nejnižší prioritu. Z tohoto důvodu není nutné se škálovatelnosti věnovat.

| Typ | Popis |
|---|---|
| Kritické chyby | • V případě chybného odeslání není chyba šablony nebo analyzátoru znova zpracována. |
| Úplnost funkcionality | • Systém nevyhledává internetové obchody.<br>• Systém nevyhledává adresy detailů produktů.<br>• Systém neplánuje práci v požadovaných intervalech.<br>• Neexistence procesu párování. |
| Rozšiřitelnost | • Návrh tříd byl vyhodnocen jako nevhodný pro rozšiřování.<br>• 60 % pokrytí testy dle nástroje Cobertura.<br>• 3 kritické chyby statické analýzy kódu.<br>• 1 minoritní chyba statické analýzy kódu. |

Tabulka 4.2: První část přehledu nedostatků vytvořeného týmového řešení.

| Typ | Popis |
|---|---|
| Neefektivní chování | • Stránka je nově stažena v každém požadavku (i v případě uložení při chybě).<br>• Každá chyba analyzátoru znamená samostatný požadavek a chybu.<br>• V případě neexistenci šablony detailu pro obchod jsou vytvořeny požadavky pro všechny adres, kdy každá způsobí chybu parsování, kterou je nutné vyřešit. |
| Uživatelská přívětivost | • Nutnost řešit chyby analyzátoru po jedné.<br>• Řešené chyby neobsahují bližší informace (informace o chybě, vyparsované hodnoty, použitá adresa detailu). |
| Škálovatelnost | • Nemožnost vytvořit více instancí modulů. |

Tabulka 4.3: Druhá část přehledu nedostatků vytvořeného týmového řešení.

Kapitola 5

# Návrhy na vylepšení

V této kapitole se věnuji návrhům na možná vylepšení. Samotné implementaci se poté věnuji v následující kapitole. Nedostatky, které byly zjištěny až v průběhu implementace vylepšení a nebyly zároveň i opraveny, budou zmíněny v kapitole týkající se zhodnocení provedených vylepšení. Při implementaci jsem funkčnost provedených změn průběžně ověřoval pomocí jednotkových testů a testovacích dat použitých při zhodnocení týmového projektu. Hledání produktů a párování bylo ověřováno na základě dat nových. Těmto datům se budu podrobněji věnovat v další kapitole, *Zhodnocení provedených vylepšení.*

## 5.1 Refaktorování stávajícího řešení

Na základě přezkoumání kódu bylo určeno, že by bylo vhodné provést refaktorování, jelikož existujícím kódu není možné stavět opravy nebo přidání nových funkcionalit. Příčinou jsou konstrukce jako zneužívání výjimek, dlouhé metody, velké třídy nebo dlouhé seznamy parametrů. Z těchto důvodů je třeba vhodně interní část refaktorovat tak, aby bylo možné kód lépe udržovat a rozvíjet. Provedené změny důkladně otestovat pomocí jednotkových testů.

V rámci refaktorování je nutné se pokusit zachovat co nejvíce původního kódu, obzvlášť takového, kde je ověřena funkcionalita. Dále pro větší přehlednost přesunout všechny servisní třídy do samostatného balíku a sjednotit je.

Při úpravách týkající se rozdělování jednotlivých tříd do samostatných jednotek, musí být dán důraz na jejich přehledné a rozšiřitelné komunikační rozhraní.

### 5.1.1 Řízení aplikace

Chování modulu ProductProvider je řízeno pomocí chytání výjimek obsahující informace o chybě. Výjimky by bylo vhodné odstranit a návratové hodnoty změnit na objekt obalující celkový výsledek. Tento návrh ulehčí implemen-

taci procesů, kde není žádoucí skončit při první chybě. Kód bude možné lépe rozdělit a metody následně zkrátit, což výrazně zlepší přehlednost kódu.

## 5.2 Plánování práce

Samotná logika plánování práce, nebo-li nalezení adres detailů produktů, které chceme použít při vytváření požadavků, se ukázala být nedostatečná. Chybí požadovaná funkcionalita, tedy použití intervalu určující, kdy je požadován nový výstup.

Nový návrh by měl hledat adresy podle těchto kritérií:

- Adresy, které mají produkty v aktivní kampani a požadovaný interval hledání odpovídá aktuálnímu dni.
- Adresy bez produktů.
- Adresy, pro které neexistuje šablona pro parsování.
- Adresy, které mají vyřešenou chybu.

Po nalezení těchto disjunktních množin a odstranění duplicit, vyřadit adresy, které z nějakého důvodu nevyhovují svým stavem. Nežádoucí stavy jsou tyto:

- Pro obchod existuje nevyřešená chyba šablony.
- Existuje nevyřešená chyba analyzátoru.
- Požadavek pro adresu byl již odeslán.

Z důvodu možnosti, že nežádoucí stavy bude pravděpodobně požadované přidat či odebrat, je potřeba implementaci navrhnout tak, aby bylo možné kontroly kdykoliv modifikovat bez velkých zásahů do interní funkcionality.

## 5.3 Oprava komunikace Manager - ProductProvider

Vzhledem k problémům popsaných v kapitole 4.3.2 je požadováno komunikaci modulů Manager a ProductProvider optimalizovat. Neefektivní chování by se mohlo ukázat jako velký problém při zpracování velkého počtu dat z důvodu nutnosti opakovaného stahování webových stránek.

Při analýze kódu se ukázalo, že v aktuálním návrhu aplikace není možné tuto funkcionalitu jednoduše implementovat, aniž by se nejednalo o rychlou opravu neefektivním řešením. Oprava by znamenala, že pro každou vytvářenou

adresu se systém musí pokusit najít existující chybu. Hledání by tak probíhalo ve většině případů zbytečně, jelikož chyba by neexistovala.

Korektní oprava je úzce spjata s předchozími kapitolami. Především s refaktorováním a úpravou plánování práce. Pokud úprava plánování zachová informaci o důvodu udávající, proč je adresa pro vytvoření požadavku zařazena, stačí chybu obsahující staženou stránku nalézt pouze u příslušných požadavků.

Informace důvodu nalezení by však měla být zachována i při odesílání. V rámci odesílání požadavků jsou nastavovány příznaky o odeslání a zpracování chyby. Když se požadavek nepodaří odeslat musí existovat možnost příznak změnit.

Z hlediska použití stránky v DataProvideru stačí zkontrolovat, zda požadavek stránku uchovává a v kladném případě ji použít pro parsování.

## 5.4 Spojení chyb analyzátoru a vylepšení rozhraní

Analyzátor provádí kontrolu získaných dat. V případě podezření o nevalidních datech je vytvořena chyba určená k vyřešení administrátorem. Validace kontrolují ekvivalenci identifikátorů vůči již uloženým a velké výkyvy cen na daném obchodu.

Administrátor má možnost chybu vyřešit a uložit příznak, aby se chyba do budoucna ignorovala. Jediné informace, které má při řešení k dispozici je typ validace. Webové rozhraní ani nemá možnost zobrazit dodatečné informace o chybě, protože nejsou ukládána. Proto by interní část měla takové informace zpracovat a uložit.

V případě adresy, která obsahuje více chyb, je nutné řešit každou samostatně. Po každém vyřešení se musí počkat na zpracování interní částí. Pro vylepšení administrátorské rozhraní je požadováno chyby spojit do jedné, tak aby bylo možné vyřešit všechny chyby analyzátoru najednou.

## 5.5 Monitorování

Na virtuálním serveru probíhá sestavení aplikace, včetně všech dodatečných procesů. Zároveň zde běží vývojová a produkční verze interní i webové části. Momentální stav poskytuje pouze omezenou možnost, jak sledovat využití prostředků virtuálního serveru.

Pro lepší přehled běžících prostředků by bylo vhodné zvolit monitorovací službu umožňující sledování probíhajících procesů na serveru, včetně vytížení a stav zobrazuje na externí službě, která bude funkční i v případě, kdy server neběží.

## 5.6 Získání adres obchodů a příslušných detailů produktů

Interní část vyžaduje ke své funkcionalitě již uložené adresy detailů produktů. Pomocí nich jsou získávána nová data o produktech. Původní návrh počítal s modulem Finder, který se nepodařilo zapojit v rámci týmového projektu. Ten měl za úkol hledat internetové obchody na cenových srovnávačích a na nich pomocí vyhledávaní nalézt konkrétní adresy.

Funkce Finderu je navržena jako duální, zajišťuje jak hledání samotných obchodů, tak i detailů adres. Komunikační třída představující příslušný požadavek, proto musí obsahovat příznak o jaký typ požadavku se jedná. Jelikož předávané informace jsou odlišné, vytvořený požadavek obsahuje velké množství prázdných hodnot, což přispívá k celkové nepřehlednosti.

Z tohoto důvodu navrhuji rozdělení Finderu na dva samostatné moduly. První bude zastávat funkci hledání obchodů a druhý vyhledávat na obchodu a získávat požadované adresy detailů produktů na daném obchodě.

Samotné hledání obchodů pak není kritická funkcionalita a je možné ji nahradit manuálním vložení internetových obchodů pomocí webového rozhraní.

## 5.7 Párování produktu

Po nalezení adresy detailu produktu, jejím stažení v DataProvider modulu a následném vyparsování hodnot je třeba adresu spárovat s produktem uloženým v databázi. Příčinou nutnosti párování je, že po nalezení adresy detailu není jisté, jestli opravdu patří produktu, pro který byla nalezena.

Párování musí být provedeno s velkou jistotou. Z tohoto důvodu navrhuji vytvořit proces, který se nejprve pokusí produkt spárovat na základě shody některého z identifikátoru, což představuje název, modelové číslo nebo EAN produktu.

Proces není možné zcela zautomatizovat, jelikož část internetových obchodů nemusí poskytovat informace totožné s těmi uloženými. Obchod může používat odlišný název. Odlišnosti identifikátorů může způsobit například jiná barva nebo přidaná velikost za nebo před modelové číslo. Jako řešení se jeví hledat podřetězec modelového čísla a EANu, což řeší problém pokud je ze stránky vyparsován text okolo identifikátoru. Obchod také může poskytovat pouze název. To lze demonstrovat na obchodu *glamot.cz*, například pro produkt BaByliss PRO Difuser Murano *[cit. 24.4.2017]*.

V případě neúspěchu párování, musí existovat možnost produkt spárovat manuálně, akcí administrátora. Výše uvedený proces spoléhá na to, že vložená data jsou při vytváření kampaně validní. V případě nevalidních dat jako příliš obecných a krátkých názvů by párování mohlo proběhnout chybně.

## 5.8 Neimplementované návrhy

### 5.8.1 Pokročilé párování produktu

Párování produktu lze vylepšit o možnost uchování více hodnot u identifikátorů produktů. V praxi by to znamenalo, že produkt je vytvořen s hlavními identifikátory. V průběhu hledání na obchodech a po potvrzení administrátorem by bylo možné uložit identifikátory alternativní.

Alternativní název by bylo možné využít při párování produktu, pokud by při použití hlavního názvu nebyla nalezena shoda.

### 5.8.2 Uchování a využití hodnot z nespárovaných adres

Vyhledáváním na obchodu je zpravidla výsledek, který obsahuje více adres detailů produktu. Na všech získaných adresách je proveden pokus o spárování, takže jsou nejdříve vyparsována příslušná data z těchto stránek. Pokud systém neuchovává všechny existující produkty prodávané na internetu, pak část adres vždy náleží neznámým produktům a není v době hledání potřebná. Získáná data by bylo možné uchovávat a pokusit se je spárovat při přidání nových produktů do databáze. To umožní odlehčení zátěže na stahování stránek a urychlí celý proces nalezení detailu adresy a párování produktu.

Kapitola 6

# Realizace vylepšení

Kapitola Realizace vylepšení se věnuje implementovaným vylepšením. Popisuje, jak byly navržené změny provedeny. V průběhu realizace byly objeveny nové nedostatky, z nichž některé byly také zpracovány, i když se s nimi původně nepočítalo. Změny jsou v repozitáři webového rozhraní a interní části přehledně označeny. Původní verze týmového projektu byla označena jako tag *v0.53*, realizované vylepšení jsou označeny jako *v0.6*.

## 6.1 Refaktorování stávajícího řešení

První krok realizace bylo refaktorování stávajícího řešení. Zde bylo provedeno především přesunutí tříd do jednotného balíčku, rozdělení tříd, odstranění přebytečných výjimek, zkrácení a zmenšení počtu parametrů metod.

### 6.1.1 Servisní třídy

Pro větší přehlednost byly všechny servisní třídy přesunuty do nadřazeném balíčku *service*. Servisní třídy jsou takové, které nespadají do ani jedné z těchto skupin:

- Obsluha frekvenčního probouzení aplikace v daném intervalu.
- Rozhraní komunikující s frontami.
- Třídy přistupující k databázi (DAO).
- Fasády, které obalují komunikaci servisních a DAO tříd.
- Konfigurační soubory automatické správy závislostí.
- Komunikační třídy.
- Pomocné třídy.

Třídy jsem pojmenoval pomocí nové konvence, kdy důležité servisní třídy obsahují prefix zkratky modulu, kterého se týkají a postfix *Service*. Důvodem byla větší přehlednost v projektu, kdy docházelo k podobným názvům napříč moduly.

Změnu lze demonstrovat například na třídě zajišťující získávání dat ze stažené stránky. Třída *cvut.fit.dataprovider.parser.ParserImpl* byla pak změněna na *cvut.fit.dataprovider.service.parser.DPParserServiceImpl*.[1]

### 6.1.2 Řízení aplikace

Aplikace, především v DataProvider modulu byla řízena pomocí výjimek, které způsobovaly problémy v návrhu a samotné funkcionalitě. V návaznosti na návrh byly nahrazeny úpravou návratového typu, který obsahuje příznak výsledku a doplňující informace.

Návratový typ lze demonstrovat na následují třídě *DPParserRespose*[2], která je vrácena v DataProvider modulu po provedení parsování.

```
1  /**
2   * Entity to keep parsed response. Almost every
3   * attribute can be null,
4   * so getters return {@link Optional} of nullable type.
5   *
6   * @author Jakub Tucek
7   * @created 24.1.2017
8   */
9  public class DPParserResponse {
10
11     /**
12      * Flag for keeping result of parsing
13      */
14     boolean finishedProperly;
15
16     /**
17      * Parsed name of the product
18      */
19     private String name;
20
21     public boolean isFinishedProperly() {
22         return finishedProperly;
23     }
24
25     public void setFinishedProperly(
26       boolean finishedProperly) {
27
```

[1] Uvedené názvy jsou včetně nadřazených balíčků. Název třídy je v prvním případě pouze *ParserImpl*.

[2] Ukázka třídy je zkrácena oproti reálné verzi.

```
28          this.finishedProperly = finishedProperly;
29      }
30
31      public Optional<String> getName() {
32          return Optional.ofNullable(name);
33      }
34
35      public void setName(String name) {
36          this.name = name;
37      }
38  }
```

Třídy zajišťující parsování parsování hodnot ze stažené stránky v modulu DataProvider používají uvedenou strukturu. Ukázka kódu ukazuje použití příznaku označující, zda parsování proběhlo úspěšně. Další důležitý prvek je zapouzdření proměnných uchovávající data[31], jelikož přístup k proměnným je umožněn pouze pomocí *get* a *set* metod.

Metody *get* jsou oproti standardnímu návrhu pozměněny a nevrací přímo proměnnou, ale *Optional* této proměnné. Optional je kontejner, který může obsahovat požadovanou hodnotu[32] nebo být prázdný. Před přístupem k hodnotě nepřímo vyžaduje ověření stavu objektu. Očividnost prázdnoty návratové hodnoty nutí vývojáře s touto možností počítat, což omezuje výskyt nežádoucích výjimek, především *NullPointerException*[33].

Po nahrazení návratovými typy, jsou výjimky použity pouze v případech, kdy nastal neočekávaný stav a je nutné přerušit následující akce.

### 6.1.3 Odstranění výjimky a spouštění validací

Odstranění výjimek z interní části lze demonstrovat na třídách zajišťující analyzování získaných výsledků v modulu DataProvider. Hlavní změny v této části jsou tři.

První je přesunutí hlavního rozhraní *Analyser* a jeho implementace *AnalyserImpl* z balíčku *cvut.fit.dataprovider.analyser* do *cvut.fit.dataprovider.service.analyser*.

Druhá změna představuje změnu rozhraní, kdy bylo potřeba zmenšit počet parametrů a odstranit výjimku, která byla vyhozena v případě nalezení chyby.

Třetí změnou je samotné spouštění validací. V původním řešení byla třída závislá na všech příslušných validacích, které spouštěla. Skončila však vždy při první chybě, což je jedna z příčin chování, na kterou reaguji v návrhu na spojení chyb analyzátoru, 5.4.

Vytvořil jsem nový návrh, který je v modifikovaných verzí použit i na ostatních místech interní části. Analyzující servisní třídě jsem odebral jednotlivé závislosti na konkrétních validacích a nahradil kolekcí obsahující validační rozhraní. Jednotlivé implementace validačního rozhraní jsou pak do třídy nastaveny konfigurační třídou automatické správy závislostí.

Validačnímu rozhraní byl změněn návratový typ na *Optional* třídy obsahující zprávu o chybě. Pokud chyba nenastala, vrací prázdnou hodnotu. Implementace validačního rozhraní byly rozděleny podle toho, jakou hodnotu kontrolují. Při úpravě validací jsem zjistil, že základní validace lze rozdělit na dvě skupiny: validace řetězce a čísla.

V případě těchto skupin se vytvořený kód lišil pouze v tom, jaká hodnota se má získat ze získaných dat a z dat již uložených. Poslední rozdíl byl pouze v chybové hlášce. Z toho důvodu jsem společnou logiku obou skupin implementoval pomocí abstraktní a generické třídy *AbstractAnalysis*. Vlastnosti kontrol jednotlivých skupin zajišťují třídy *AbstractStringAnalysis* nebo *AbstractPriceAnalysis*. Výsledná validace kontrolující získaný název s uloženým vypadá následovně:

Listing 6.1: Validace kontrolující hodnotu získaného jména produktu.

```
1  /**
2   * NameAnalysis is extension of {@link AbstractStringAnalysis} for
        analysing Name.
3   *
4   * @author Jakub Tucek
5   * @created 27.1.2017
6   */
7  public class NameAnalysis extends AbstractStringAnalysis {
8
9      /**
10      * {@inheritDoc}
11      */
12     @Override
13     boolean skipAnalysis(DataProviderRequest request) {
14         Optional<ComAnalyserFlags> analyserFlags =
               request.getAnalyserFlags();
15         return
               analyserFlags.map(ComAnalyserFlags::isIgnoreDifferentName)
16                 .orElse(false);
17     }
18
19     /**
20      * {@inheritDoc}
21      */
22     @Override
23     Optional<String> getOptionalProperty(DPParserResponse
           parserResponse) {
24         return parserResponse.getName();
25     }
26
27     /**
28      * {@inheritDoc}
29      */
30     @Override
```

```
31      List<String> getComProductValues(ComProduct comProduct) {
32          return comProduct.getNames();
33      }
34
35      /**
36       * {@inheritDoc}
37       */
38      @Override
39      AnalysisErrorMessage generateAnalysisErrorMessage(String
            comProductValue, String parsedValue) {
40          return new AnalysisErrorMessage()
41                  .withErrorMessage(
42                          String.format("Parsed name value[%s] doesn't
                                match known name value[%s]",
43                                  parsedValue, comProductValue)
44                  )
45                  .withErrorType(AnalysisErrorType.DIFFERENT_NAME);
46      }
47  }
```

Jednotlivé validace implementují pouze metody vracející příslušná data se kterými validace pracuje. Jedná se o vyparsované hodnoty, historická data a příznak určující, jestli administrátor nastavil ignorování chyby. Poslední implementovaná metoda vytváří informace o případné chybě. Tuto informaci je poté možné využít pro detailnější zobrazení v administračním rozhraní. Vylepšení samotného webového rozhraní chyb analyzátoru se věnuji v následující sekci.

Konečné spuštění validací bylo ve výsledku zkráceno na metodu obsahující jeden řádek kódu, ačkoliv tento řádek obsahuje více zřetězených volání.

Listing 6.2: Upravená implementace hlavní metody ve třídě zajišťující spouštění validací analyzátoru.

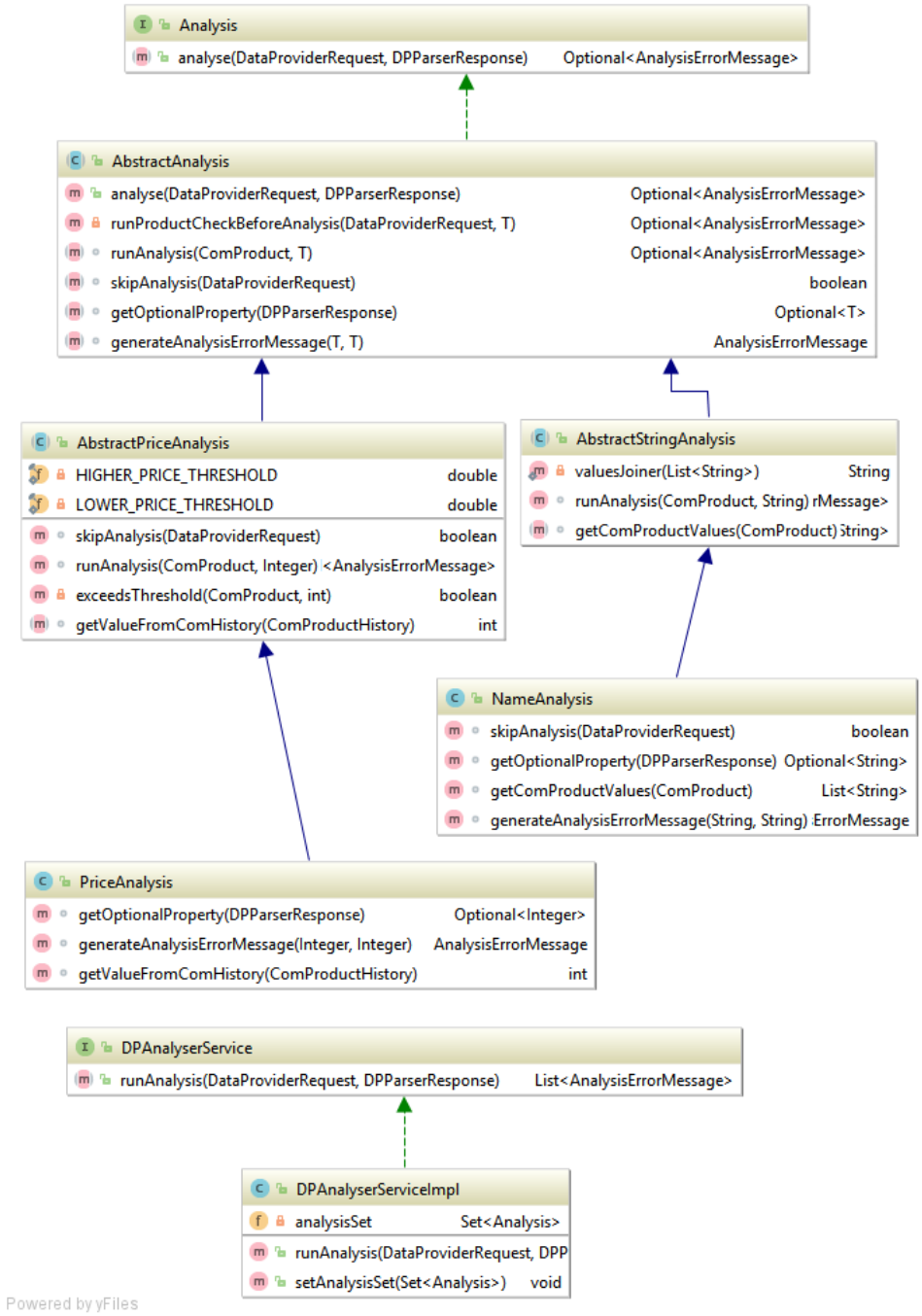
```
1       /**
2        * Runs analysis for given {@link DataProviderRequest} and {@link
             DPParserResponse}.
3        * Error are returned as list of {@link AnalysisErrorMessage}.
4        * Injected set of {@link Analysis} is executed one by one,
             result unwrapped and kept if present.
5        * Set of analysis result error messages is returned.
6        *
7        * @param request      dp request
8        * @param parserResponse parsed data
9        * @return list of errors or empty (if result was valid)
10       */
11      @Override
12      public List<AnalysisErrorMessage> runAnalysis(DataProviderRequest
            request, DPParserResponse parserResponse) {
13          return analysisSet.stream()
```

```
14              .map(x -> x.analyse(request, parserResponse))
15              .filter(Optional::isPresent)
16              .map(Optional::get)
17              .collect(Collectors.toList());
18
19      }
```

Listing 6.3: Původní implementace hlavní metody ve třídě zajišťující spouštění validací analyzátoru.

```
1       /**
2        * Analyses the new product info in comparison with the history
3        *
4        * @param newInfo      the new product info to be analysed
5        * @param newData
6        * @param oldInfo      the old product info
7        * @param productHistory the history of the product info @throws
             AnalyserException when analysing fails, contains error type
8        * @param analyserFlags
9        */
10      @Override
11      public void analyse(ComProduct newInfo,
12                          ComProductHistory newData,
13                          ComProduct oldInfo,
14                          List<ComProductHistory> productHistory,
15                          ComAnalyserFlags analyserFlags) throws
                                AnalyserException {
16          ValidatorData data = new ValidatorData(newInfo, newData,
                oldInfo, productHistory, analyserFlags);
17          try {
18              eanValidator.validate(data);
19              partNumberValidator.validate(data);
20              priceValidator.validate(data);
21              namesValidator.validate(data);
22          } catch (AnalyserException e) {
23              logger.info("Analysis failed for product id: {}",
                    newInfo.getProductId(), e);
24              throw e;
25          }
26          if (!data.getWarnings().isEmpty()) {
27              //No handling neeeded at the moment
28          }
29      }
```

Jak již bylo zmíněno, podobná architektura spouštění validací se vyskytuje na více místech interní části, například u validace po provedení parsování nebo kontrol, zda má být adresa použita pro vytvoření požadavku.




Obrázek 6.1: Diagra tříd analyzátoru. Obsahuje pouze některé validace pro zachování přehlednosti.

## 6.2 Plánování práce

Plánování práce bylo rozděleno do tří částí, nalezení adres, vytvoření požadavků a samotné odeslání. Tato sekce se týká pouze samotného nalezení adres, které jsou následně použity v dalším zpracování.

V rámci hledání samotných adres byly implementovány servisní třídy, které adresy hledají podle kritérií popsaných v návrhu vylepšeních. Následně jsou vráceny v obalovací třídě, která uchovává důvod nalezení adresy. Nalezení adres nově počítá i s frekvencí nastavenou u kampaně.

Nalezené adresy všech typů jsou vyfiltrovány o nežádoucí stavy. K tomuto účelu jsem navrhl rozhraní kontrol, rozhodující o tom, zda adresu použít. Tento návrh se podobá validačním kontrolám analyzátoru v modulu DataProvider.

## 6.3 Oprava komunikace Manager - ProductProvider

Důležitý prvek samotné opravy komunikace modulů Manager a ProductProvider je vytváření požadavků z nalezených adres. Vzhledem ke změně rozhraní, které zachovává způsob nalezení adres, stačilo vytvořit ostatní části modulu Manager, tak aby odpovídalo novému rozhraní.

Hlavní metoda servisní třídy *DPRequestSenderServiceImpl* volající všechny tří části vypadá následovně:

```
1   /**
2    * Creates requests and sends them.
3    *
4    * @throws MQConnectionException when sending fails
5    */
6   @Override
7   public void createRequests() throws MQConnectionException {
8       ProductUrlSets requiredProductUrls =
            prioritizeService.findRequiredProductUrls();
9
10      DPRequestProductUrlWrapper dpRequestWrapper =
            requestBuilderService.create(
11              requiredProductUrls);
12
13
14      senderService.sendDPRequests(dpRequestWrapper);
15  }
```

Nejdříve jsou nalezeny adresy, což bylo popsáno v kapitole 6.2. Poté jsou vytvořeny výsledné požadavky a odeslány. Servisní třída zodpovědná za odeslání zároveň ukládá příznaky stavu do databáze.

Vzhledem v zásadní změně architektury neuvádím původní kód, jelikož celkový způsob vytváření požadavků je značně odlišný a není proto možné

jednoduché porovnání. Vytváření bylo v předchozím řešení ve stejné třídě, která zajišťovala i odesílání, což způsobovalo problém při ukládání příznaků do databáze.

Nový návrh vytváření požadavků tento proces deleguje do nové třídy *DPRequestBuilderServiceImpl*, která zajišťuje uložení všech potřebných atributů do nového požadavku. Návratový typ této třídy slučuje skupiny požadavků obsahující adresy bez produktů a ty v aktivní kampani, jelikož nastavení příznaku o chybném odeslání je u obou typů totožné.

Vzhledem k potřebě pracovat s adresou detailu i při odesílání požadavků, obsahuje *DPRequestProductUrlWrapper* kromě vytvořeného požadavku i původní adresu detailu produktu. Důvod je možnost přímého přístupu k chybě přes databázovou entitu adresy.

### 6.3.1 Odesílání požadavků

Odesílání požadavků jsem upravil, aby odpovídalo novému návrhu. Před odesláním je každý požadavek uložen do databáze a nastaven příznak o zpracování. V případě neočekávané chyby při odesílání, jsou tyto příznaky korektně změněny.

Samotné odeslání má u všech požadavků stejný postup. Nejprve jsem proto extrahoval části obsahující ukládání a změnu stavu požadavků či chyb do samostatné třídy *DPRequestPersistServiceImpl*. Poté jsem využil nativního rozhraní Java 8, *Consumer<T>*, které reprezentuje operaci přijímající jeden vstupní parametr a nevrací žádný výsledek. Rozhraní jsem použil k reprezentaci operace uložení a změny stavu v případě chyby.

Listing 6.4: Společná metoda zajišťující odeslání DataProvider požadavků.

```
1   /**
2    * Sends request via {@link RequestHandler}.
3    * Request is first persisted via {@link
       PersistanceDPRequestFacade} and it's id is set to the
       request in wrapper
4    * object. If failure while sending object through MQ occurs,
       then {@link Consumer} failureHandler is called,
5    * exception logged and rethrown.
6    * Package private because of static code analysis.
7    *
8    * @param requestProductUrl wrapped object containing {@link
       ProductUrl} and {@link DataProviderRequest}
9    * @param persistConsumer persisting consumer called before
       sending
10   * @param revertConsumer revert consumer called in case of
       sending failure
11   */
12  void send(DPRequestProductUrl requestProductUrl,
13          Consumer<DPRequestProductUrl> persistConsumer,
```

```
14              Consumer<DPRequestProductUrl> revertConsumer) {
15          try {
16              persistConsumer.accept(requestProductUrl);
17              providerRequestHandler.send(
18                      requestProductUrl.getDataProviderRequest());
19          } catch (MQConnectionException e) {
20              revertConsumer.accept(requestProductUrl);
21              logger.error("Sending dataProviderRequest error.", e);
22              throw new IllegalStateException(e);
23          }
24      }
```

Zde je nutné podotknout důvod, proč není po odchytnutí a zpracování výjimky vrácena opět *MQConnectionException*. Zvolený způsob iterace nad objekty a samotného volání odesílací metody totiž neumožňuje obsahovat v těle metodu vyhazující výjimku rozšiřující třídu *Exception*. Nedostatek architektury lze obejít pomocí *IllegalStateException*, která potomkem třídy *Exception* není.

Listing 6.5: Příklad zavolání metody odesílající požadavky.

```
1       dpRequestWrapper.getAnalyserErrors().forEach(x -> send(
2               x,
3               dpRequestPersistService::persistRequestAnalyserError,
4               dpRequestPersistService::revertRequestAnalyserError
5               )
6       );
```

### 6.3.2 Komunikační objekt a využití stažené stránky

Nejdříve je nutné popsat změnu komunikační třídy *DataProviderRequest*. Tato komunikační třída představuje jeden požadavek odeslaný pomocí front do DataProvider a skládá se z jednotlivých základních atributů a fragmentů. Fragmentem je myšlena serializovatelná třída představující jeden celek, například informace o produktu nebo šabloně.

Původní návrh počítal s příznakem označující typ požadavku pro DataProvider. Příznak označoval, zda je obsažena stažená stránka či nikoliv. Stav, kdy požadavek obsahoval stránky však nikdy nenastal z důvodu implementace plánování práce a vytváření požadavků. Jelikož jediný rozdíl těchto dvou typů byl v atributu uchovávající staženou stránku, odstranil jsem ho.

Některé další atributy či fragmenty jako produkt nebo šablona nemusejí být nastaveny. U všech těchto atributů a fragmentů jsem proto provedl změnu u *get* metod, aby vracely kontejner *Optional*. Tím je jasně indikovaná možnost, že nemusí být obsaženy.

V rámci DataProvideru stačilo vytvořit při přístupu k jednomu z těchto atributů dvě možné větvení aplikace. Například pokud byla stránka obsažena

v požadavku, třída *DPDownloaderServiceImpl* vrátila validní odpověď o stažení obsahující tuto stránku, což je demonstrováno na následující ukázce.

Listing 6.6: Veřejná metoda třídy *DPDownloaderServiceImpl* zajišťující stažení stránky obsahující detail produktu.

```
1    /**
2     * Downloads requested page and returns {@link
          DownloaderResponse} object.
3     *
4     * @param dataProviderRequest the request containing url to be
          downloaded
5     * @return DownloaderResponse encapsulating downloaded data or
          error
6     */
7    @Override
8    public DownloaderResponse download(DataProviderRequest
         dataProviderRequest) {
9        return dataProviderRequest.getDownloadedPage()
10                   .map(x -> new
                         DownloaderResponse(Jsoup.parse(x)))
11                   .orElseGet(
12                           () -> doDownload(dataProviderRequest)
13                   );
14   }
```

## 6.4 Spojení chyb analyzátoru a vylepšení rozhraní

Oprava více četnosti chyb a samotného rozhraní je posloupnost několika oprav. Bylo již zmíněno odstranění přebytečných výjimek, což zajistilo spouštění všech validací. Další krok je změna komunikačního objektu, který nově uchovává všechny chybné validace a detailnější informace o chybě.

Původní řešení obsahovalo dvě třídy určené pro odpověď, protože DataProvider má dvě výstupní fronty: pro validní odpověď a chybu. Třída pro validní odpověď je *DataProviderResponse*, pro chybu pak *DataProviderResponseError*.

Z důvodu velkého množství podobných atributů, obzvlášť po přidání vyparsovaných hodnot do chyby, jsem změnil návrh těchto objektů. Z *DataProviderResponseError* jsem odstranil společné atributy a jako jeho rodiče jsem zvolil přímo *DataProviderResponse*. Chybná odpověď nově obsahovala všechny atributy validní odpovědi.

Z hlediska Managera bylo potřeba tyto hodnoty uložit, jelikož se ukládaly pouze získané ceny. Vytvořil jsem novou tabulku uchovávající informace o vyparsovaných hodnotách, které mohou být použity v případě zobrazení chyby administrátorovi. Dále bylo nutné uložit detailní informace o chybách.

Proto jsem vytvořil další tabulku, která je spojena vazbou 1:n s databázovou reprezentací chyby.

Dalším krokem bylo upravení webového rozhraní, aby odpovídalo změněné databázové struktuře. První změna se týkala samotného zobrazení informacích o chybě. Zde jsem využil nově uložených dat. Administrátor má tak možnost vidět použité hodnoty při analyzování a všechny vyskytnuté chyby. Změna je viditelná na ukázkách webového rozhraní: 6.2 a 6.4.

Poslední úprava spočívala v zpracování vstupů od administrátora, kdy bylo potřeba uložit všechny možné příznaky pro budoucí analyzování.

## 6.5 Monitorování

Na virtuální server jsem nasadil službu DataDog[34], která po jednoduché instalaci umožňuje sledování běžících služeb a vytížení serveru. Data jsou odesílána přímo do služby DataDog. Externí webové rozhraní umožňuje zobrazení posbíraných údajů.

Základní funkcionalita poskytuje pouze informace o využití prostředků a přístup k logům. Ačkoliv základní funkcionalita postačovala pro potřeby mého projektu, službu je možné rozšířit o doplňky. Pomocí těch je možné sledovat například výsledek sestavení v Jenkins, ale i obsah a využití RabbitMQ front.

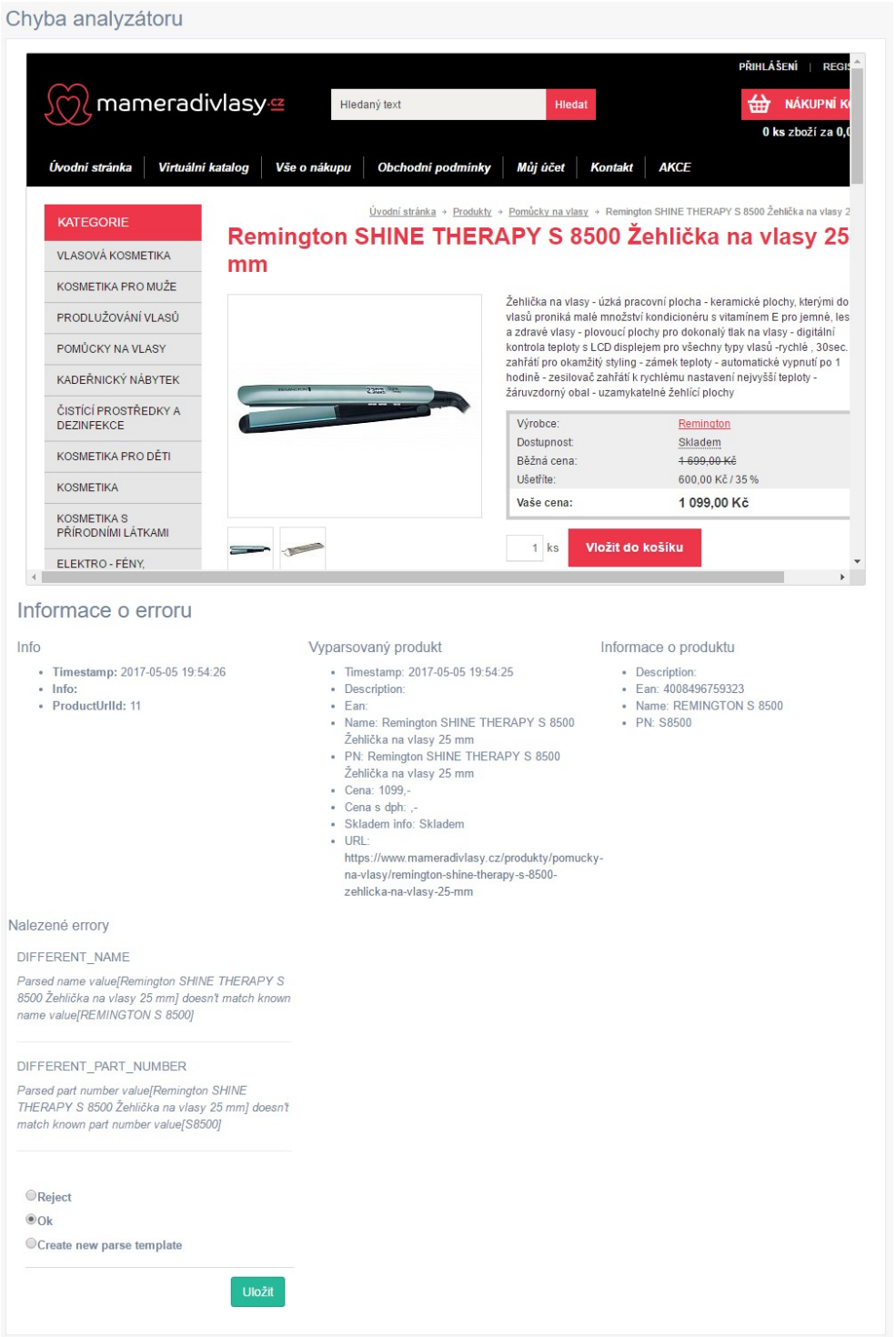
## 6.6 Získání adres obchodů a příslušných detailů produktů

Finder byl na základě důvodů popsaných v návrhu rozdělen na dvě části. Část vyhledávající na internetových obchodech adresy detailů popisující diagram aktivit 6.3 a na část, která samotné obchody vyhledává.

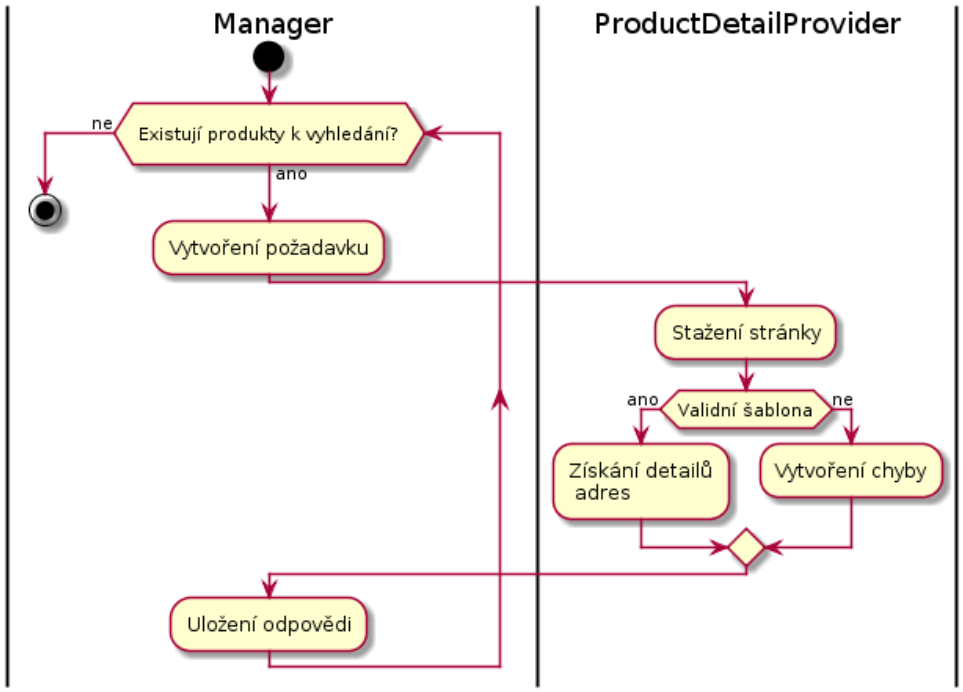
Implementována byla pouze první část, jelikož hledání samotných obchodů lze nahradit manuálním přidání obchodů, na kterých chceme vyhledávat, případně využít některý z veřejných seznamů internetových obchodů v České republice a ty manuálně vložit do databáze. Jako seznam by bylo možné využít například webovou stránku i-shopy.cz *[cit. 5.5.2017]*, která indexuje 3091 internetových obchodů.

Modul Finder byl zcela odstraněn a nahrazen modulem novým, nazvaným ProductDetailProvider. Tento modul zajišťuje hledání adres detailů produktů, což je dosaženo na základě šablony internetového obchodu, která obsahuje tyto atributy:

- formát URL sloužící k vyhledání produktu na obchodě,
- oddělovač slov v URL adrese,
- selektory pro výběr adres vedoucí na detaily produktů.




Obrázek 6.2: Ukázka webového rozhraní pro vyřešení chyby analyzátoru po provedení vylepšeních.




Obrázek 6.3: Aktivity diagram ilustrující nalezení adres detailů produktů.

Pro vytvoření požadavku je nutná existence šablony, která obsahuje informace, jak na obchodě vyhledávat. Po nalezení adres je vytvořena chyba pro administrátora, aby specifikoval výběr samotných adres detailů. Webové rozhraní bylo pro tento proces vytvořeno již v rámci týmového projektu.

## 6.7 Párování produktu

Byl implementován proces, který se nejprve pokusí produkt spárovat automaticky, pokud nalezne přímou shodu názvu, EANu nebo modelového čísla. Pro určení shody probíhá porovnání nalezeného identifikátoru s uloženým. Shoda je hledána na základě podřetězců.

Pokud je vyhledání podřetězců neúspěšné, jsou provedeny heuristiky hledající pravděpodobné shody. K detekci shody jsou použity algoritmy počítající společná slova a nejdelší společný podřetězec. Ze všech nalezených možností je vytvořena chyba, kterou musí zpracovat administrátor.

Do webového části bylo vytvořeno rozhraní, které administrátorovi umožňuje jednoduché přiřazení adresy k produktu nebo všechny nabízené možnosti odmítnout.

## 6.8 Detekce neexistující stránky a nenalezeného produktu

V případě použití interního vyhledávání se stávalo, že pro hledanou hodnotu nebyl nalezen žádný produkt, což bylo zpracováno jako chyba šablony, protože struktura stránky definované šabloně neodpovídala. Podobný případ byl u nalezení adresy detailu, která neexistuje, ačkoliv byla vrácena jako výsledek při vyhledávání.I pro tuto možnost byla vytvořena příslušná chyba šablony.

Nejprve jsem se snažil při nenalezení žádných hodnot výsledek porovnat se stránkou, která byla obchodem vrácena při hledání náhodného řetězce dlouhé délky. Myšlenka byla, že pokud produkt opravdu na obchodu není, stránka vrácena při nesmyslném hledání bude mít podobnou strukturu.

Problémem tohoto řešení se ukázala přílišná odlišnost HTML stažených stránek a specifičnost obchodů. Algoritmus fungoval pouze na malé části obchodů a proto bylo toto řešení zavrženo.

Nakonec jsem zvolil možnost, kdy administrátor má v rámci řešení chyby šablony možnost zadat řetězec značící neexistenci produktu (popř. neexistující stránky detailu produktu), viz. obrázek 6.4. Tento řetězec je pro každý obchod specifický a před vytvořením chyby šablony je nejdříve zkontrolováno, zda se řetězec na stránce nenachází. Pokud ano, nejedná se o chybu šablony, ale pouze nebylo nic nalezeno.

## 6.9 Více šablon detailů produktů

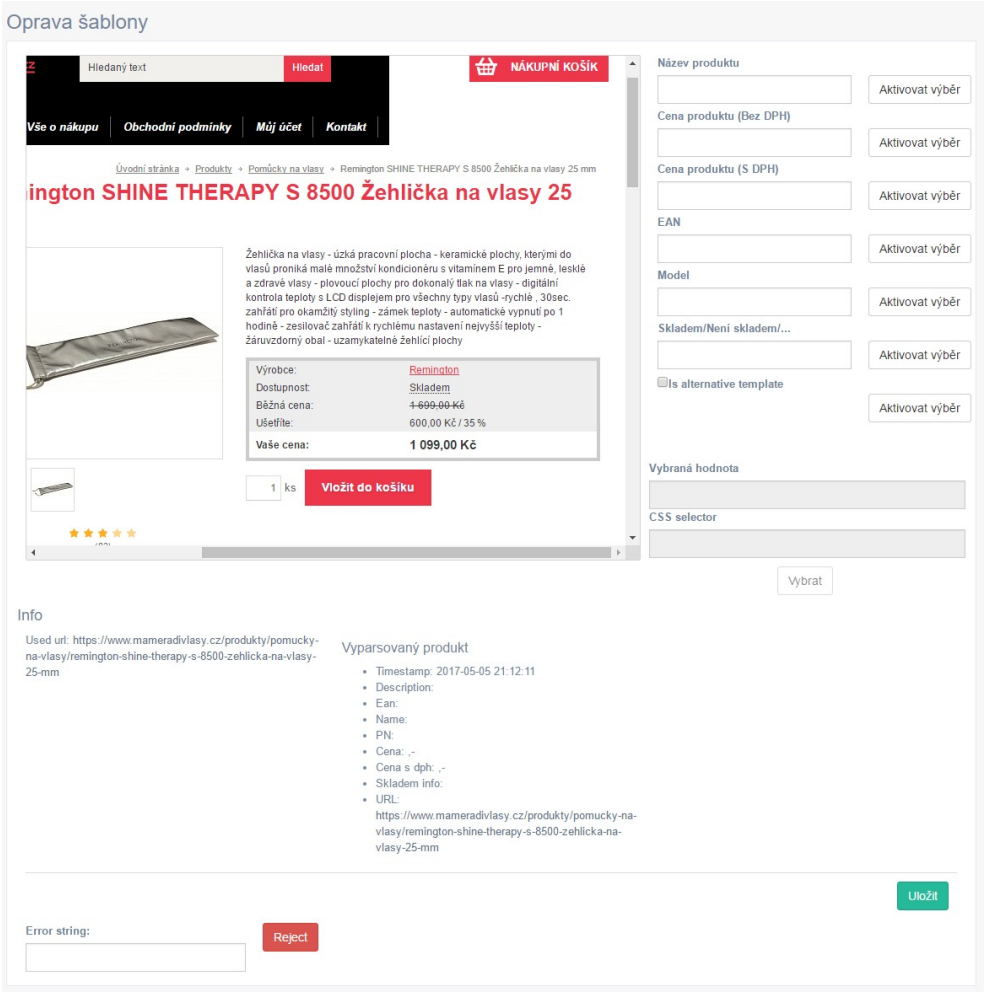
Původní návrh počítal se stavem, kdy stránka detailu produktu má stejnou strukturu napříč celým obchodem. Tento předpoklad se při testování ukázal jako chybný, kdy například výskyt slevy způsobil odlišný název elementu.

Důsledek byla chyba šablony. Abych problém odstranil implementoval jsem podporu alternativních šablon, které jsou použity v případě, že hlavní šablona selže. Možnost uložení šablony jako alternativní jsem zapracoval do běžného rozhraní pro editaci šablony detailu.

## 6.10 Více stejných chyb

Systém se i po změnách potýkal se stavem, kdy se v administrátorském rozhraní objevilo více chyb šablon nebo analyzátoru. V původním řešení tento stav nastával pokaždé, když byly vytvořeny požadavky pro všechny adresy vedoucí na obchod bez šablony.

Chyby lze predikovat a v případě neexistující šablony odeslat pouze jediný požadavek pro ProductProvider. Ačkoliv jsem upravil plánování práce a zároveň přidal kontrolu, která kontroluje zda není takový požadavek již ve frontě, tak se stávalo, že administrátor byl v některých případech zahlcen chybami. Zahlcení v tomto způsobovala šablona, která přestala fungovat.




Obrázek 6.4: Administrátorské rozhraní pro opravu detailu šablony. Původní řešení neobsahovalo informaci o použité URL adrese, vyparsované hodnoty a formulář pro chybný řetězec.

Jelikož tento stav není možné predikovat, zvolil jsem možnost, kdy je pouze upraveno webové rozhraní. Úprava webového rozhraní spočívala v omezení zobrazování chyb, kdy související chyby jsou v seznamu pouze jednou. Změněné zpracování formuláře „vyřešilo“ všechny chyby stejného typu, čímž odpadá nutnost je řešit.

## 6.11 Skladem

V rámci získávání dat u detailu produktu jsem přidal možnost získávat informaci, zda je produkt skladem. Provedl jsem tak na základě konverzace s provozovatelem internetového obchodu, který tento údaj považuje za důležitý.

Pro úpravu jsem upravil nejdříve rozhraní pro vytváření šablony a strukturu databáze, aby uchovávala atribut u šablony a výsledek parsování. Zajistil jsem, aby byl tento atribut zahrnut v rámci komunikace modulu Manager a DataProvider, kde bylo potřeba upravit správné uložení nových atributů do komunikačních tříd, získání hodnoty při parsování a následné uložení do databáze.

## 6.12 Ostatní

Při realizaci jsem provedl několik menších změn a oprav. Jedna z nich byla například nastavení připojení do databáze. Původní stav obsahoval konfiguraci připojení v samostatných *xml* souborech, ve kterých byly zduplikovány registrace databázových tříd.

Tento způsob jsem přepsal, aby nastavení databázových tříd bylo pouze v jednom souboru. Samotné připojení definují *properties* soubory, které mají následovnou strukturu:

Listing 6.7: Nastavení připojení do databáze.

```
1  hibernate.hikari.dataSource.url
2       =jdbc:mysql://localhost:3306/infoweb-db?characterEncoding=UTF-8
3  hibernate.hikari.dataSource.user=root
4  hibernate.hikari.dataSource.password=
5  hibernate.hikari.dataSourceClassName
6    =com.mysql.jdbc.jdbc2.optional.MysqlDataSource
7  database.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
8  hibernate.hbm2ddl.auto=create-drop
9  hibernate.hbm2ddl.import_files=sql/importScript.sql
10 hibernate.hbm2ddl.import_files_sql_extractor
11   =org.hibernate.tool.hbm2ddl.MultipleLinesSqlCommandExtractor
```

Úprava pak ulehčuje změny v databázových třídách a umožňuje jednoduchou konfiguraci pro více vývojových prostředí jako například testovací, vývojové a produkční.

Kapitola 7

# Zhodnocení provedených vylepšení

Zde bych se rád věnoval konečné funkcionalitě systému. Zhodnocení bude provedeno stejným způsobem a podle stejných kriterií jako zhodnocení týmového projektu, což je podrobně popsáno v sekci 4.1. Jediný rozdíl je použití odlišných testovacích dat, což jsem nastínil v předchozí kapitole.

## 7.1 Data

Testování samotného systému jsem provedl nad reálným vzorkem dat, které mi poskytl provozovatel internetového obchodu. Data se skládají z údajů o produktech popsaných v tabulce 7.1 a internetových obchodů vyjmenovaných v tabulce 7.2. U každého internetového obchodu bylo v rámci dat zaznamenáno jaké produkty se na něm vyskytují, což mi umožnilo jednoduše kontrolovat, zda systém nalezl všechny adresy detailů.

## 7.2 Funkcionalita

Díky implementaci požadované funkcionality a oprav kritických chyb, je v upraveném řešení možné využít celý proces hledání informací. Administrátor nově musí zadat přes webové rozhraní obchody, na kterých chceme hledat. Po vytvoření kampaně je také nutné nastavit příslušné šablony a spárovat produkty, kde to nebylo možné provést automaticky.

| **značka** | **model** | **název** |
|---|---|---|
| Scholl | F215311016 (43) | AIR BAG - přírodní<br>zdravotní pantofle |
| Scholl | F200781065 (43) | CLOG SUPERCOMFORT<br>bílá |
| Scholl | 4002448095262 | Velvet Smooth wet & dry<br>- elektrický pilník na chodidla do vody |
| Beurer | BEU-FB30 | FB 30 nožní masáž |
| Sanitas | SAN-SEM43 | SEM 43 svalový a nervový<br>elektrostimulátor |
| Beurer | BEU-464.15 | GL 44 / GL 50 / GL 50 EVO<br>testovací proužky 464.15 (2x25ks) |
| Beurer | BEU-IPL10000+ | IPL 10000+ Depilace SalonPro<br>System - depilace s dlouhodobým účinkem |
| Salter | SA1008GSBKXR | SALTER 1008 GSBKXR |
| Homedics | MIR-8150 | MIR M-8150 - kosmetické zrcadlo |
| Philips | Phil-71768/08/16 | Softpal Battery Olaf White |

Tabulka 7.1: Produkty použité v testovacích datech pro zhodnocení výsledného řešení.

| | | |
|---|---|---|
| muj-scholl.cz | kudrnka.cz | 4home.cz |
| muj-beurer.cz | zdravotnickaprodejna.cz | medipharma.cz |
| muj-sanitas.cz | eobuv.cz | obuv-scholl.cz |
| muj-salter.cz | mall.cz | k24.cz |
| muj-homedics.cz | pilulka.cz | alza.cz |
| elektrocr.cz | expert.cz | cesky-obchodak.eu |
| ajtrade.cz | datart.cz | diagnosticketesty.cz |
| nakupka.cz | | |

Tabulka 7.2: Obchody použité v testovacích datech pro zhodnocení výsledného řešení.

## 7.3 Dodatečné opravy

Plynulost a funkčnost celého procesu hledání dat byla dosažena pouze díky opravám chyb, které jsem nalezl v průběhu testování. V kapitole Realizace vylepšení 6 jsem popisoval implementaci dodatečných oprav. Je však nutné zmínit, jak byly tyto nedostatky nalezeny.

### 7.3.1 Více šablon detailů produktu a Více stejných chyb

Chyba, kdy obchod obsahoval více možných struktur pro detaily produktů a zobrazení více stejných chyb ve webovém rozhraní měly stejnou příčinu.

Systém při vyhledávání na obchodech *alza.cz* a *nakupka.cz* nalezl detaily produktů, které měly jinou strukturu než uložená šablona. Výsledek byla neočekávaná chyba, což způsobilo více stejných chyb. Chyby zahlcovaly webové rozhraní. Oprava rozhraní ovšem neřešila problém odlišné struktury a proto bylo třeba vytvořit i podporu pro šablony alternativní.

### 7.3.2 Neexistující stránka

Neúspěšné vyhledávání se vykytovalo na většině obchodech, protože hledaný produkt se na stránkách nenacházel. Systém prázdnou stránku detekoval jako chybu šablony.

Neexistující stránka detailu pak byla objevena na obchodě kudrnka.cz, kdy interní vyhledávání vracelo odkaz na neexistující detail produktu, což opět způsobilo chybu šablony, jelikož nebyla vyparsována žádná hodnota.

## 7.4 Párování

Při testování bylo zjištěno, že velké množství obchodů neobsahuje správné nebo přímo odlišné identifikátory produktů. Nejčastější je výskyt pouze jména, které často neodpovídá tomu uloženému.

Obchody poskytující pouze název znamenají nutnost provést párování manuálně. Systém se párování snaží ulehčit rozhraním, nicméně i při malém množstvím produktů je celkový počet těchto chyb velmi rozsáhlý.

Například na obchodě *mall.cz* bylo nutné manuálně spárovat 4 ze 6 produktů. Počet všech nabízených možností párování pro administrátora bylo 30. Jeden produkt se na obchodě nepodařilo nalézt a spárovat vůbec, jelikož obsahoval kompletně odlišný název oproti uloženému.

Proces párování by bylo možné vylepšit, pokud by systém uchovával více identifikátorů. Více uložených jmen by pak zátěž na administrátora mohla s časem klesnout, protože použitých jmen napříč obchody je omezené množství. U kompletně odlišných názvů je jediné řešení, vytvoření rozhraní umožňující čistě manuální párování.

## 7.5 Webové rozhraní

Nalezl jsem nedostatky ve webovém rozhraní, kdy jeho funkčnost sloužící k opravám šablony detailu, nebyla na některých obchodech funkční. Jednalo se například o obchod *pilulka.cz*, kde se při zobrazení staženého HTML nezobrazoval EAN, takže nebylo možné nastavit příslušný selektor. U obchodu *mall.cz* byl pro změnu obsah stránky překryt upozorněním o nepodporovaném prohlížeči.

Problém obsahovala i část pro šablonu sloužící k nalezení adres detailů. Pokud bylo použito rozhraní, šablona nebyla ve většině případů funkční.

Chyby v obou částech nejsou kritické, jelikož při zadání cesty k elementu manuálně, uložení šablony proběhne správně a systém může pokračovat v práci.

## 7.6 Návrh a testy

Zásadní refaktoring a změna návrhu komunikace tříd velmi pozměnila interní část. Interní část vyhovuje nárokům na formu. Je složena z přehlednějšího, udržovatelnějšího a lépe rozšiřitelného kódu.

Tyto vlastnosti byly výrazné při úpravách, kdy změny byly často triviální záležitost. Kromě subjektivního pocitu rychlejší orientace v kódu, nová architektura umožnila jednoduché přidání vylepšení a po provedení bylo zachována původní funkcionalita. Toto dávám za důsledek lepšímu pokrytí testy a rozdělení tříd do menších celků. Oproti původnímu řešení narostl počet jednotkových testů ze 170 na 492. Pokrytí v procentech demonstruje následující vizualizace.

| Name | Packages | Files | Classes |
|---|---|---|---|
| Cobertura Coverage Report | 80% 51/64 | 83% 185/224 | 82% 187/228 |

| Methods | Lines | Conditionals |
|---|---|---|
| 79% 584/735 | 79% 2321/2951 | 66% 312/472 |

Obrázek 7.1: Pokrytí testy po provedených vylepšení. Získáno pomocí nástroje Cobertura. Vizualizace výsledků byla vytvořena při sestavení na Jenkins s příslušným doplňkem.

Pokrytí vzrostlo zhruba o 20%. Základní funkcionalita servisních tříd je testy pokrytá celá. Neotestované části jsou především větve, která nastanou při neočekávané chybě a vyhození výjimky, kterou není možné jednoduše v testech nasimulovat. Další neotestované třídy se týkají komunikace s frontami a jednoduchých objektů, které vrací v *get* metodě *Optional*, což je detekováno jako neotestovaný řádek.

Po provedení změn zůstala nemožnost vytvoření více instancí jednotlivých modulů, tedy ProductDetailProvider a DataProvider. Pro změnu však stačí

přidat instanční rozhraní do jednotlivých modulů, jelikož momentálně jsou spouštěny přímo modulem Manager. Oddělení neovlivní samotnou komunikaci jednotlivých části, nicméně vzroste zátěž na nastavení infrastruktury a nasazení aplikace na server.

## 7.7 Rozhraní administrátora

Systém se stal z hlediska pro administrátora uživatelsky přívětivější. Při řešení chyb je rozhraní přehlednější a informace o chybách obsáhlejší. Dále byla snížena zátěž na administrátora, jelikož odpadla nutnost řešit všechny požadavky týkající se stejného obchodu či chyb analyzátoru.

Nevýhodou jsou nedostatky týkající se vytváření šablon, které byly zmíněny v kapitole 7.5.

## 7.8 Nemožnost vyhledávání na některých obchodech

V rámci testování jsem objevil nemožnost vyhledávat na obchodech, které mají vyhledávání implementované pomocí POST požadavku, což systém aktuálně nepodporuje. Jedná se například o obchod *elektrocr.cz*.

## 7.9 Shrnutí

| Typ | Popis |
|---|---|
| Kritické chyby | |
| Úplnost funkcionality | • Systém nevyhledává internetové obchody.<br>• Nemožnost vyhledávat na některých obchodech. |
| Rozšiřitelnost | • Návrh se ukázal jako vhodný pro úpravy.<br>• 80 % pokrytí testy dle nástroje Cobertura.<br>• 0 chyb statické analýzy kódu. |

Tabulka 7.3: První část přehledu nedostatků po implementaci navržených vylepšení.

| Typ | Popis |
|---|---|
| Neefektivní chování | |
| Uživatelská přívětivost | • Velká náročnost párování detailů adres s produkty.<br>• Rozhraní pro editace šablony detailu není funkční na některých stránkách.<br>• Nefunkční rozhraní pro editaci šablony pro vyhledávání na obchodě. |
| Škálovatelnost | • Nemožnost vytvořit více instancí modulů. |

Tabulka 7.4: Druhá část přehledu nedostatků po implementaci navržených vylepšení.

Kapitola 8

# Závěr

Vytvořené řešení týmového projektu nevyhovovalo požadavkům na funkcionalitu a možnostem na budoucí rozvoj. Funkcionalita nebyla splněna především z důvodu nezapojení části, která má za úkol hledat samotné obchody a adresy detailů produktů, které obsahují hledaná data. Nebyl také vyřešen proces párování produktu k adrese. Návrh některých tříd se ukázal jako nevhodný pro budoucí rozšíření.

V návaznosti na tyto nedostatky bylo navrženo rozsáhlé refaktorování, změna architektury problémových tříd a opravení stávajících procesů. Bylo potřeba opravit procesy, které nebyly funkční, ztěžovaly práci administrátora nebo byly neefektivní. U funkcionality, kterou měla zajišťovat nezapojená část bylo rozhodnuto o jejím rozdělení, kdy podpora hledání samotných obchodů byla nahrazena manuálním vložením přes webové rozhraní.

Refaktorování a změna návrhu tříd potvrdila, že se jedná do budoucna o výhodný krok zajišťující lepší udržovatelnost a rozšiřitelnost kódu. Což se ukázalo už při následných změnách, které bylo možné provést v rámci vytvořeného návrhu. Oprava stávajících procesů pak byla nejvíce viditelná v lepší uživatelské přívětivosti pro administrátora umožňují případné problémy řešit rychleji a spolehlivěji.

Po implementaci hledání detailu produktu na stránkách obchodů a procesu párování produktů s nalezenými adresami byly objevené nové nedostatky.

Kritické problémy opravila nová implementace, ale ukázalo se, že i po provedení změn není možné systém použít pro provoz, kde je požadavek na univerzalitu systému. Systém je po správném nastavení schopný dlouhodobě a automaticky sbírat určitý vzorek dat. Byly však objeveny obchody, kde to není možné. Důvod je především odlišná funkcionalita vyhledávání produktů na sledovaných obchodech, kterou vytvořený systém aktuálně nepodporuje.

Ačkoliv v závěru objevené nedostatky způsobují diskomfort při používání, věřím že přidání funkcionalit navržených v 7. kapitole, umožní plnohodnotné použití výsledného systému.

# Literatura


[1] Extensible Markup Language (XML) 1.0 (Fifth Edition). [online], 2008, [cit. 2017-03-13]. Dostupné z: https://www.w3.org/TR/2008/REC-xml-20081126/#sec-intro

[2] Virginia Tech - College of engineering: Department of computer science. [online], 2002, [cit. 2017-03-13]. Dostupné z: http://courses.cs.vt.edu/~cs1204/XML/htmlVxml.html

[3] HTML5: A vocabulary and associated APIs for HTML and XHTML. [online], 2014, [cit. 2017-03-13]. Dostupné z: https://www.w3.org/TR/html5/introduction.html#html-vs-xhtml

[4] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. [online], 2011, [cit. 2017-03-13]. Dostupné z: https://www.w3.org/TR/CSS21/selector.html#q5.0

[5] Rozhovor s Jiřím Hunkou, nar. 1985-05-12, provozovatel eshopů. 2016-11-28.

[6] Heuréka. [online], [cit. 2017-02-10]. Dostupné z: http://www.heureka.cz

[7] Zboží. [online], [cit. 2017-04-14]. Dostupné z: http://www.zbozi.cz

[8] Heuréka - Sortiment Report. [online], [cit. 2017-05-01]. Dostupné z: https://sluzby.heureka.cz/napoveda/sortiment-report/

[9] Price checking. [online], [cit. 2017-04-14]. Dostupné z: http://www.price-checking.cz/

[10] Pricing intelligence. [online], [cit. 2017-04-14]. Dostupné z: http://pricingintelligence.cz/

[11] Sledování trhu. [online], [cit. 2017-04-14]. Dostupné z: http://www.sledovanitrhu.cz/

[12] Pricebot. [online], [cit. 2017-04-14]. Dostupné z: http://www.pricebot.cz

[13] Screen scraper. [online], [cit. 2017-04-14]. Dostupné z: http://www.screen-scraper.com

[14] Web extractor. [online], [cit. 2017-04-14]. Dostupné z: http://www.webextractor.com

[15] The Java™ Tutorials. [online], 2015, [cit. 2017-04-14]. Dostupné z: https://docs.oracle.com/javase/tutorial/jdbc/overview/database

[16] Redmine. [online], 2017, [cit. 2017-04-10]. Dostupné z: https://redmine.org/

[17] GitLab. [online], 2017, [cit. 2017-05-06]. Dostupné z: https://gitlab.com/

[18] Jenkins. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://jenkins.io/

[19] SonarQube. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://www.sonarqube.org/

[20] Nette. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://nette.org/

[21] Composer. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://getcomposer.org/

[22] Gradle. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://www.gradle.org/

[23] Maven Repository. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://mvnrepository.com/

[24] Cobertura. [online], 2017, [cit. 2017-04-07]. Dostupné z: http://cobertura.github.io/cobertura/

[25] RabbitMQ by Pivotal. [online], 2011, [cit. 2017-04-14]. Dostupné z: https://www.rabbitmq.com/

[26] The Java™ Tutorials. [online], 2015, [cit. 2017-04-14]. Dostupné z: https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html

[27] Google Guice. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://github.com/google/guice/

[28] Hibernate. [online], 2017, [cit. 2017-04-07]. Dostupné z: http://hibernate.org/

[29] Apache Commons. [online], 2017, [cit. 2017-04-07]. Dostupné z: https://commons.apache.org/

[30] The Java™ Tutorials: Exceptions. [online], 2015, [cit. 2017-04-14]. Dostupné z: https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

[31] Scott, M. L.: *Programming language pragmatics*. Morgan Kaufmann Pub., druhé vydání, c2006, ISBN 0126339511.

[32] Java™ PlatformStandard Ed. 8. [online], 2016, [cit. 2017-04-14]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html

[33] Java™ PlatformStandard Ed. 8. [online], 2016, [cit. 2017-04-14]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html

[34] DataDog Docs. [online], 2017, [cit. 2017-04-14]. Dostupné z: https://docs.datadoghq.com/guides/basic_agent_usage/

[35] Git –fast-version-control. [online], 2017, [cit. 2017-04-14]. Dostupné z: https://git-scm.com/

[36] Huizinga, D.; Kolawa, A.: *Automated defect prevention: best practices in software management*. IEEE Computer Society, c2007, ISBN 9780470042120.

[37] Continuous Integration. [online], 2006, [cit. 2017-02-12]. Dostupné z: https://www.martinfowler.com/articles/continuousIntegration.html

[38] Standard - ECMA - 404: The JSON Data Interchange Format. 2013: s. 1–14, [cit. 2017-04-14]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[39] Introducing JSON. [online], [cit. 2017-04-14]. Dostupné z: http://www.json.org/

[40] Beck, K.: *Test-driven development: by example*. Addison-Wesley, c2003, ISBN 0321146530.

[41] Fowler, M.: *Refaktoring: zlepšení existujícího kódu*. Moderní programování, Grada, 2003, ISBN 8024702991.

Příloha A

# Seznam pojmů

## A.1 Web API

Web Aplication Programming Interface je rozhraní, které na definovaný HTTP dotaz vrátí požadované data. Data jsou standardně vracena ve formátech JSON nebo XML.

## A.2 Verzovací systém Git

Git[35] je verzovací systém umožňující vytváření a sdílení jednotlivých verzí projektu. Umožňuje jednoduchý přehled nad rozpracovanými částmi každého vývojáře. Úložiště systému se nazývá repozitář, který obsahuje veškerý kód.

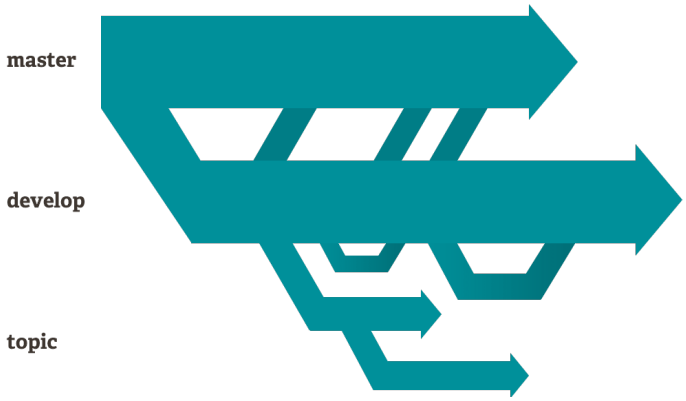
Repozitář existuje v lokálních verzi a zároveň serverové, tedy sdílené. Sdílený repozitář zajišťuje distribuci aktuální verze do lokálních repozitářů. Pro lepší správu existují nadstavby nad serverovou částí repozitáře, které umožňují jednoduchou správu nad kódem a spouštění dalších služeb v závislosti na změnách v kódu.

Základní jednotku tvoří verze, které jsou postupně vytvářeny vývojáři na základě provedených změn. Verze jsou uchovávány v jednotlivých větvích programu.

Repozitář je obvykle rozdělen na hlavní a vedlejší větve. Vedlejší větve slouží pro samotný vývoj. Hlavní větve tento kód spojují a reprezentují aktuální vývojovou a produkční verzi. Git také obsahuje nástroj pro slučování větví.

## A.3 Jednotkové a integrační testy

Jednotkovými testy se rozumí sada kladných a záporných testů ověřující funkcionalitu jediné třídy. Jednotkové testy jsou nezávislé na ostatních třídách a testech.[36]




Obrázek A.1: Zobrazení větví v repozitáři, kde *master* je produkční, *develop* vývojová a *topic* představuje větev vedlejší

Integrační testy pokrývají komunikaci více tříd nebo komunikaci s operačním systémem, hardwarem či rozhraním různých systémů.[36]

Důvody pro psaní testů jsou například jednodušší nalezení chyby nebo lepší udržovatelnost projektu. V případě neexistujících testů nelze ověřit původní funkcionalitu při modifikaci aplikace, což může způsobit nutnost nejprve chyby před opravou nalézt.[36]

## A.4 Statická analýza kódu

Statická analýza kódu je analýza softwarového produktu, která běží bez spuštění samotné aplikace. Kontroluje pouze kód. Označuje kritické konstrukce vedoucí k chybám nebo nedodržení programátorských konvencí daného jazyka.

## A.5 Průběžná integrace

Průběžnou integrací se rozumí sada nástrojů sloužící k urychlení softwarového vývoje. Základem je průběžné sestavení a spouštění testů aplikace na základě změn ve sdíleném repozitáři. Lze tak rychle odhalit případné chyby před zařazením příslušné verze do produkce.[37]

## A.6 Sdílení dat pomocí front

Sdílení dat pomocí front funguje na principu odesílání zpráv reprezentující objekty. Zprávy jsou po zařazení producentem do fronty odebírány konzumenty, které je zpracovávají. Příklad implementace takového systému pak může být RabbitMQ.[25]

## A.7 JSON

JSON označuje specifikaci formátu pro výměnu dat[38]. Jedná se o formát, který je čitelný nejenom pro lidské oko, ale i pro stroj[38], Zpracování toho formátu je implementováno pro většinu programovacích jazyků[39]. Skládá se z párů označující klíč a hodnotu. Hodnota může být řetězec, číslo nebo pole. Pole pak může uchovávat opět pole, řetězec nebo číslo.[38]

Listing A.1: Ukázka formátu JSON

```
{
  "key": [
    1,
    2,
    3
  ],
  "boolean": true,
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d",
    "e": "f"
  },
  "string": "Hello World"
}
```

## A.8 Mock

V objektově orientovaném programování se Mock objekt používá pro simulování chování konkrétní třídy.[40] Při testování je tak možné docílit takových testů, které nejsou závislé na ostatních třídách, kromě té přímo testované.

Testovaná třída obvykle vyžaduje závislost na jiných třídách či rozhraní. Pomocí Mocku je možné chování těchto tříd simulovat. Mimo nadefinování požadovaného chování, lze také na Mock objektu sledovat jaká na něm byla provedena volání, včetně toho s jakými parametry. Díky tomu je možné testovat i vnitřní chování testované třídy a ne pouze návratovou hodnotu na základě obdrženého vstupu.[40]

## A.9 Refaktorování kódu

Refaktorování v softwarovém vývoji chápeme jako proces restrukturalizace existující kódu, aniž by byla pozměněna funkcionalita. Provádí se za účelem dosáhnout průhlednějšího a čitelnějšího kódu, který se lépe udržuje a

rozšiřuje.[41] Hlavní spouštěcí příčina refaktorování kódu je však existence konstrukcí značící špatný návrh aplikace.

V kontextu této práce jsou důležité především následující konstrukce značící možné problémy[41]:

- Dlouhá metoda.
- Velká třída.
- Dlouhý seznam parametrů.
- Složité struktury podmínek.

Příloha B

# Seznam použitých zkratek

**EAN** European Article Number,

**XML** Extensible markup language,

**HTML** Hypertext Markup Language,

**CSS** Cascading style sheets,

**JSON** JavaScript Object Notation,

**HTTP** Hypertext Transfer Protocol,

**DAO** Data Access Object,

**URL** Uniform Resource Locator.

Příloha C

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `build` ..... adresář se sestavenými verzemi systému
  - `backend` ..... adresář obsahující sestavenou interní část
  - `backend-old` ..... adresář obsahující ses. interní část původního řešení
- `docs` ..... dokumentace systémuom
  - `backend` ..... dokumentace interní částí
  - `frontend` ..... dokumentace webového rozhraní
- `src` ..... zdrojové kódy implementace
  - `backend` ..... zdrojové kódy interní částí
  - `backend-old` ..... zdrojové kódy interní částí původního řešení
  - `frontend` ..... zdrojové kódy webového rozhraní
  - `frontend-old` ..... zdrojové kódy webového rozhraní původního řešení
- `text` ..... zdrojová forma práce ve formátu LaTeX
  - `BP_Tucek_Jakub_2017.pdf` ..... text práce ve formátu PDF