

Sem vložte zadání Vaší práce.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
KATEDRA . . . (SOFTWAREVÉHO INŽENÝRSTVÍ)



Bakalářská práce

InfoWeb - Nástroj získávání informací z webů

Vedoucí práce: Ing. Jiří Hunka

8. 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 8. května 2017

.....

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2017 Jakub Tuček. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

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 předcházela týmový projekt z předmětů BI-SP1 a BI-SP2 je popsána 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 needs is for example trends of product prices sold by competitors. Because issue was already addressed in team project implemented in courses BI-SP1 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 team project along with implementation of most important changes. Created system is evaluated based on thorough testing. Finally, additional improvements are designed for possible future.

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 Stav	21
4.2 Nedostatky	22
4.3 Shrnutí	26
5 Návrhy na vylepšení	29
5.1 Refaktorování stávajícího řešení	29
5.2 Plánování práce	30

5.3	Oprava komunikace Manager - ProductProvider	30
5.4	Spojení chyb analyzátoru a vylepšení rozhraní	31
5.5	Monitorování	31
5.6	Získání adres obchodů a příslušných detailů produktů	32
5.7	Párování produktu	32
5.8	Neimplementované návrhy	33
6	Realizace vylepšení	35
6.1	Refaktorování stávajícího řešení	35
6.2	Plánování práce	42
6.3	Oprava komunikace Manager - ProductProvider	42
6.4	Spojení chyb analyzátoru a vylepšení rozhraní	46
6.5	Monitorování	46
6.6	Získání adres obchodů a příslušných detailů produktů	48
6.7	Párování produktu	48
6.8	Detekce neexistující stránky a nenalezeného produktu	49
6.9	Více šablon detailů produktu	50
6.10	Více stejných chyb	51
6.11	Skladem	51
6.12	Ostatní	51
7	Zhodnocení provedených vylepšení	53
7.1	Funkcionalita	53
7.2	Kampaň	53
7.3	Párování	54
7.4	Webové rozhraní	55
7.5	Návrh a testy	55
7.6	Rozhraní administrátora	56
7.7	Nemožnost vyhledávání na některých obchodech	56
8	Závěr	57
	Literatura	59
A	Seznam pojmů	63
A.1	Web API	63
A.2	Verzovací systém Git	63
A.3	Jednotkové a integrační testy	63
A.4	Statická analýza kódu	64
A.5	Průběžná integrace	64
A.6	Sdílení dat pomocí front	64
A.7	JSON	65
A.8	Mock	65
A.9	Refaktorování kódu	65

B Seznam použitých zkratk	67
C Obsah přiloženého CD	69

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	22
4.2	Pokrytí testy vytvořeného řešení	26
6.1	Diagram tříd analyzátoru	41
6.2	Webové rozhraní pro vyřešení chyby analyzátoru po provedení vy- lepšení	47
6.3	Aktivita diagram nalezení adres detailů produktů	49
6.4	Administrátorské rozhraní pro opravu detailu šablony	50
7.1	Pokrytí testy po provedených vylepšení	55
A.1	Větve v Git repozitáři	64

Seznam tabulek

4.1	První část tabulky přehledu nedostatků vytvořeného týmového řešení	27
4.2	Druhá část tabulky přehledu nedostatků vytvořeného týmového řešení	28
7.1	Tabulka testovaných produktů	54

Úvod

Práce uvádí čtenáře do problematiky získávání informací z webů s důrazem na požadavky internetových obchodů popřípadě distributorům zboží, konkrétně sledováním vývoje cen prodáváných 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.

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 specifitě dat, která jsou v kontextu činnosti zajímavá a kvůli unikátnosti webových stránek není možné jednoznačně určit jednotný a zcela automatizovaný postup, jak data získat v požadovaném formátu.

1.2 Výběr dat

Nejčastější 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 zajímavá.

Získávání informací ze stažených stránek lze zjednodušit na problematiku určení elementů v HTML. Element pak může obsahovat pouze požadovanou část, 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 použít XPath 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 být vždy unikátní, ale řetězení selektorů je možné element jednoznačně v HTML dokumentu vybrat.

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 zajímaví 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 prodávány 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 *heureka* 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 také 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 formou manuální či vstupem ze souboru nebo přímých napojením na e-shop. Následně je možné konkrétní data zobrazit v grafech označující vývoj cen, trendů či náhlých změn. Dále umožňují externí výstup do souboru v dostupných formátech.

Největší rozdíl služeb je, zda jsou data získávána přímo z obchodů nebo ze srovnávačů. Další odlišností je možnost, jestli služba dokáže sledovat i zahraniční trh.

Cena služeb se nejvíce 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

1. POPIS PROBLEMATIKY ZÍSKÁVÁNÍ INFORMACÍ Z WEBŮ



Shop	# Prices	± Prices	# Null Prices	± Null Prices	# Empty producers	± Empty producers
Czech Republic - Electro	1320	-1 (-0 %)	0 (0 %)	0 (N/A %)	1	35 (0 %)
	2959	-10 (-0 %)	0 (0 %)	0 (N/A %)	2	48 (0 %)
	474	0 (0 %)	0 (0 %)	0 (N/A %)	1	22 (0 %)
	670	0 (0 %)	0 (0 %)	0 (N/A %)	2	27 (0 %)
	1414	-3 (-0 %)	0 (0 %)	0 (N/A %)	0	37 (N/A %)
	3244	1 (0 %)	61 (2 %)	-2 (-3 %)	0	22 (N/A %)
	3961	25 (1 %)	0 (0 %)	0 (N/A %)	0	24 (N/A %)
	6746	-9 (-0 %)	0 (0 %)	0 (N/A %)	0	51 (N/A %)
	24025	45 (0 %)	3 (0 %)	2 (200 %)	8	448 (-11 %)
	680	16 (2 %)	0 (0 %)	0 (N/A %)	1	19 (N/A %)
	7377	95 (1 %)	0 (0 %)	0 (N/A %)	6	145 (20 %)
	4140	21 (1 %)	60 (1 %)	8 (15 %)	0	29 (N/A %)
	3368	-2 (-0 %)	3 (0 %)	0 (0 %)	0	21 (N/A %)
	11573	-2909 (-20 %)	0 (0 %)	0 (N/A %)	15	89 (400 %)
	260	0 (0 %)	0 (0 %)	0 (N/A %)	0	1 (N/A %)
	6440	3 (0 %)	32 (0 %)	1 (3 %)	1	47 (0 %)
	13878	3953 (40 %)	0 (0 %)	0 (N/A %)	2	59 (-33 %)
	13759	-1 (-0 %)	124 (1 %)	119 (2380 %)	8	175 (0 %)
	5960	-1473 (-20 %)	0 (0 %)	0 (N/A %)	67	216 (644 %)
	10421	-186 (-2 %)	0 (0 %)	0 (N/A %)	0	73 (N/A %)
	2476	37 (2 %)	0 (0 %)	0 (N/A %)	0	38 (N/A %)
	37549	-111 (-0 %)	3 (0 %)	-1 (-25 %)	0	143 (N/A %)
	13906	37 (0 %)	10 (0 %)	1 (11 %)	5	130 (-17 %)
	7321	-9 (-0 %)	0 (0 %)	0 (N/A %)	0	53 (N/A %)
	16016	7 (0 %)	0 (0 %)	0 (N/A %)	0	83 (N/A %)
	9193	-3527 (-28 %)	0 (0 %)	-2 (-100 %)	10	128 (-17 %)
	6666	-295 (-4 %)	2 (0 %)	0 (0 %)	0	35 (N/A %)
	5072	269 (6 %)	0 (0 %)	0 (N/A %)	3	47 (0 %)
	4356	-19 (-0 %)	17 (0 %)	0 (0 %)	4	136 (0 %)

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

- nespécifikované 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, pricmania.sk, cen-neo.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

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 funkcionality 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 nejdříve v BI-SP1 vytvořen návrh systému a v BI-SP2 následně implementován.

Cílem tohoto projektu byla maximální možná míra automatizace získávání informací o produktech prodáváných konkurencí. Důraz byl především kladen na optimalizaci počtu nutných lidských úkonů. Byl navržen způsob, kdy systém při pouze nezbytných krocích spoléhá na administrátora. U administrátora se předpokládá technická zdatnost průměrného uživatele.

2.2 Požadovaná funkcionality

Požadovaný stav projektu umožňuje uživateli vložit produkty do systému ve formátu *csv* 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 sledované 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 komunikujících spolu pomocí front. Díky tomu, že každý modul zajišťuje určitou funkcionalitu a je možné vytvářet více jeho 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(klienti).

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 přístup pouze lokální. Moduly jsou detailněji popsány v následujících podsekcích.

2.5.1 Manager

Manager je hlavní modul, který má jako jediný interní část možnost připojení do databáze a 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 požadavků.

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 prodáváných 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

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 vyučujícím. Funkcionalita se skládala z jednotlivých úkolů rozložených do sprintů. Úkoly byly přiděleny jednotlivým členům týmu. Stav úkolů uchovával systém Redmine[16], který umožňoval sledovat jejich stav. Úkoly bylo možné 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 spouštění služeb na základě definovaných aktivit v repozitáři. Repozitář se skládal ze 4 částí (větví):

- Master - hlavní větev uchováající 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 přístup k modifikacím větví Master a Develop měl pouze vedoucí projektu, musel být pro každou Feature a Fix větev vytvořen požadavek o zařazení (Merge request). Po kontrole vedoucím byl požadavek zařazen nebo vrácen k opravě.

Na konci každé iterace byla poslední verze označena pomocí *tagu* a poté prezentována vedoucím. Označení bylo zvoleno na základě pořadí iterace. 1. iterace je označena verzí „0.1“.

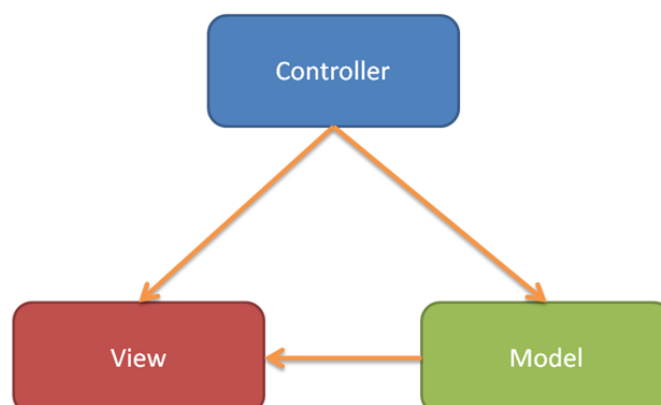
Pro vývoj se využil princip průběžné integrace. Každá verze byla zkompilována, otestována a zanalyzována na vzdáleném serveru. Tyto činnosti zajišťovaly systémy Jenkins[18] a Gitlab. Po změně v repozitáři byl spuštěn úkol v Jenkins. Ten aplikaci sestavil, spustil testy a statickou analýzu kódu zajištěnou 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 vrstvu *view* šablony. Šablony definují 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 formuláře. *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 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 větvi 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 posílány 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 řetězci 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

3. VÝVOJ A IMPLEMENTACE TÝMOVÉHO PROJEKTU

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. Doplnky v Jenkins umožňovaly zobrazení přehledných výsledků, jak je kód pokryt testy 4.2. Přesné pokrytí testů mi poté umožňovalo jednoduše kontrolovat jaké části kódu jsou otestované.

Zhodnocení týmového projektu

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

4.1 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.

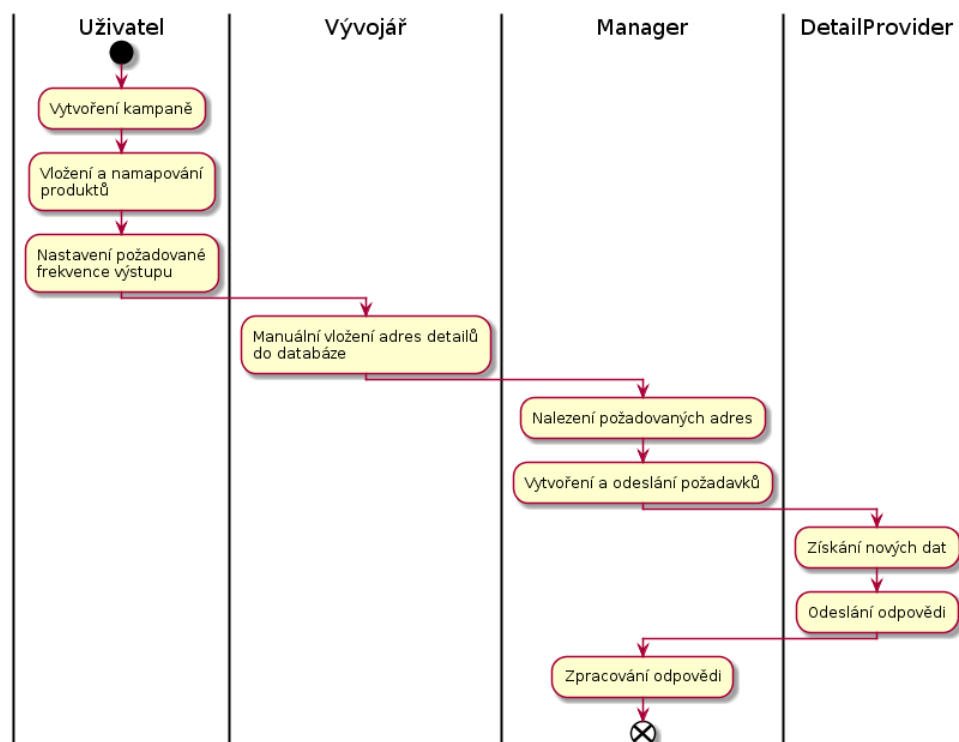
Uživatelská část umožňovala správu a vytvoření kampaní. Splňovala návrh z analytické části 2.4.1. Na žádost jiného týmu bylo také vytvořeno Web API rozhraní, poskytující získané ceny pro daný produkt ve formátu JSON.

Interní část byla schopná pracovat pouze na základě dříve uložených url adres vedoucí na detaily produktů. Manager dokázal vybrat adresy detailů, které je potřeba aktualizovat.

Na základě adres vytvořil Manager požadavky obsahující potřebná data. Požadavky odeslal pomocí front pro zpracování do DataProvideru. Ten na základě obdržených dat stránku stáhnul, vyparsoval a zanalyzoval výsledek vůči historickým datům a odeslal informace o produktu, který se na adrese nachází.

Analýza se skládala z kontrol velkých výkyvů cen a rozdílných identifikátorů produktu. V případě, že požadavek neobsahoval šablonu pro vyparsování nebo byl výsledek označen jako chybný, tato chyba byla odeslána zpět do Manageru, kde se výsledek uložil pro zobrazení ve webovém rozhraní, ať už se jednalo o chybu nebo nalezení ceny. Vyřešení chyby Manager detekoval a adresu využil k vytvoření nového požadavku.

4. ZHODNOCENÍ TÝMOVÉHO PROJEKTU



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

4.2 Nedostatky

Vytvořené řešení obsahovalo spoustu nedostatků, které bylo třeba v rámci této práce detekovat a ty nejdůležitější se pokusit opravit. Z důvodu kontextu a odlišnosti od původního návrhu nejprve popíšu systém plánování práce, který byl častý důvod nežádoucího chování systému.

4.2.1 Vytváření požadavků pro ProductProvider

Některé nedostatky byly úzce spjaty s tím, jak aplikace plánovala práci. Plánováním práce je myšlen proces nalezení adres detailů produktů, kde je požadováno získat nové data. Z nich jsou vytvořeny požadavky pro ProductProvider k zpracování.

Jak bylo řečeno, 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 poté vybráno prvních 10 adres, odebrány již odeslané a ty které mají nevyřešenou chybu. Opakované spuštění v předdefinovaném intervalu zajistilo, že požadavky byly vytvořeny 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, že byly zpracovány.

V případě neúspěchu při odeslání požadavku už však příznak v případě chyb nebyl změněn. To způsobilo, že chyba nebyla v příštích hledání znovu zpracována.

4.2.2 Neefektivní chování modulu Manager a ProductProvider

První zásadní problém bylo neefektivní chování komunikace modulu Manager s ProductProviderem. V rámci testování jsem zjistil, že v případě chyby při parsování stránky není použit uložený HTML dokument. To pramenilo z výše popsaného návrhu plánování, které našlo korektně adresy, ale vytváření samotného objektu představující požadavek, bylo stejné pro všechny adresy. Vytváření neobsahovalo použití již staženého dokumentu, na jehož základě byla vytvořena chyba parsování nebo analyzování. Každý požadavek vyústil v opětovné stažení příslušné stránky.

4.2.3 Chyby analyzování

Poslední fáze procesu v modulu ProductProvider byla navržena jako analyzování získaných dat vůči již dříve uloženým. Implementace analyzátoru spouštěla jednotlivé validace, jejichž 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 označena, že jsou pravděpodobně chybná. Dále probíhala kontrola získané ceny „s“ a „bez“ DPH oproti cenám získaných na konkrétní stránce dříve.

Kontrola porovnávala průměr historických hodnot se získanými hodnotami cen. V případě, že rozdíl byl větší jak 25%, byl výsledek označen jako možná chyba.

Pokud validační třída objevila nežádoucí data, vyhodila výjimku, ve které byly uloženy informace o chybě. Tento způsob řízení programu však způsoboval, že celá validace skončila při první chybě. Zároveň obsah výjimky pouze

určoval typ validace bez přídavných informací. Informace byly následně odeslány a Manager na jejich základě vytvořil chybu pro administrátora k vyřešení. Administrátor tak mohl označit, zda je to opravdu chyba nebo se toto hlášení má do budoucna ignorovat.

Problém nastával pokud se na adrese objevilo více chyb. To znamenalo, že každý požadavek vytvořil další chybu analyzátoru a administrátor musel všechny vyřešit. Při každém požadavku pak bylo nutné obsah adresy znova stáhnout.

4.2.4 Vytváření chyb šablon

Jako další problém 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. To vyústilo ve vytvoření mnoha chyb, které musel administrátor postupně vyřešit.

4.2.5 Modul Finder

Modul Finder nebyl zapojený do systému. Existovala pouze hlavní implementace interních procesů, jejichž 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 funkcionality 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 však jedná o správný detail produktu je však třeba ověřit, jelikož výstupem může být více adres. Po získání hodnot ze stránky se musí nežádoucí adresy vyloučit a ostatní spárovat s produktem.

4.2.6 Plánování práce

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

4.2.7 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.

4.2.8 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 `DataProviderServiceImpl` zajišťující celý proces v modulu `DataProvider`.

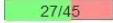
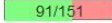
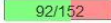
Je nutné podotknout, že spousta špatných konstrukcí byla eliminována již v průběhu vývoje týmového projektu. První důvod byla statická analýza kódu detekující konstrukce, které jsou zdrojem častých chyb. Statická analýza však nebyla schopná najít všechny problémy. Druhý důvod byla moje kontrola při schvalování vytvořeného kódu pro zadaný úkol. Z důvodu časové tísně, ale nebyl vždy prostor na to vrátit kód k přepsání 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.

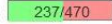

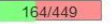
Další problém návrhu byly procesy v `DataProvider` modulu řízené pomocí výjimek obsahující přídavné informace. A to i v případech, kdy byl takový výsledek očekávaný nebo dokonce chtěný. Toto použití je však v rozporu ideou použití výjimek, které mají signalizovat neočekávaný stav, kdy není možné dále pokračovat.[30] Pro uchování přídavných informací bylo nutné vytvářet výjimky vlastní, které obsahují přesnější údaje o chybě. Ani tyto údaje však nebyly dostatečné a tak nebylo možné přesně rozpoznat ve webovém rozhraní přesný důvod chyby.

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 analyzátozem. 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.

Systém proto v tomto stavu nebyl vhodný pro následný rozvoj.

4. ZHODNOCENÍ TÝMOVÉHO PROJEKTU

Name	Packages	Files	Classes
Cobertura Coverage Report	60%  27/45	60%  91/151	61%  92/152

Methods	Lines	Conditionals
50%  237/470	50%  1026/2038	37%  164/449

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

4.3 Shrnutí

Jako metriky zhodnocení byly zvoleny následující kritéria seřazené od nejpodstatnějších k nejméně důležitým:

- Kritické chyby
- Úplnost požadované funkcionality
- Rozšiřitelnost (návrh, pokrytí testy, statická analýza kódu)
- Neefektivní chování
- Uživatelská přívětivost
- Škálovatelnost

Rozšiřitelnost je hodnocena na základě návrhu, pokrytí testy a počtu chyb statické analýzy kódu. Návrh byl podrobně zmíněn v kapitole 4.2.8.

Na základě shrnutí nedostatků v tabulkách 4.3 a 4.3 je možné prohlásit, že pro základní použití systému je potřeba opravit kritické chyby a doplnění požadované funkcionality. Pro pohodlné použití je však zapracovat i na zbylých požadavcích, jinak je možné, že systém bude systém z hlediska administrátorského rozhraní velmi nepřívětivý. Neefektivní chování se může negativně odrazit při velké zátěži. Škálovatelnost má v tomto případě prioritu nejmenší.

Tabulka 4.1: První část tabulky přehledu nedostatků vytvořeného týmového řešení

Typ	Počet	Popis
Kritické chyby	2	<ul style="list-style-type: none">• V případě chybného odeslání není chyba šablony nebo analyzátoru znova zpracována.
Úplnost funkcionality	4	<ul style="list-style-type: none">• Systém nevyhledává internetové obchody.• Systém nevyhledává adresy detailů produktů.• Systém neplánuje práci v požadovaných intervalech.• Neexistence procesu párování.
Rozšiřitelnost		<ul style="list-style-type: none">• Návrh tříd byl vyhodnocen jako nevhodný pro rozšiřování.• Pokrytí testy dle nástroje cobertura. - 60%.• 3 kritické chyby statické analýzy kódu.• 1 minoritní chyba statické analýzy kódu.

4. ZHODNOCENÍ TÝMOVÉHO PROJEKTU

Tabulka 4.2: Druhá část tabulky přehledu nedostatků vytvořeného týmového řešení

Typ	Počet	Popis
Neefektivní chování	4	<ul style="list-style-type: none">• Stránka je nově stažena v každém požadavku (i v případě chyby).• V případě chybného odeslání není chyba šablony nebo analyzátoru znova zpracována.• Každá chyba analyzátoru znamená samostatný požadavek a chybu.• Vytvoření více požadavků pokud neexistuje šablona na obchodě.
Uživatelská přívětivost	2	<ul style="list-style-type: none">• Nutnost řešit chyby analyzátoru po jedné.• Řešené chyby neobsahují bližší informace (informace o chybě, vyparsované hodnoty, použitá adresa detailu).
Škálovatelnost	1	<ul style="list-style-type: none">• Nemožnost vytvořit více instancí modulů.

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.

5.1 Refaktorování stávajícího řešení

Problémy původního řešení je třeba opravit. Na existujícím kódu však 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 následně 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. Jednotlivé funkcionality budou poté v samostatných balíčcích.

Při úpravách týkající se rozdělování jednotlivých tříd, 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 poté ulehčí běh 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. Není také implementováno odstranění adres u kterého není vyžadováno vytvářet a odesílat nový požadavek.

Nový návrh hledá 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 jsou vyřazeny adresy, které z nějakého důvodu nevyhovují svým stavem. Nežádoucí stavy jsou momentálně tyto:

- Pro obchod existuje nevyřešená parsovací chyba
- 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 popsáných v kapitole 4.2.2 je požadováno komunikaci modulů Manager a ProductProvider optimalizovat. Neefektivní chování by se mohl 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 a následně využít staženou stránku. Hledání by tak probíhalo ve většině případů zbytečně, jelikož chyba by neexistovala.

Korektní oprava je úzce spjata s předchozími kapitolami, především s re-faktorováním a úpravou plánování práce. Pokud úprava plánování zachová informaci o příčině udávající, proč je adresa pro vytvoření požadavku zařazena, stačí poté pouze nalézt potřebné atributy a ty uložit.

Informace by však měla být zachována i při odesílání, kdy se nastaví i příznaky o odeslání a zpracování šablony. V případě kdy se požadavek nepodaří odeslat musí příznak odpovídat tomuto stavu. Z hlediska DataProvider stačí pouze zkontrolovat, zda existuje stažená stránka v požadavku a v kladném případě ji nestahovat znova.

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. Toto rozhodnutí je však založené pouze na informaci obsahující jaká validace se nepovedla. Nemá možnost porovnat jaká data byla využita při validaci. Webové rozhraní aktuálně nemůže možnost zobrazit dodatečná data, protože nejsou k dispozici. Interní část by měla takové informace získávat a uložit do databáze. Následně může webové rozhraní tyto informace použít.

V případě adresy, která obsahuje více chyb, je nutné řešit každou samostatně. Po každém vyřešení se musí počkat na zpracování interní částí. Pro zrychlení je tak nutné požadováno chyby spojit do jedné, tak aby administrátor mohl vyřešit všechny možné 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 tedy vhodné zvolit monitorovací službu, která umožňuje unifikovat sledování probíhajících procesů na serveru, včetně vytížení a zobrazuje stav na jedné stránce.

5.6 Získání adres obchodů a příslušných detailů produktů

Interní část vyžaduje ke své funkcionalitě již uložené adresy detailů produktů, ze kterých jsou získávána data pro produkty. Původní návrh počítal s modulem Finder, který se bohužel nepodařilo zapojit v rámci týmového projektu. Ten měl za úkol hledat internetové obchody na cenových srovnávačích a na nich pomocí vyhledávání 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 požadavků 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.

5.7 Párování produktu

Po nalezení adresy detailu produktu, 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 přesné shody některého z identifikátorů, 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 i 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, tedy akcí administrátora. Výše uvedený proces poté spoléhá na to, že vložená data při vytváření kampaně jsou validní. V případě nevalidních dat utřeba příliš obecných a krátkých názvů, by párování proběhlo chybně.

5.8 Neimplementované návrhy

5.8.1 Pokročilé párování produktu

Párování produktu lze vylepšit o uchovávání více hodnot pro identifikátory produktů, které systém může použít při dalším párování na jiných obchodech. Ukládání nových identifikátorů by mělo probíhat pouze se souhlasem administrátora tak, aby byla zajištěna validita dat.

5.8.2 Uchování a využití hodnot z nespárovaných adres

Vyhledáváním na obchodu je zpravidla výsledek, který obsahuje více adres detailů produktu. Většina jich je v době hledání nepotřebná, nicméně v budoucnu by mohly být využity. Pro dlouhodobě efektivnější chod systému se jeví praktické uchovávat získaná data z těchto detailů. V případě přidání nových produktů do systému se poté pokusit najít shodu v těchto datech. To umožní odlehčení zátěže na stahování stránek a celkově zrychlí chod systému.

Adresy však mohou být neaktuální. Zde je proto nutné nastavit mechanismus ověření funkčnosti adres detailů produktů.

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 označeny. Původní verze týmového projektu byla označena jako tag *v0.53*, realizované vylepšení jsou poté 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, zkrácení metod, zmenšení počtu parametrů metod a nahrazení výjimek za návratové typy.

6.1.1 Servisní třídy

Pro větší přehlednost byly všechny servisní třídy přesunuty do nadřazeném balíčku *service*. Servisní třídy jsou takové, které nespádají 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 jakého modulu 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*. Uvedené názvy jsou včetně nadřazených balíčků, kdy například název třídy je v prvním případě pouze *ParserImpl*.

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 další příslušné informace.

Návratový typ lze demonstrovat na následující zkrácené třídě *DPParserResponse*, 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
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 }
```

Tato struktura je použita jako návratový typ rozhraní a implementace části pro parsování hodnot ze stažené stránky v modulu `DataProvider`. Ukazuje použití příznaku označující, zda parsování proběhlo korektně. Další důležitý prvek je zapouzdření proměnných uchovávající data.[31] 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 je proto vyžadováno se objektu zeptat na jeho stav. Očividnost prázdnoty návratové hodnoty nutí vývojáře s touto možností počítat, čímž je omezen 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 změnách tříd 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 spustila. Skončila však vždy při první chybě, což je jedna z příčin chování, které jsem popsal v kapitole poukazující na možnosti spojení chyb analyzátoru section 5.4.

Vytvořil jsem nový návrh, který je použit i na ostatních místech interní části v závislosti na prováděných vylepšeních. Analyzující servisní třídě jsem odebral jednotlivé závislosti na konkrétních validacích a nahradil je množinou obsahující validační rozhraní. Validační rozhraní je nastaveno v konfiguračním souboru automatické správy závislostí, kde je možné definovat jaké validace se mají použít.

Validačnímu rozhraní byl změněn návratový typ na *Optional* třídy obsahující zprávu o chybě. Pokud chyba nenastala, je vrácena prázdná hodnota.

Implementace těchto rozhraní byly rozděleny podle toho, jakou hodnotu kontrolují. Při úpravě validací jsem zároveň 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 kontroluje, zda získaná hodnota odpovídá již uložené hodnotě, vypadá následovně:

Listing 6.1: Validace kontrolující hodnotu získaného jména produktu

```
1  /**
2   * NameAnalysis is extension of {@link AbstractStringAnalysis} for
3   * analysing Name.
4   *
5   * @author Jakub Tucek
6   * @created 27.1.2017
7   */
8  public class NameAnalysis extends AbstractStringAnalysis {
9
10     /**
11      * {@inheritDoc}
12      */
13     @Override
14     boolean skipAnalysis(DataProviderRequest request) {
15         Optional<ComAnalyserFlags> analyserFlags =
16             request.getAnalyserFlags();
17         return
18             analyserFlags.map(ComAnalyserFlags::isIgnoreDifferentName)
19                 .orElse(false);
20     }
21
22     /**
23      * {@inheritDoc}
24      */
25     @Override
26     Optional<String> getOptionalProperty(DPParserResponse
27         parserResponse) {
28         return parserResponse.getName();
29     }
30
31     /**
32      * {@inheritDoc}
33      */
34     @Override
35     List<String> getComProductValues(ComProduct comProduct) {
36         return comProduct.getNames();
37     }
38 }
```

```

33     }
34
35     /**
36      * {@link Doc}
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]",
                        parsedValue, comProductValue)
43             )
44             .withErrorType(AnalysisErrorType.DIFFERENT_NAME);
45     }
46 }
47

```

Jednotlivé validace implementují pouze metody, které vrací jaká hodnota byla získaná a historické hodnoty. Dále kontroluje zda danou validaci ignorovat či nikoliv. Poslední implementovaná metoda vytváří informace o případné chybě. Tuto informaci je poté možné využít v zobrazení administračním rozhraní. V původním řešení tato možnost chyběla, administrátor musel o chybě rozhodnout pouze na základě typu chyby a adresy vedoucí na detail produktu. K tomuto vylepšení se vracím v samostatné kapitole.

Konečně spuštění validací bylo ve výsledku zkráceno na metodu obsahující jeden řádek kódu, ačkoliv tento řádek obsahuje více zřetěžených 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)

```

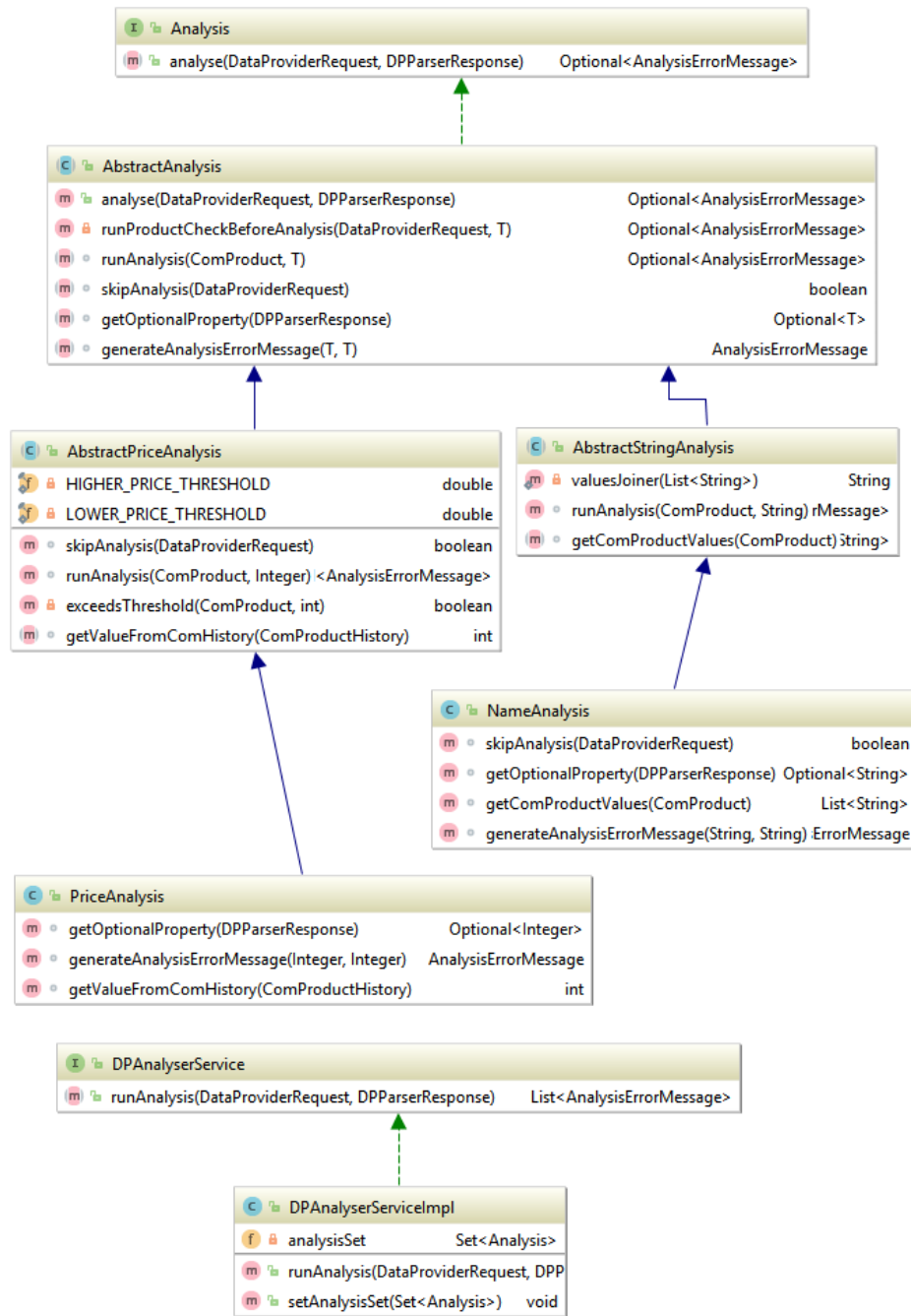
6. REALIZACE VYLEPŠENÍ

```
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
8   *                       AnalyserException when analysing fails, contains error type
9   * @param analyserFlags
10  */
11  @Override
12  public void analyse(ComProduct newInfo,
13                     ComProductHistory newData,
14                     ComProduct oldInfo,
15                     List<ComProductHistory> productHistory,
16                     ComAnalyserFlags analyserFlags) throws
17                     AnalyserException {
18      ValidatorData data = new ValidatorData(newInfo, newData,
19      oldInfo, productHistory, analyserFlags);
20      try {
21          eanValidator.validate(data);
22          partNumberValidator.validate(data);
23          priceValidator.validate(data);
24          namesValidator.validate(data);
25      } catch (AnalyserException e) {
26          logger.info("Analysis failed for product id: {}",
27                      newInfo.getProductId(), e);
28          throw e;
29      }
30      if (!data.getWarnings().isEmpty()) {
31          //No handling needed at the moment
32      }
33  }
```

Jak již bylo zmíněno, podobná architektura spouštění validací se vyskytuje na více místech interní části, například validace po provedení parsování nebo kontrol, zda má být požadavek či adresa použita při plánování práce.



Powered by yFiles

Obrázek 6.1: Diagram tříd analyzátoru. Obsahuje pouze některé validace z důvodu přehlednosti.

6.2 Plánování práce

Plánování práce bylo rozděleno do tří částí, Nalezení adres, vytvoření požadavků a samotné odeslání. Tato kapitola se týká pouze samotného nalezení adres, které je požadováno použít v dalším zpracování.

V rámci hledání samotných adres byly implementovány servisní třídy, které samotné adresy hledají podle kritérií popsaných v návrhu vylepšeníh. Následně jsou vráceny rozdělené dle způsobu nalezení. Nalezení adres nově počítá i s frekvencí, která je nastavena u aktivní 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 k změně rozhraní, které zachovává způsob nalezení, stačilo vytvořit ostatní části Managera, tak aby odpovídalo novému rozhraní.

Hlavní metoda servisní třídy *DPRequestSenderServiceImpl* volající všechny tři 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 =
9          prioritizeService.findRequiredProductUrls();
10
11     DPRequestProductUrlWrapper dpRequestWrapper =
12         requestBuilderService.create(
13             requiredProductUrls);
14
15     senderService.sendDPRequests(dpRequestWrapper);
16 }
```

Nejdříve jsou nalezeny adresy, což bylo popsáno v kapitole 6.2. Po nalezení 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. Návrátový typ této třídy slučuje množiny požadavků obsahující adresy bez produktů a ty v aktivní kampani, jelikož po vytvoření požadavků není potřeba odlišné chování k těmto požadavkům. Ostatní množiny jsou zachovány. Vzhledem k potřebě pracovat s adresou detailu i při odesílání požadavků, obsahuje *DPrequestProductUrlWrapper* množiny obsahující dvojice požadavku a adresy detailu. Důvod je odpadnutí nutnosti hledání chyby podle identifikátoru, což je potřeba při ukládání příznaku chyby o zpracování.

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, která přijímá jeden vstup 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
4   *     PersistenceDPrequestFacade} and it's id is set to the
5   *     request in wrapper
6   * object. If failure while sending object through MQ occurs,
7   *     then {@link Consumer} failureHandler is called,
8   * exception logged and rethrown.
9   * Package private because of static code analysis.
10  *
11  * @param requestProductUrl wrapped object containing {@link
12  *     cvut.fit.persistence.entity.ProductUrl}, {@link
13  *     DataProviderRequest}
14  * @param persistConsumer persisting consumer that is called
15  *     before sending
16  * @param revertConsumer revert consumer that is called in case
17  *     of sending failure
18  */
```

```
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*. Tento nedostatek architektury lze pak vyřešit pomocí *IllegalStateException*, která není potomkem třídy *Exception* a lze ji v rozhraní využít.

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 popsána změna komunikační třídy *DataProviderRequest*, která byla mírně upravena. Tato komunikační třída představuje jeden požadavek odeslaný pomocí front a skládá se z jednotlivých základních atributů a fragmentů. Fragmentem je zde myšlena složitější struktura, například komunikační třída obsahující informace o produktu.

Původní návrh počítal s příznakem označující typ požadavku pro Data-Provider. Příznak označoval, zda je obsažena stažená stránka či nikoliv. Stav, kdy požadavek obsahoval stránky však nikdy nenastával z důvodu implementace plánování práce. Jelikož jediný rozdíl těchto dvou typů byl v atributu uchovávací 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 byla jasně indikovaná možnost, že hodnoty nemusí být obsaženy.

V rámci DataProvideru stačilo vytvořit při přístupu k jednomu z těchto atributů nebo fragmentu dvě možné větvení aplikace. Například pokud byla stránka obsažena v požadavku, byla vytvořena validní odpověď o stažení obsahující tuto stránku, což je demonstrováno na ukázce.

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

```
1  /**
2   * Downloads requested page and returns {@link
3   *   DownloaderResponse} object.
4   *
5   * @param dataProviderRequest the request containing url to be
6   *   downloaded
7   * @return DownloaderResponse encapsulating downloaded data or
8   *   error
9   */
10 @Override
11 public DownloaderResponse download(DataProviderRequest
12     dataProviderRequest) {
13     return dataProviderRequest.getDownloadedPage()
14         .map(x -> new
15             DownloaderResponse(Jsoup.parse(x)))
16         .orElseGet(
17             () -> doDownload(dataProviderRequest)
18         );
19 }
```

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 přesnější informace o datech použitých při parsování.

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 vyparovaných hodnot, jsem změnil návrh těchto objektů. Z *DataProviderResponseError* jsem odstranil společné atributy a jako jeho rodiče jsem zvolil přímo *DataProviderResponse*. Chybná odpověď tak nově obsahovala i všechny informace co validní odpověď.

Z hlediska *Managera* bylo potřeba tyto hodnoty nově uložit, jelikož se ukládali pouze získané ceny. Vytvořil tedy jsem novou tabulku uchovávající informace o vyparovaných hodnotách, které mohou být použity v případě zobrazení chyby administrátorovi. Dále bylo nutné uložit detailní informace o chybách, k čemuž jsem opět vytvořil novou tabulku, která je spojena vazbou 1:n s původní reprezentací chyby.

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

Poslední změna už pouze spočívala v zpracování vstupů od administrátora, kdy bylo potřeba uložit všechny možné příznaky pro budoucí analyzování. Přidal jsem také možnost přeměrovat administrátora do rozhraní, kde může změnit šablonu pro parsování stránky, neboť je možné, že analýza je chybná z důvodu změny ve struktuře stránky.

6.5 Monitorování

Na virtuální server jsem nasadil službu *DataDog* [34], která po jednoduché instalaci umožňuje sledování běžících služeb a vytížení serveru. Data jsou odesílány přímo do služby *DataDog*. Webové rozhraní umožňuje sledovat posbírané údaje.

Základní funkcionalita poskytuje pouze informace o využití prostředků a přístup k logům. Službu je však možné rozšířit o velký počet doplňků. Pomocí těch je možné sledovat například výsledek sestavení v *Jenkins*, ale i obsah a využití *RabbitMQ* front.

Chyba analyzátoru

[PŘIHLÁŠENÍ](#) | [REGISTRACE](#)

[NÁKUPNÍ KOŠÍK](#)
0 ks zboží za 0,00 Kč

[Úvodní stránka](#) | [Virtuální katalog](#) | [Vše o nákupu](#) | [Obchodní podmínky](#) | [Můj účet](#) | [Kontakt](#) | [AKCE](#)

KATEGORIE

[VLASOVÁ KOSMETIKA](#)
[KOSMETIKA PRO MUŽE](#)
[PRODLUŽOVÁNÍ VLASŮ](#)
[POMŮCKY NA VLASY](#)
[KADEŘNICKÝ NÁBYTEK](#)
[ČISTIČÍ PROSTŘEDKY A DEZINFEKCE](#)
[KOSMETIKA PRO DĚTI](#)
[KOSMETIKA](#)
[KOSMETIKA S PŘÍRODNÍMI LÁTKAMI](#)
[ELEKTRO - FÉNY,](#)

[Úvodní stránka](#) > [Produkty](#) > [Pomůcky na vlasy](#) > Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm

Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm

Žehlička na vlasy - úzká pracovní plocha - keramické plochy, kterými do vlasů proniká malé množství kondicionéru s vitamínem E pro jemné, les a zdravé vlasy - plovoucí plochy pro dokonalý tlak na vlasy - digitální kontrola teploty s LCD displejem pro všechny typy vlasů - rychlé, 30sec. zahřátí pro okamžitý styling - zámek teploty - automatické vypnutí po 1 hodině - zesilovač zahřátí k rychlému nastavení nejvyšší teploty - žáruvzdorný obal - uzamykatelné žehlicí plochy

Výrobce:	Remington
Dostupnost:	Skladem
Běžná cena:	4 699,00 Kč
Ušetříte:	600,00 Kč / 35 %
Vaše cena:	1 099,00 Kč

ks

Informace o erroru

Info

- Timestamp: 2017-05-05 19:54:26
- Info:
- ProductUrlId: 11

Vyparsovaný produkt

- Timestamp: 2017-05-05 19:54:25
- Description:
- Ean:
- Name: Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm
- PN: Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm
- Cena: 1099,-
- Cena s dph: -,-
- Skladem info: Skladem
- URL: https://www.mameradivlasy.cz/produkty/pomucky-na-vlasy/remington-shine-therapy-s-8500-zehlicka-na-vlasy-25-mm

Informace o produktu

- Description:
- Ean: 4008496759323
- Name: REMINGTON S 8500
- PN: S8500

Nalezené error

DIFFERENT_NAME

Parsed name value[Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm] doesn't match known name value[REMINGTON S 8500]

DIFFERENT_PART_NUMBER

Parsed part number value[Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm] doesn't match known part number value[S8500]

☐ Reject
 ☒ Ok
 ☐ Create new parse template

Obrázek 6.2: Ukázka webového rozhraní pro vyřešení chyby analyzátoru po provedených vylepšení.

6.6 Získání adres obchodů a příslušných detailů produktů

Vzhledem k malé možnosti využitelnosti implementované části v modulu Finder především z důvodu dlouhých metod, které zajišťují základní stavební kámen tohoto modulu, jsem se rozhodl modul Finder rozdělit 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 přidáním 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 detailů produktu, tím je dosaženo na základě šablony pro daný e-shop, která obsahuje tyto atributy:

- formát url vyhledávající produkt na obchodu
- oddělovač slov v url adrese
- selektory pro výběr url adres vedoucí na detaily produktu

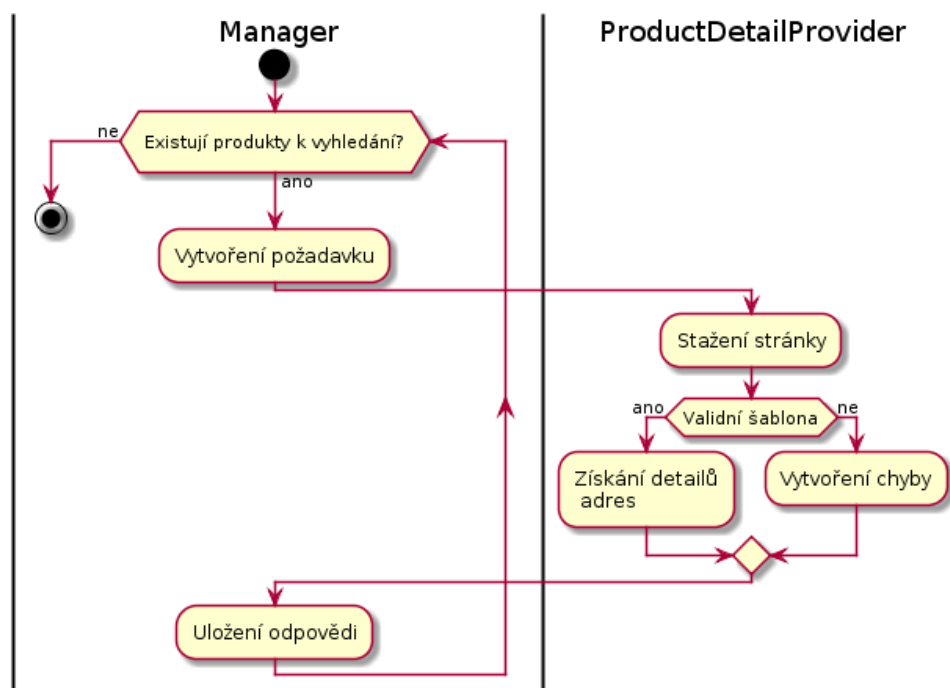
Pro vytvoření požadavku je důležitá existence částečné šablony, která obsahuje informace jak na obchodě vyhledávat. Po pokusu nalezení adres se vytvoří chyba pro administrátora, aby specifikoval, jak na stránce vyhledávat samotné detaily adres. Webové rozhraní bylo pro tento proces vytvořeno v rámci týmového projektu.

6.7 Párování produktu

Byl navržen proces, který se nejprve pokusí produkt spárovat automaticky, pokud nalezne přímou shodu názvu, EANu nebo modelového čísla. Pro shodu probíhá hledání podřetězce nalezeného identifikátoru v tom již uloženém. Pokud se hledání nepodaří je vyzkoušeno hledání podřetězce v opačném směru.

Pokud jsou oba způsoby (pro všechny identifikátory) neúspěšné, jsou provedeny heuristiky hledající pravděpodobné shody. K detekci shody jsou použity algoritmy počítající společná slova a nejdelší společný podřetězec. Z množiny těchto možností je vytvořena chyba, kterou musí zpracovat administrátor.

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



Obrázek 6.3: Aktivita diagram nalezení adres detailů produktů

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 při nalezení adresy detailu, která ač je obchodem vrácena jako výsledek při hledání, neexistuje. I pro tuto možnost byla vytvořena chyba šablony ve formátu jiného typu.

Nejprve jsem se snažil v případě detekce jiné struktury porovnávat se stránkou, která byla obchodem vrácena při hledání náhodného řetězce dlouhé délky. Myšlenka byla, že pokud produkt opravdu na obchodu není bude stránka navracená při hledání s náhodným řetězcem podobná struktura.

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 zavřeno.

Nakonec jsem zvolil možnost administrátora při řešení chyby zadat řetězec značící neexistenci produktu (popř. neexistující stránky detailu produktu), viz. obrázek 6.4. Tento řetězec je pro každý obchod specifický a před vytvořením

6. REALIZACE VYLEPŠENÍ

Oprava šablony

Hledaný text


Hledat

NÁKUPNÍ KOŠÍK

Vše o nákupu | Obchodní podmínky | Můj účet | Kontakt

Úvodní stránka > Produkty > Pomůcky na vlasy > Remington SHINE THERAPY S 8500 Žehlička na vlasy 25 mm

Remington SHINE THERAPY S 8500 Žehlička na vlasy 25



Žehlička na vlasy - úzká pracovní plocha - keramické plochy, kterými do vlasů proniká malé množství kondicionéru s vitamínem E pro jemnější, lesklé a zdravé vlasy - plovoucí plochy pro dokonalý tlak na vlasy - digitální kontrola teploty s LCD displejem pro všechny typy vlasů - rychlé, 30sec. zahřátí pro okamžitý styling - zámek teploty - automatické vypnutí po 1 hodině - zesilovač zahřívání k rychlému nastavení nejvyšší teploty - žáruvzdorný obal - uzamykací žehlicí plochy

Výrobce:	Remington
Dostupnost:	Skladem
Běžná cena:	1 999,00 Kč
Ušetříte:	600,00 Kč / 35 %
Vaše cena:	1 099,00 Kč

1 ks

Vložit do košíku

★★★★★

0

Název produktu

Aktivovat výběr

Cena produktu (Bez DPH)

Aktivovat výběr

Cena produktu (S DPH)

Aktivovat výběr

EAN

Aktivovat výběr

Model

Aktivovat výběr

Skladem/Není skladem/...

Aktivovat výběr

☐ Is alternative template

Aktivovat výběr

Vybraná hodnota

CSS selector

Vybrat

Info

Used url: <https://www.mameradivlasy.cz/produkty/pomucky-na-vlasy/remington-shine-therapy-s-8500-zehlicka-na-vlasy-25-mm>

Vyparovaný produkt

- Timestamp: 2017-05-05 21:12:11
- Description:
- Ean:
- Name:
- PN:
- Cena: -
- Cena s dph: -
- Skladem info:
- URL: <https://www.mameradivlasy.cz/produkty/pomucky-na-vlasy/remington-shine-therapy-s-8500-zehlicka-na-vlasy-25-mm>

Error string:

Reject

Uložit

Obrázek 6.4: Administrátorské rozhraní pro opravu detailu šablony. Původnímu řešení neobsahovalo informaci v použité url adrese, vyparované hodnoty a formulář pro chybný řetězec.

příslušné chyby šablony je nejdříve zkontrolováno, zda se pouze produkt pouze na obchodě nenachází (popř. stránka neexistuje) a to vyhledání řetězce na stránce. Pokud na stránce řetězec je, pak se nejedná o chybu šablony.

6.9 Více šablon detailů produktu

Původní návrh počítal s možností, kdy stránka detailu produktu je ve stejné struktuře napříč celým obchodem. Tento předpoklad se ukázal jako chybný, například v případě slevy je element obsahující cenu odlišný.

Jiná struktura pak způsobila chybu šablony. Abych problém odstranil implementoval jsem podporu alternativních šablon, které jsou použity v případě, že hlavní šablona selže. Uložení této šablony jsem zapracoval do rozhraní administrátora, kdy je třeba vyřešit chybu šablony detailu. Administrátor má možnost původní hlavní šablonu opravit nebo uložit jako alternativní.

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 tehdy, když neexistovala šablona pro detail nebo vyhledávání na obchodě. V případě více požadavků týkajícího se stejného obchodu, vytvoření způsobilo pro každý požadavek chybu.

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

Tento stav není možné predikovat. Zvolil jsem možnost, kdy je pouze upraveno webové rozhraní a následné uložení opravené chyby. Úprava webového rozhraní spočívala v zobrazení chyb týkajícího se stejného obchodu v seznamu pouze jednou. Změna uložení pak způsobí vyřešení i ostatních chyb na obchodu, čímž odpadá nutnost administrátora všechny vyřešit.

6.11 Skladem

V rámci získávání dat u detailu produktu jsem přidal možnost uložit i zda je produkt skladem.

Pro úpravu jsem upravil nejdříve rozhraní pro vytváření šablony a strukturu databáze, aby uchovávala atribut u šablony a výsledku 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 bylo například nastavení připojení do databáze. Původní stav obsahoval konfiguraci připojení v samostatných *xml* souborech, což znamenalo, že každá nová databázová třída musela být přidána do všech těchto souborů.

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

Zhodnocení provedených vylepšení

Zde bych se rád věnoval konečné funkcionalitě systému. V návaznosti na výsledný stav popíšu nedostatky objevené v rámci testování, které jsem uskutečnil. Celkové zhodnocení jsem provedl vůči výslednému návrhu a funkcionalitě. Funkcionalitu jsem testoval na vzorku produktů popsané v tabulce 7.1 a 20 internetových obchodů. Pozorování z testování se bude věnovat věnovat v následujících kapitolách.

7.1 Funkcionalita

Nejpodstatnější rozdíl při porovnání starého řešení a nového, je samotná funkcionalita, která byla značně rozšířena. Především je možné využít celý proces hledání informací. Administrátorem stačí zadat přes webové rozhraní obchody na kterých chceme hledat. Oproti původnímu řešení není nutné definovat adresy detailů přímo vložním do databáze. Podpora více šablon rozšiřuje použití na více obchodech.

7.2 Kampaně

Vytvoření kampaně zůstalo stejné jako v původním řešení. Rozdíl je nejvíce markantní v rozhraní pro administrátora, kdy je možné vyřešit všechny možné chyby, což zajišťuje validní běh kampaně. Výsledek je pak očekávaný výstup, data ze sledovaných produktů. Uživatel má pak možnost si výstup zobrazit ve webovém rozhraní nebo stáhnout export jako v původním řešení.

Jako nevýhoda a hlavní nedostatek se však ukázal velký důraz na administrátora při řešení prvotních situací, především párování produktů.

Při možném reálném použití systému je pak potřeba zajistit, aby v rámci kampaně byla možnost výběru jaké obchody jsou požadovány sledovat, jelikož

7. ZHODNOCENÍ PROVEDENÝCH VYLEPŠENÍ

Tabulka 7.1: Tabulka testovaných produktů

značka	model	název
Scholl	F215311016 (43)	AIR BAG - přírodní zdravotní pantofle
Scholl	F200781065 (43)	CLOG SUPERCOMFORT bílá
Scholl	4002448095262	Velvet Smooth wet & dry - elektrický pilník na chodidla do vody
Beurer	BEU-FB30	FB 30 nožní masáž
Sanitas	SAN-SEM43	SEM 43 svalový a nervový elektrostimulátor
Beurer	BEU-464.15	GL 44 / GL 50 / GL 50 EVO testovací proužky 464.15 (2x25ks)
Beurer	BEU-IPL10000+	IPL 10000+ Depilace SalonPro 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

aktuální stav hledá na všech obchodech v systému.

7.3 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.

Párování je nutné na obchodech, které poskytují pouze název, provést ve většině množstvím případů manuálně. Systém se manuální párování snaží ulehčit rozhraním, nicméně i při malém množství produktů je počet těchto chyb velmi rozsáhlý.

Proces párování by bylo možné vylepšit, pokud by systém uchovával více identifikátorů. Více uložených jmen by pak zátěž na administrátora mohla s časem klesat, protože použitých jmen napříč obchody by mělo být omezené množství. Podpora pro tuto možnost však zcela nebyla zcela implementována, ačkoliv v některých částech byla interní část na tuto možnost již připravena.

7.4 Webové rozhraní


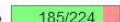

Nalezl jsem nedostatky ve webovém rozhraní, kdy jeho funkčnost sloužící k opravám šablony detailu, nebylo na některých obchodech funkční. Stačilo zadat cestu k elementům manuálně.


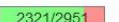

Problém obsahovala i část pro šablonu sloužící k nalezení adres detailů. Pokud bylo použito rozhraní, šablona nebyla ve většině případů funkční, je proto nutné zadávat cestu k elementu manuálně.

7.5 Návrh a testy

Zásadní refaktoring a změna návrhu komunikace tříd velmi pozměnila interní část. Interní část vyhovuje nárokům na formu v původních částech kódu, ale i v případě nové funkcionality. Je složena z přehlednějšího, udržitelnějšího a lépe rozšiřitelného kódu.

Lepší rozšiřitelnost byla výrazná především při přidávání vylepšení, kdy tyto změny byly často triviální záležitost a po jejich provedení bylo zachována původní funkcionality. Toto dávám za důsledek především lepšího pokrytí testy a rozdělení tříd do menších celků. Oproti původnímu řešení narostl počet jednotkových testů ze 170 na 492. Pokrytí 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 funkcionality servisních tříd je, až na výjimky testy, pokrytá celá. Zbylé neotestované třídy jsou především inicializační třídy front a jejich komunikační rozhraní, které je závislé na jiné běžící službě.

I po provedení těchto změn zůstala nemožnost vytvoření více instancí jednotlivých modulů, tedy ProductDetailProvider a DataProvider. Tato změna je však požadována, až při nárokových na zpracování velkého množství dat. Pro změnu stačí přidat instanční rozhraní jednotlivých modulů, jelikož momentálně jsou spouštěny přímo modulem Manager. Oddělení neovlivní samotnou komunikaci jednotlivých částí, nicméně vzroste zátěž na nastavení infrastruktury nasazení aplikace na server.

7.6 Rozhraní administrátora

Systém se stal z hlediska pro administrátora uživatelsky přívětivější. Při řešení chyb je rozhraní přehlednější a informace o chybách obsáhlejší, tím se docílí lepší rozhodování. Dále bylo podstatně zrychleno celkové zpracování. Odpadla nutnost řešit každý požadavek, ať už se jedná o víc chyb šablon u stejného obchodu či chyb analyzátoru.

Nevýhodou jsou nedostatky týkající se vytváření šablon, které byly zmíněny v kapitole 7.4.

7.7 Nemožnost vyhledávání na některých obchodech

V rámci testování jsem objevil nemožnost vyhledávat na některých obchodech. Jednalo se o dva menší obchody: *k24.cz* a *elektrocr.cz*. Příčina byla odlišná implementace hledání, kdy nestačilo pouze stáhnout obsah adresy, která je v příslušném formátu.

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 udržitelnost 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 2016.
- [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.: *Refactoring: zlepšení existujícího kódu*. Moderní programování, Grada, 2003, ISBN 8024702991.

Seznam pojmů

A.1 Web API

Web Application 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 verzích 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 ověřujících funkcionality 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 udržitelnost projektu. V případě neexistujících testů nelze ověřit původní funkcionality 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 označujících 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

```
1 {  
2   "key": [  
3     1,  
4     2,  
5     3  
6   ],  
7   "boolean": true,  
8   "null": null,  
9   "number": 123,  
10  "object": {  
11    "a": "b",  
12    "c": "d",  
13    "e": "f"  
14  },  
15  "string": "Hello World"  
16 }
```

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

Seznam použitých zkratk

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

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.....	zdrojové kódy interní částí původního řešení
└ frontend.....	zdrojové kódy webového rozhraní
└ frontend.....	zdrojové kódy webového rozhraní původního řešení
text.....	zdrojová forma práce ve formátu \LaTeX
└ thesis.pdf.....	text práce ve formátu PDF