

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

29. dubna 2017



---

## Poděkování

Chtěl bych poděkovat za trpělivost vedoucímu, Ing. Jiřímu Hunkovi.



---

## 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 29. dubna 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

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

**Klíčová slova** Nahradte seznamem klíčových slov v češtině oddělených čárkou.

---

## Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

**Keywords** Nahradte seznamem klíčových slov v angličtině oddělených čárkou.



---

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Popis problematiky získávání informací z webů</b>	<b>3</b>
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
<b>2 Analýza týmového projektu</b>	<b>13</b>
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
<b>3 Vývoj a implementace týmového projektu</b>	<b>17</b>
3.1 Pojmy . . . . .	17
3.2 Vývoj . . . . .	19
3.3 Implementace . . . . .	19
3.4 Má role . . . . .	21
<b>4 Zhodnocení týmového projektu</b>	<b>23</b>
4.1 Pojmy . . . . .	23
4.2 Stav . . . . .	23
4.3 Nedostatky . . . . .	24
<b>5 Návrhy na vylepšení</b>	<b>29</b>
5.1 Pojmy . . . . .	29

5.2	Refaktorování stávajícího řešení . . . . .	30
5.3	Plánování práce . . . . .	30
5.4	Oprava komunikace Manager - ProductProvider . . . . .	31
5.5	Spojení chyb analyzátoru a vylepšení rozhraní . . . . .	31
5.6	Monitorování . . . . .	32
5.7	Získání adres obchodů a příslušných detailů produktů . . . . .	32
5.8	Párování produktu . . . . .	32
5.9	Pokročilé párování produktu . . . . .	33
5.10	Uchování hodnot z nespárovaných adres . . . . .	33
<b>6</b>	<b>Realizace vylepšení</b>	<b>35</b>
6.1	Refaktorování stávajícího řešení . . . . .	35
6.2	Plánování práce . . . . .	40
6.3	Oprava komunikace Manager - ProductProvider . . . . .	41
6.4	Spojení chyb analyzátoru a vylepšení rozhraní . . . . .	44
6.5	Monitorování . . . . .	44
6.6	Získání adres obchodů a příslušných detailů produktů . . . . .	45
6.7	Párování produktu . . . . .	45
6.8	Detekce neexistující stránky a nenalezeného produktu . . . . .	46
6.9	Více šablon detailů produktu . . . . .	46
6.10	Více stejných chyb . . . . .	46
6.11	Skladem . . . . .	47
6.12	Ostatní . . . . .	47
<b>7</b>	<b>Zhodnocení provedených vylepšení</b>	<b>49</b>
7.1	Funkcionalita . . . . .	49
7.2	Návrh a testy . . . . .	50
7.3	Rozhraní administrátora . . . . .	50
<b>8</b>	<b>Závěr</b>	<b>51</b>
	<b>Literatura</b>	<b>53</b>
<b>A</b>	<b>Seznam použitých zkratk</b>	<b>57</b>
<b>B</b>	<b>Obsah přiloženého CD</b>	<b>59</b>

---

## Seznam obrázků

1.1	Price checking . . . . .	8
2.1	DataProvider diagram . . . . .	16
3.1	Větve v Git repozitáři . . . . .	18
3.2	MVC . . . . .	20
4.1	Pokrytí testy vytvořeného řešení . . . . .	27
6.1	Webové rozhraní pro vyřešení chyby analyzátoru po provedení vylepšení . . . . .	48
7.1	Pokrytí testy po provedených vylepšení . . . . .	50



---

# Úvod

V předmětech BI-SP1 a BI-SP2 v prostředí FIT ČVUT byl realizován týmový projekt umožňující získávání informací z webů s primárním zaměřením na potřeby obchodů. Projekt řešil problém automatizace získávání dat z webů, jelikož stávající služby neposkytují veřejné rozhraní nebo mají velkou chybovost dat.

Práce pojednává o požadavcích internetových obchodů, které jsou především tvořeny nutností držet krok s trhem a sledovat vývoj cen prodáváných produktů u konkurence.

Cílem této práce je popsat požadavky internetových obchodů, stávající stav získávání informací z webů a možná řešení problematiky. Dále na základě těchto poznatků zhodnotit vytvořené řešení a včetně korektnosti zvolených postupů navrhnout vylepšení. Ty implementovat, řádně vylepšení otestovat a zhodnotit výsledný stav projektu.





# Popis problematiky získávání informací z webů

V této kapitole se budu nejprve zabývat samotnou problematikou získávání informací z webů s důrazem na internetové obchody. Jelikož je tato problematika již řešena existujícími službami, nejprve 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, jelikož jinak z dat nejsme schopni vyčíst potřebné informace. Vzhledem k specifitě dat, které 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 je kombinace automatizace a prvku lidské inteligence. To je obvykle dosaženo roboty, kteří data stahují a lidské práce určující jaké informace nás ve stažených datech zajímají.

Získávání informací ze stažených stránek lze poté zjednodušit na problematiku určení elementů v HTML, které jsou pro nás zajímavé. Lokaci elementu v HTML se kterým je potřeba pracovat, lze poté jednoznačně určit například pomocí těchto dvou možností:

1. XPath
2. CSS Selector

Možnost jednoznačného určení elementů na stránce pak umožňuje vybrat pouze části, které jsou v kontextu hledání zajímavé. Přesná struktura dat, která lze z vybraných dat vytvořit pak může sloužit k dalšímu zpracování.

### 1.3 XML Path Language

XML Path Language[1] nazývaný zkráceně XPath je jazyk, který slouží k výběru elementu v dokumentu ve formátu XML[2].

XML chápeme jako jazyk popisující strukturu dat, které jsou strojově i lidsky čitelné. HTML lze chápat jako strukturu podobnou XML, ačkoliv se přímo o XML dokument nejedná [3]. HTML popisuje obsah dat pro prezentaci ve webovém prohlížeči pomocí předem definované struktury, které prohlížeče rozumí. Díky této vlastnosti lze 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 definované v HTML. K určení prvků se kterými pracuje používá selektory, které označují tento prvek v HTML. Buď pomocí samotného prvku, přiřazené třídy nebo nastaveného identifikátoru. Jako selektor může být použit jak samotný název prvku, tak vlastní definované třídy.[4]

Pomocí řetězení těchto selektorů je poté možné jednoznačně získat element v HTML dokumentu.

## 1.5 Současný stav řešení potřeb internetových obchodů

I v kontextu malého trhu jako Česká republika se lze bavit o velké konkurenci na poli maloobchodů prodávající své zboží na internetu. Internetové obchody potřebují monitorovat konkurenci a trh. Vzhledem k jejich zaměření je tedy nejvíce zajímaví obchody prodávající stejné zboží. Potřebné informace o prodáváných produktech konkurencí se skládají z následujících atributů:

1. Název
2. Model
3. EAN
4. Cena
5. Inzerovaný název
6. Dostupnost

S těmito daty je poté možné dále pracovat, například při analýze konkurence schopnosti na trhu nebo pokud potřebuje distributor sledovat za jaké ceny jsou prodávány jeho produkty. [5]

### 1.5.1 Srovnávače cen

Data lze získat pomocí srovnávačů cen jako *zbozi.cz*[6] nebo *heureka.cz*[7]. Problém u těchto služeb spočívá v určení pro koncové zákazníky, kterým umožňuje nalezení nejlepší ceny na trhu pro hledaný produkt. S tím souvisí to, že největší srovnávače cen neposkytují veřejně svá data nebo rozhraní přes která by je bylo možné 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 ní
- 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 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 údajně výstup velmi podobný, nebo, jak bylo výše řečeno, data neposkytují. Díky tomu se ukázaly srovnávače 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 srovnávacích. Uživatel si zadá okruh či seznam produktů, buďto formou manuální či vstupem ze souboru, případně 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, zda 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
Download date 01.03.2012						
Czech Republic - Electro						
1320	-1	(-0 %)	0	(0 %)	1	35
2959	-10	(-0 %)	0	(0 %)	2	48
474	0	(0 %)	0	(0 %)	1	22
670	0	(0 %)	0	(0 %)	2	27
1414	-3	(-0 %)	0	(0 %)	0	37
3244	1	(0 %)	61	(2 %)	0	22
3961	25	(1 %)	0	(0 %)	0	24
6746	-9	(-0 %)	0	(0 %)	0	51
24025	45	(0 %)	3	(0 %)	2	448
680	16	(2 %)	0	(0 %)	0	19
7377	95	(1 %)	0	(0 %)	6	145
4140	21	(1 %)	60	(1 %)	8	29
3368	-2	(-0 %)	3	(0 %)	0	21
11573	-2909	(-20 %)	0	(0 %)	15	89
260	0	(0 %)	0	(0 %)	0	1
6440	3	(0 %)	32	(0 %)	1	47
13878	3963	(40 %)	0	(0 %)	2	59
13759	-1	(-0 %)	124	(1 %)	119	175
5960	-1473	(-20 %)	0	(0 %)	67	216
10421	-186	(-2 %)	0	(0 %)	0	73
2476	37	(2 %)	0	(0 %)	0	38
37549	-111	(-0 %)	3	(0 %)	-1	143
13906	37	(0 %)	10	(0 %)	1	130
7321	-9	(-0 %)	0	(0 %)	0	53
18016	7	(0 %)	0	(0 %)	0	83
9193	-3527	(-28 %)	0	(0 %)	-2	10
6666	-295	(-4 %)	2	(0 %)	0	35
5072	269	(6 %)	0	(0 %)	3	47
4356	-19	(-0 %)	17	(0 %)	4	136

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 školní výuky na ČVUT FIT v akademickém roce 2015/16. Nejprve popíši cíl, který měl projekt za úkol řešit a jaké byly použité postupy při vývoji. Budu se také věnovat základnímu návrhu aplikace, podle kterého byla následně implementována a hlavním použitým technologiím. V poslední části zmíním mou roli v týmovém projektu.

### 2.1 Cíl týmového projektu

V předmětech BI-SP1 a BI-SP2 byl realizován týmový projekt, v souladu s osnovami těchto předmětů byl nejdříve v BI-SP1 vytvořen návrh systému a v BI-SP2 implementován.

Cíl který projekt řešil, byla maximální možná míra automatizace získávání informací o produktech prodáváných konkurencí. Důraz je především kladem na optimalizaci počtu nutných lidských úkonů administrátora u kterého se předpokládá minimální technické vzdělání. Jediná nutná problematika, co tak musí administrátor ovládat je parsování HTML stránek.

### 2.2 Požadovaná funkcionalita

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 ochody, které prodávají sledované produkty. Z nich poté v definovaných intervalech získává data. Z dat je poté vytvořen výstup pro uživatele obsahující získané informace, které se skládají především ze získaných cen. 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 interní částí a interakcí administrátora, který zajišťuje řešení problémů, které systém nedokáže sám 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 druhá 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 pak spolu sdílejí data pomocí relační databáze[15].

### 2.4 Webové rozhraní

Webové rozhraní lze rozdělit na dvě části. Uživatelskou, obsahující množinu stránek určených pouze pro konečné uživatele služby a část pro administrátory, sloužící k monitorování kampaní a řešení chyb. Webové rozhraní používá databázi, která pak obsahuje všechny uložené a získané data.

Uživatelská část umožňuje vytvořit kampaň. Kampaň je proces trvající určitý časový úsek, který sleduje vložené produkty na konkurenčních obchodech. V rámci těchto běžících kampaní má poté uživatel možnost uživatel vidět vizualizaci získaných dat, případně je umožněn export dat do formátu *csv* či *xlsx*. Získaná data obsahují, kde se sledované produkty nacházejí a za jakou cenu se prodávají.

Druhá část je určena pouze pro administrátory. Slouží k monitorování kampaní uživatelů a řešení problémů, které systém není schopný automaticky vyřešit. Což je 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 poté možné spustit jako služby ve více instancích, kromě modulu Manager. Vzhledem k možnostem front, je pak možné práci distribuovat na více serverů, aniž by byla ohrožena bezpečnost databáze, jelikož k té je umožněný přístup pouze lokální. Moduly jsou detailněji popsány v následujících podsekcích.

### 2.5.1 Manager

Manager je hlavní modul, který má jako jediná 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.

### 2.5.2 Finder

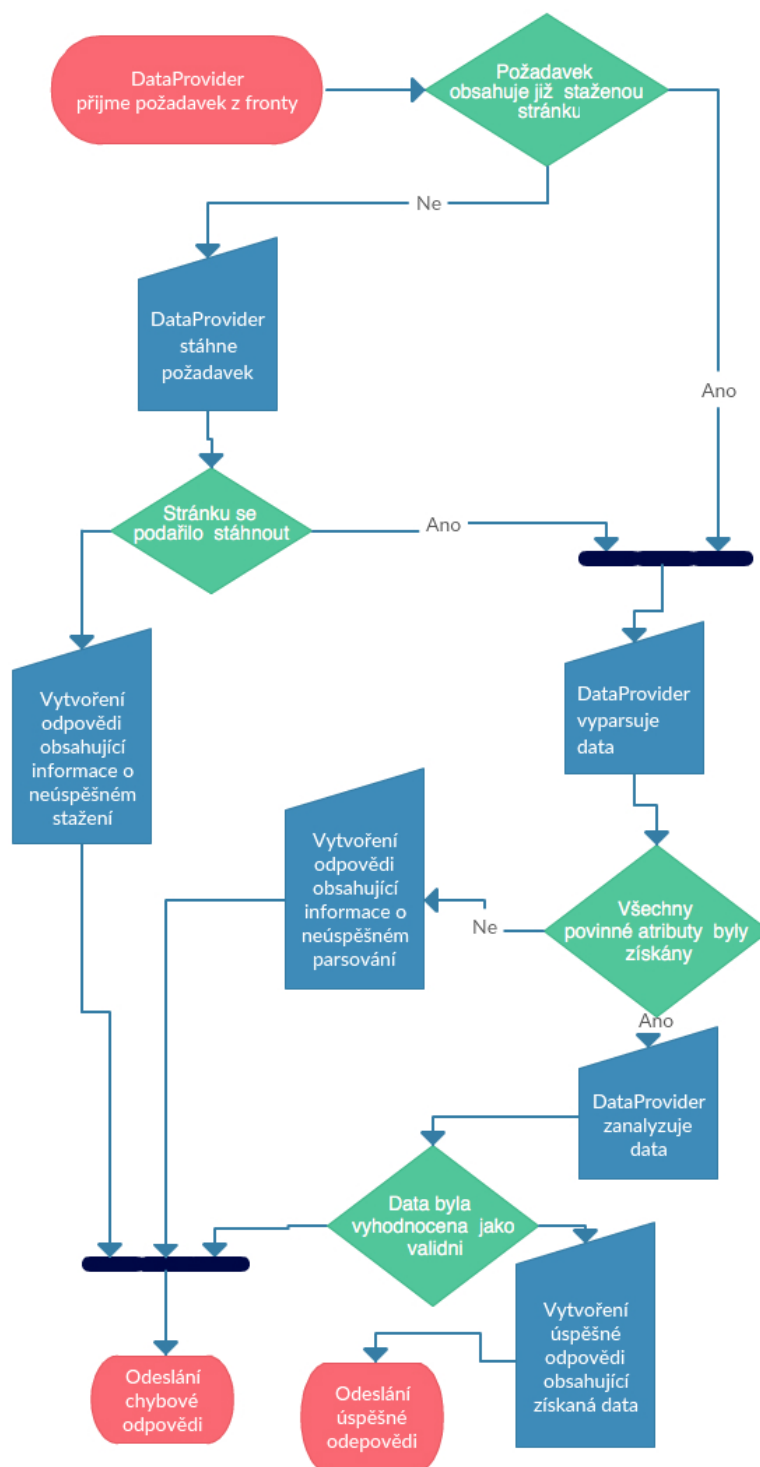
Finder je modul, který má za úkol získávat URL adresy internetových obchodů, které prodávají požadované produkty. Na nalezeném obchodě poté vyhledávat adresy vedoucí na detail vyhledávaných produktů.

Předpokládá se nemožnost přímo nalézt adresu vedoucí na detail produktu a proto je nutné ho pomocí interního vyhledávání obchodu takový detail nalézt pokud existuje. Detail produktu pak obsahuje podrobné informace o prodávaném produktu.

### 2.5.3 DataProvider

DataProvider je modul, který zpracovává adresy vedoucí na detaily produktu. Zde existují čtyři hlavní větve možností zpracování požadavků. Po stažení stránky se pokusí získat požadované hodnoty. V případě neúspěchu odešle příslušnou chybu, v opačném případě zanalyzuje korektnost získaných cen vůči historickým datům pokud existují. Výsledek je poté odeslán k zpracování a uložení do databáze Managerem.

## 2. ANALÝZA TÝMOVÉHO PROJEKTU



Obrázek 2.1: Diagram zobrazující aktivity v DataProvider modulu

## Vývoj a implementace týmového projektu

V této kapitole se věnuji průběhu vývoje týmového projektu a vytvořenému řešení. Poslední část rozebírá mou roli v tomto projektu, jelikož jsem věděl, že budu téma dále rozvíjet v rámci bakalářské práce. Z tohoto důvodu jsem se projektu věnoval nad rámec předmětu.

### 3.1 Pojmy

#### 3.1.1 Verzovací systém Git

Git [16] je verzovací systém umožňující sdílení jednotlivých verzí verzovaného projektu. Umožňuje jednoduchý přehled nad rozpracovanými částmi každého vývojáře. Úložiště systému se nazývá repositář, který obsahuje veškerý kód. Repositář pak existuje v lokálních verzích a zároveň serverové. Sdílený repositář pak zajišťuje distributivitu mezi všemi potřebnými členy. Pro lepší správu pak existují nadstavby nad serverovou částí repositáře, které umožňují jednoduchou správu nad kódy a spouštění úkolů v závislosti na definované akci. Zde například spuštění sestavení nebo notifikace při nové změně.

Základní jednotkou tvoří verze, které jsou postupně vytvářeny vývojáři po vytvoření každé malé funkcionality. Verze jsou poté uchovávány v jednotlivých větvích programu. Vedlejší větve slouží pro samotnou postupnou práci. Hlavní větve poté tento kód spojují. Git také zajišťuje základní nástroje pro slučování kódu v případě spojování nebo slučování vedlejších větví do větví hlavních.



Obrázek 3.1: Zobrazení větví v repozitáři, kde *master* je hlavní, *develop* vývojová a *topic* představuje větev vedlejší

#### 3.1.2 Jednotkové a integrační testy

Jednotkovými testy se rozumí sada kladných a záporných testů ověřující funkcionalitu pouze jedné třídy. Jednotkové testy jsou nezávislé na ostatních třídách a testech. [17] Integrační testy pak pokrývají komunikaci více tříd nebo komunikaci s operačním systémem, hardwarem či rozhraním různých systémů. [17]

Důvod psaní testů je lehčí nalezení chyby a lepší udržitelnost projektu. V případě neexistujících testů nelze poté ověřit původní funkcionalitu při modifikaci aplikace, což může způsobit nutnost chyby nalézt a opravit.[17]

#### 3.1.3 Statická analýza kódu

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

#### 3.1.4 Průběžná integrace

Průběžnou integrací se rozumí sada nástrojů sloužící k urychlení softwarového vývoje. Princip je průběžné sestavení a spouštění testů aplikace na základě změny ve sdíleném repozitáři. Lze tak rychle odhalit případné chyby před zařazením příslušné verze do produkce.[18]

#### 3.1.5 Sdílení dat pomocí front

Princip sdílení dat pomocí front je na odesílání zpráv, které reprezentují objekty. Fungují na principu, kdy jedna strana objekty odesílá a druhá přijímá.



Příklad takové implementace pak může být například RabbitMQ. [19]

## 3.2 Vývoj

Vývoj byl rozdělen do 5 iterací, z nichž každá obsahovala 10 sprintů. Před začátkem vývoje byla rozdělena práce do těchto částí, s tím, že na konci každé iterace probíhala prezentace vyučujícím. Každý sprint se skládal z jednotlivých úkolů, které byly přiřazeny členům týmu. Stav úkolů byl uchováván na systému Redmine[20], ten umožňuje přehledné řízení projektu a možnost sledování objevených chyb.

Jako verzovací systém byl zvolen systém Git, se vzdáleným repozitářem uložený na službě [21]. Gitlab poskytuje webové rozhraní pro snadnou správu a spouštění služeb na základě změn. Projekt byl v rámci repozitáře rozdělen na 4 části (větve):

- Master - hlavní větev uchováající verze určená 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 nalezenou chybu

Jelikož přístup k přidání verze do Master a Develop měl pouze vedoucí projektu, musel být pro každou Feature a Fix větev vytvořen požadavek o zařazení. Po kontrole vedoucím byl požadavek zařazen nebo vrácen k opravě.

Na konci každé iterace byla poslední verze vždy označena ve verzovacím systému a poté prezentována vedoucím. Označení bylo zvoleno na základě pořadí iterace. 1. iterace je označena verzí „0.1“.

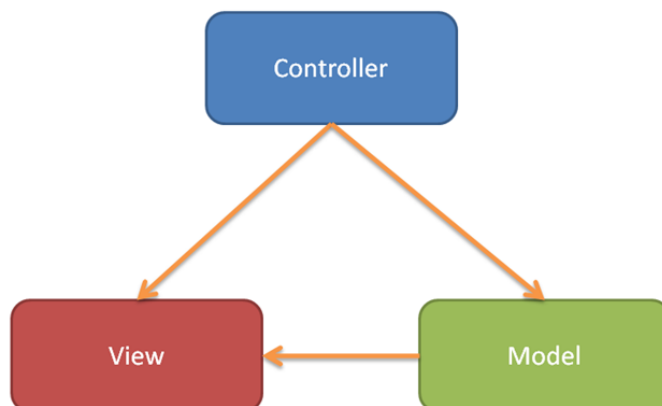
Pro vývoj se využil princip průběžné integrace. Každá verze byla zkompileována, otestována a zanalyzována na vzdáleném serveru. Tyto činnosti zajišťovaly systém Jenkins[22] a Gitlab. Jenkins aplikaci zkompileoval, spustil testy a statickou analýzu kódu zajištěnou systémem SonarQube[23]. Výsledky poté publikoval ve svém webovém rozhraní a zároveň v rozhraní Gitlab.

## 3.3 Implementace

### 3.3.1 Webové rozhraní

Webové rozhraní je implementováno v jazyce PHP verze 7. Základem aplikace je aplikační rámec Nette[24]. Nette obsahuje nástroje pro automatickou správu závislostí, komunikaci s databází, vytváření bezpečných formulářů, zabezpečení aplikace, šablonovací systém a rozhraní pro tvorbu jednotkových testů.

Dále automaticky umožňuje lehké dodržení MVC architektury, která odděluje prezenční a logickou vrstvu. Zkratka MVC značí Model-View-Controller. V případě projektu je view šablona definující vzhled webové stránky, controller třídy obsluhují šablony. Modelovou vrstvu poté zajišťují servisní třídy, vykonávající logické části aplikace jako například přístup do databáze nebo zpracování formuláře.



Obrázek 3.2: Vizualizace návrhu MVC (Model-View-Controller)

Snadnou správu závislostí umožňuje balíčkovací systém Composer[25]. Na základě souboru definující potřebné knihovny a jejich verze, jsou staženy z centrálního repozitáře. To zajišťuje jednotné verze knihovny a eliminace nutnosti knihovny manuálně stahovat či přímo přidávat do verzovacího systému.

#### 3.3.2 Interní část

Interní část je implementována v jazyce Java verze 8. Kompilaci, spouštění testů a správu závislostí zajišťuje Gradle[26]. Gradle je nástroj sloužící k automatickému sestavení aplikace. Umožňuje správu závislostí, kde je standardně nastavený jako zdroj centrální maven repozitář. [27]. Na něm jsou uloženy poté všechny knihovny, která jsou v rámci toho projektu použity.

V rámci sestavení lze pustit testy, včetně přídatných doplňků. Projekt používá doplněk Cobertura[28], který na základě spuštěné testovací sady vytváří zprávu obsahující pokrytí větvi programu. Díky tomu lze jednoduše zjistit jaké větve aplikace nejsou otestované.

Aplikace je rozdělena do nezávislých modulů běžící jako služby. Jednotlivé moduly spolu komunikují pomocí posílání zpráv definovaných front. Komunikaci zajišťuje systém RabbitMQ Server 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 k sdílené podobě objektu na obou stranách, zprávu lze jednoznačně deserializovat zpět do původní Java objektu se kterým je poté možné dále pracovat.[29]

Použité knihovny:

- Google Guice - automatická správa závislostí [30]
- Hibernate - objektově relační zobrazení databázových entit a práce s nimi [31]
- Apache Commons - pomocné knihovny pro práci s řetězci a soubory [32]
- RabbitMQ - zajišťuje komunikaci s frontami [19]

### 3.4 Má role

V druhé části týmového projektu, tedy implementace jsem byl vedoucí týmu. Jelikož jsem se již znal své téma bakalářské práce, věnoval jsem se projektu nad rámec předmětu

Především na začátku projektu jsem věnoval čas vytvoření celého ekosystému, tvořen z přidružených služeb použitých při vývoji. Zde se jedná především o projení 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 informace o výsledku
- SonarQube - zobrazování interaktivního výsledku statické analýzy přímo v rozhraní Gitlab

Samotný SonarQube bylo nejprve potřeba nastavit, aby se spouštěl při sestavení aplikace a výsledek se poté zobrazil v rozhraní Gitlabu. V rámci sestavení aplikace jsem ještě nastavil spouštění nástroje Cobertura. Doplnky v Jenkins poté umožňovaly zobrazení přehledných výsledků, jak je kód pokryt testy. Přesné pokrytí testy mi poté umožňovali jednoduše kontrolovat, zda jsou správně napsané všechny testy.



## Zhodnocení týmového projektu

Pro nutnost návaznosti na kapitolu o vylepšeních je nejprve nutné uvést v jakém kontextu jsou vylepšení navrhovány. K tomu je třeba popsat výsledný stav projektu a jeho funkcionalitu.

### 4.1 Pojmy

#### 4.1.1 JSON

JSON označuje specifikaci formátu pro výměnu dat[33]. Jedná se o formát, který je čitelný jak pro lidské oko tak pro stroj[33] a jeho zpracování je implementováno pro velké množství programovacích jazyků[34]. Základní stavební jednotkou tvoří kolekce párů klíč a hodnotu spolu se seřazeným polem hodnot.

#### 4.1.2 Web API

Rozhraní, které na definovaný HTTP dotaz vrací data. Ty jsou standardně vracena ve formátech JSON nebo XML.

### 4.2 Stav

Ačkoliv stav projektu odpovídal nárokům na úspěšné odevzdání nebyla dosažena implementace všech procesů, aby byla umožněno reálné použití systému.

Odevzdávaný stav obsahoval funkční webové rozhraní, které se skládalo ze základní funkcionality pro uživatele a pro administrátory. Část pro administrátory obsahovala správu uživatelů, možnost parsování stránek, evidenci známých obchodů a produktů či sledování chyb vzniklých v interní části. Uživatelská část umožňovala správu a vytvoření kampaně či kampaní. Implementace kampaní pak splňovala návrh z analytické části, která byla zmíněna výše. Na žádost jiného týmu jsme také vytvořili REST rozhraní, poskytující získané ceny pro daný produkt.

Interní část byla schopná pracovat pouze na základě dříve uložených url adres vedoucí detaily produktů. Manager dokázal zjistit vybrat produkty, které jsou v aktivní kampani a je třeba je aktualizovat. Na základě výsledků poté vybral potřeba data odeslal požadavek pomocí pro zpracování do DataProvideru. Ten na základě obdržených dat stránku vyparsoval a zanalyzoval výsledek vůči historickým datům a informacím o produktu.

Analýza se především skládala z kontrol velkých výkyvů cen a rozdílných identifikátorů produktu. V případě, že požadavek neobsahoval šablonu pro vyparsování nebo byl výsledek označen jako chybný a byla odeslána chyba zpět do Manageru. Manager následně výsledek korektně uložil pro zobrazení ve webovém rozhraní, ať už se jednalo o chybu nebo nalezení ceny.

Byla také obsažena detekce opravených chyb, díky kterému Manager poznal, že může pokračovat v práci hledání cen na daném.

### 4.3 Nedostatky

Vytvořené řešení obsahovalo spoustu nedostatků, které je 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 je důvodem mnoha chyb.

#### 4.3.1 Vytváření požadavků pro ProductProvider

Následující nedostatky jsou úzce spjaté s tím, jak aplikace plánovala práci. Plánování práce je myšleno proces ve kterém jsou vybrány požadované adresy detailů a z nich vytvářeny požadavky pro ProductProvider k zpracování.

Jak bylo řečeno, prvotním kritériem jsou adresy detailů. Ty se mohou vyskytovat v různých stavech. Systém je vybral v následujícím pořadí, z kterého následně vybral první 10:

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

Z této množiny byly poté odebrány odeslané a ty které mají nevyřešenou chybu. Opakované spuštění v předdefinovaném intervalu poté 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, kde je potřeba označit,

že byla zpracována. Zde se však ovšem nastavil příznak o zpracování, ale pokud se požadavek nepodařilo odeslat nebyl již změněn. To způsobilo, že chyba již nebyla nikdy zpracována.

#### 4.3.2 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 tedy neobsahovalo použití již staženého dokumentu, na jehož základě byla vytvořena chyba parsování nebo analyzování. Každý požadavek tedy vyústil v opětovné stažení příslušné stránky.

#### 4.3.3 Chyby analyzování

Poslední fáze procesu v modulu ProductProvider byla navržena jako analyzování získaných dat vůči již dříve uloženým. Implementace analyzátoru spouštěla jednotlivé validace, jejíž 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 získané na konkrétní stránce dříve. Kontrola zde porovnávala průměr historických hodnot se získanými cenami. V případě, že rozdíl byl větší jak 25%, byl výsledek označen jako možná chyba.

V případě, že 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 přidavných informací.

Informace byly následně odeslány a manager na jejich základě vytvořil chybu pro administrátora k vyřešení. Administrátor mohl poté označit, zda je to opravdu chyba nebo se má do budoucna ignorovat.

V prvním případě se chyba přestala používat, v druhém byl vytvořen příznak označující, že se na daném obchodě vyskytují jiné identifikátory nebo skokové změny ceny a validační chyba byla zahozena. Problém nastával pokud se na adrese objevilo více chyb. To znamenalo, že každý další požadavek vytvořil další chybu analyzátoru a administrátor tak všechny vyřešit a při každém požadavku bylo nutné obsah adresy znova stáhnout.

#### 4.3.4 Vytváření chyb šablon

Jako další problém se ukázalo plánování práce založené na kritériu, kdy jsou k vytvoření požadavku vybrány všechny adresy, které neobsahují šablonu. Myš-

lenka byla taková, že pro vytvoření samotné šablony, je nutné nejdříve stránky stáhnout, nechat vyhodit chybu parsování a následně chybu vyřešit.

Systém však odeslal požadavek pro všechny uložené adresy na obchod. To vyústilo ve vytvoření mnoho chyb, které musel administrátor všechny postupně vyřešit.

##### 4.3.5 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 třídy Managera zajišťující vytváření příslušných požadavků neumožňovaly ověření celkové funkcionality této části. Z tohoto důvodu nebyl systém jako celek použitelný pro jakékoliv reálné použití, jelikož jediná možnost jak využít funkcionalitu interní částí, bylo vytvořit SQL insert skripty, obsahující adresy detailů produktů a ty spustit nad databází, kterou systém používal.

Další důsledek byla neexistence procesu párování produktů. Finder byl navržený tak, že po nalezení internetového obchodu pomocí interního vyhledávání nalezne detaily produktů, u kterých je velká pravděpodobnost, že patří hledanému produktu. To je však třeba ověřit. Po získání hodnot ze stránky je tak třeba nežádoucí adresy vyloučit a ostatní spárovat s produktem.

##### 4.3.6 Plánování práce

Projekt neosahoval plánování práce vůči požadovanému intervalu, kdy mají být nová data stažena. Jak bylo řečeno výše, aktuální stav pouze hledal adresy produktů, které se nachází v zaplacené kampani nebo měli vyřešenou chybu.

##### 4.3.7 Škálovatelnost

Původní návrh počítal se škálovatelností aplikace na více serverech, kde je možné vytvořit neomezený počet instancí DataProvider a Finder. Reálný stav na konci projektu však tuto možnost nevyužíval, ač to bylo možné. Interní část tak běžela jako jedna služba. Po spuštění byly inicializovány všechny moduly běžící v samostatných vláknech.

##### 4.3.8 Obecný návrh a testy

Implementace samotná, byla velmi nepřehledná. Vykytovaly se prvky značící špatný návrh aplikace. Zde bych rád zmínil například dlouhé a nepřehledné metody v „DataProviderServiceImpl:process a AbstractFinderUrlListWorkerImpl:getProductUrl“, kde ačkoliv 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, který byl již zmíněn a popsán výše.

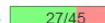
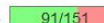



Je nutné podotknout, že spoustu špatných konstrukcí byly eliminovány již v průběhu vývoje týmového projektu. První důvod byla automatická analýza kódu detekující mnoho konstrukcí, které by se neměly v kódu vyskytovat. Automatická analýza však nebyla schopná najít všechny problémy. Druhý důvod byla moje kontrola při schvalování vytvořeného kódu pro zadaný úkol. Z důvodu časové tísně, ale nebyl vždy čas na to vrátit kód k přepsání a opravení všech chyb. To způsobilo, že se vědomě dostaly konstrukce, které nebyly považovány za ideální s myšlenkou, že budou přepracovány později, což se ne vždy povedlo.


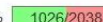

Další problém návrhu byly procesy v DataProvider modulu řízené pomocí výjimek obsahující přídavné informace i v případech, kdy byl takový výsledek očekávaný nebo dokonce chtěný. Toto použití je však v rozporu ideou použití výjimek, které mají signalizovat neočekávaný stav, kdy není možné dále pokračovat.[35] Pro uchování přídavných informací bylo nutné vytvářet výjimky vlastní, tak aby byl k údajům možný pozdější přístup.

V kritických částech dále 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 funkcionality bez možnosti rychlého ověření. Tím by mohla být jakákoliv změna velmi časově náročná a s velmi nejistým konečným výsledkem.

Systém proto v tomto stavu nebyl vhodný pro následný rozvoj.

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



## Návrhy na vylepšení

V této kapitole popíšu návrhy možná vylepšení. Tyto návrhy považuji za nejvíce důležité a proto jsou v rámci této práce i implementované. Samotné implementaci se poté věnuji v následující kapitole. Návrhy, které nebyly uskutečněny, jelikož se v době návrhu nezdály tolik důležité nebo byly zjištěny až v průběhu vylepšování, budou zmíněny v samostatné kapitole.

### 5.1 Pojmy

#### 5.1.1 Mock

V objektově orientovaném programování se Mock objekt používá pro simulování chování konkrétní třídy.[36] Při testování je tedy možné docílit testů, které nejsou závislé na ostatních třídách, kromě té která je přímo testována. Jelikož testovaná třída obvykle vyžaduje závislost na jiných třídách či rozhraní, jsou tyto části pomocí Mocku simulované. Mimo nadefinování požadovaného chování, lze také na Mock objektu sledovat jaká na něm byly provedené 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.

#### 5.1.2 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. [37] 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: [37]

- Dlouhá metoda

- Velká třída
- Dlouhý seznam parametrů
- Složité struktury podmínek

### 5.2 Refaktorování stávajícího řešení

Vzhledem ke výše popsaným problémům týkající se samotné implementace v interní části, nebylo možné na napsaném kódu stavět případné opravy nebo celková vylepšení. Z tohoto důvodu by bylo vhodné upravit téměř celou interní část, tak aby bylo možné kód lépe udržovat. Zároveň se pokusím, zachovat co nejvíce původních částí. Obzvlášť takové, které jsou ověřeny že fungují pomocí testů. Při přepsání se budu především věnovat nastavení komunikace jednotlivých tříd mezi sebou. Kvůli tomu bude potřeba změnit jejich komunikační rozhraní, nicméně vnitřní logická funkcionality by měla ve většina případů zůstat stejná.

Dále pro větší přehlednost by bylo vhodné přesunout všechny servisní třídy do samostatného balíku a sjednotit je. Jednotlivé funkcionality budou poté v samostatných balíčcích.

#### 5.2.1 Řízení aplikace

Aplikace, především v modulu `ProductProvider`, byla řízena 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.

### 5.3 Plánování práce

Samotná logika plánování práce, tedy nalezení adres detailů, které chceme použít, byla velmi nedostatečná a proto jí je třeba doplnit o požadovanou funkcionality.

Základním kamenem požadavků je vždy adresa url. Až na základně adresy jsou poté zjištěny všechny ostatní údaje, které jsou uloženy do požadavku, který je odeslán a zpracován. Požadavky je možné rozdělit podle těchto kritérií a priority, kde první má prioritu nejvyšší:

- Url bez známého produktu
- Vyřešená chyba analyzátoru
- Vyřešená chyba šablony pro parsování

- Url adresy detailů, které jsou v aktivní kampani

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
- Není požadován výstup z důvodu potřeb kampaní
- Požadavek pro adresu byl již odeslán
- Product není třeba v aktuálním dni zpracovat

Stavy je tedy potřeba implementovat ideálně takovým způsobem, kdy bude velmi jednoduché kdykoliv kontrolu přidat nebo odebrat.

## 5.4 Oprava komunikace Manager - ProductProvider

Vzhledem k problémům zmíněných výše, je třeba komunikaci těchto dvou modulů optimalizovat. Neefektivní chování je velký problém při zpracování velkého počtu dat, především při 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, která může způsobit nefunkčnost systému. Rychlá oprava by znamenala, že pro každou vytvářenou adresu je nutné se nejdříve pokusit najít vyřešenou chybu šablony nebo analyzátoru, ze které případně použít staženou stránku. Hledání by tak probíhalo ve většině případů zbytečně, jelikož chyba by neexistovala.

Korektní oprava je úzce spjata s předchozími kapitolami, především s refaktorováním a úpravou plánování práce. Vzhledem k úpravě vytváření požadavků je možné se zachovat k příslušným adresám a požadavkům z nich vytvořených odlišně. Jedná se o především správné nastavení atributů a příznaků o stavu odeslání.

V případě refaktorování, pak stačí pouze ověřit zda se šablona vyskytuje v šabloně a v kladném případě opětovně nestahovat stránku.

## 5.5 Spojení chyb analyzátoru a vylepšení rozhraní

Analyzátor provádí kontrolu získaných dat. V případě podezření o nevalidních datech je vytvořena chyba, určena k vyřešení administrátorem. Validace

kontrolují ekvivalenci identifikátorů vůči již uloženým a cenám získaných v minulém hledání na daném obchodu.

Administrátor má poté 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 tak možnost porovnat jaká data byla využita při validaci.

V případě adresy, která obsahuje více chyb je nutné řešit každou samostatně, kdy je po každém vyřešení počkat na zpracování DataProviderem. Zde je nutné požadováno chyby spojit do jedné, tak aby administrátor mohl vyřešit všechny možné chyby analyzátoru najednou.

### 5.6 Monitorování

Na virtuálním serveru probíhá sestavení aplikace, včetně všech jeho procesů. Dále 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 lepší monitorovací službu, která umožňuje unifikovat sledování probíhajících procesů na serveru a zobrazovalo stav na jedné stránce.

### 5.7 Získání adres obchodů a příslušných detailů produktů

Interní část vyžaduje ke své funkcionalitě, již uložené adresy detailů produktů, se kterými následně pracovala, resp. pracoval především modul ProductProvider. Původní návrh počítal s modulem Finder, který se však nepodařilo zapojit v rámci týmového projektu. Ten měl za úkol hledat internetové obchody na cenových srovnávacích a na nich pomocí vyhledávání nelézt adresy.

Funkce Finderu je však navržena jako duální, zajišťuje tedy 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 ale 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.

### 5.8 Párování produktu

Po nalezení detailu produktu, stažení v DataProvider modulu a následném vyparsování hodnot je třeba adresu spárovat s existujícím produktem. Příčina

nutnosti párování je, že po nalezení adresy detailu není jisté, zda opravdu patří produktu, pro který byla nalezena.

Párování musí být provedeno s velkou jistotou. Proto navrhuji vytvořit proces, který se nejprve pokusí produkt spárovat v případě přesné shody některého z identifikátorů, což představuje název, modelové číslo nebo EAN kód produktu.

Proces není možné zcela zautomatizovat, jelikož velká část internetových obchodů neposkytuje na svých stránkách validní informace.[5] Buď obchod používá název, který není oficiální od distributora nebo jsou odlišné od uložených identifikátorů jako modelové číslo nebo EAN kód. Odlišnosti těchto dvou identifikátorů může způsobit například jiná barva nebo přidaná velikost za nebo před modelové číslo. Jako řešení se jeví hledat podřetězec modelového čísla a EAN kódu, což řeší i problém pokud je ze stránky vyparsován text okolo identifikátoru. Obchod také může poskytovat pouze název, což 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 pak spoléhá na to, že vložená data při vytváření kampaně jsou validní. V případě nevalidních dat jako třeba příliš obecných a krátkých názvů by pak párování proběhlo chybně.

## 5.9 Pokročilé párování produktu

Párování produktu lze vylepšit o uchovávání více hodnot pro identifikátory produktů, která systém může použít při dalším párování na jiných obchodech. Ukládání nových identifikátorů by mělo probíhat pouze se souhlasem administrátora, tedy při úspěšném párování nebo manuálním vložením.

## 5.10 Uchování hodnot z nespárovaných adres

Vyhledáváním na obchodu je zpravidla dosažen výsledek hledání, který obsahuje větší množství adres než pouze jediná hledaná. Většina jich je tak v době hledání nevyužitelná, nicméně v budoucnu mohou být využity. Pro dlouhodobě efektivnější chod systému se proto jeví uchovávat získaná data. Zde se jeví možnost získaná data z detailů uchovávat mimo interně uložené produkty a v případě přidání nových produktů do systému v rámci vytvořené kampaně se pokusit najít shodu v těchto datech. To umožní odlehčení zátěže na stahování stránek a celkové zrychlí chod systému.

Získaná data a k nim adresy však mohou být neaktuální, kdy obchod už daný produkt neprodává a nebo adresa již nefunguje. Zde je proto nutné nastavit mechanismus maximálního stáří dat při použití nebo ověření zda jsou platná.





## Realizace vylepšení

Kapitola realizace vylepšení se věnuje implementovaným vylepšení. 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 verzí *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 na více malých, zkrácení dlouhých metod, z menšení počtu parametrů metod a nahrazení výjimek za návratové typy.

#### 6.1.1 Servisní třídy

Pro větší přehlednost byly všechny servisní třídy do nadřazeném balíčku „service“. Servisní třídou jsou takové, které nespádají do ani jedné z těchto skupin:

- Obsluha frekvenčního probouzení aplikace v daném intervalu
- Přímá komunikace s frontami
- Třídy přistupující k databázi, zkráceně DAO
- Fasády, které obalují komunikaci servisních tříd s DAO objekty
- Konfigurační soubory automatické správy závislostí
- Pomocné třídy

Třídy jsem poté pojmenoval pomocí nové konvence, kdy důležité servisní třídy obsahují prefix jakého modulu se týkají a postfix *service*. Důvod byl 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*. Následující řetězce uvádějí název včetně nadřazených balíčků, kdy název samotné třídy je v prvním případě *ParserImpl*.

### 6.1.2 Řízení aplikace

Řízení aplikace, především v DataProvideru bylo řízeno pomocí výjimek, které způsobovaly problémy popsané v návrhu na vylepšení. V návaznosti na tento návrh tak byly odstraněny a nahrazeny úpravou návratového typu, který obsahuje příznak výsledku a příslušné informace o něm.

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 proměnným uchovávanými data a přístup k nim je možný pouze pomocí *get* a *set* metod.

Třídy přístupující k datům, jsou tak odstíněny k implementačnímu detailu třídy a odstínění irelevantních detailů implementace [38]. Metody *get* jsou pak oproti standardnímu návrhu pozměněny tak, že nevrací přímo proměnnou, ale *Optional* této proměnné. *Optional* je kontejner, který může nebo nemusí obsahovat prázdnou hodnotu [39]. Před přístupem k hodnotě je proto vyžadováno se objektu zeptat, zda obsahuje hodnotu. Mechanismus očividné demonstrace, že hodnota může být prázdná, pak zamezuje nežádoucím výjimkám, především *NullPointerException* [40].

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

### 6.1.3 Spouštění validací

Porovnání změn lze demonstrovat na 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 změna je poté samotné spouštění validací. V původním řešení byla třída závislá na všech příslušných validacích, které spustila, ale skončila při první chybě, což je jedna z příčin chování, které jsem popsal v kapitole poukazující na možnosti spojení chyb analyzátoru section 5.5.

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

Validačnímu rozhraní byl také změněn návratový typ na *Optional* případné zprávy 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 jaká hodnota se má získat ze získaných dat a z dat již uložených. Poslední rozdíl byl poté pouze v chybové hlášce. Z toho důvodu jsem společnou logiku obou skupin implementoval pomocí abstraktní a generické třídy *AbstractAnalysis*. Vlastnosti jednotlivých skupin pak obsahují třídy *AbstractStringAnalysis* nebo *AbstractPriceAnalysis*. Výsledná základní validace kontrolující, zda získaná hodnota odpovídá nějaké historické hodnotě vypadá následovně:

---

```
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) {
```

---

```

32     return comProduct.getNames();
33 }
34
35 /**
36  * {@inheritDoc}
37  */
38 @Override
39 AnalysisErrorMessage generateAnalysisErrorMessage(String
40     comProductValue, String parsedValue) {
41     return new AnalysisErrorMessage()
42         .withErrorMessage(
43             String.format("Parsed name value[%s] doesn't
44                 match known name value[%s]",
45                     parsedValue, comProductValue)
46         )
47         .withErrorType(AnalysisErrorType.DIFFERENT_NAME);
48 }
49 }

```

---

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

Listing 6.1: 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) {

```

```
23         logger.info("Analysis failed for product id: {}",
24                       newInfo.getProductId(), e);
25         throw e;
26     }
27     if (!data.getWarnings().isEmpty()) {
28         //No handling needed at the moment
29     }
30 }
```

---

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

---

```
1  /**
2   * Runs analysis for given {@link DataProviderRequest} and {@link
3   *   DParserResponse}.
4   * Error are returned as list of {@link AnalysisErrorMessage}.
5   * Injected set of {@link Analysis} is executed one by one,
6   *   result unwrapped and kept if present.
7   * Set of analysis result error messages is returned.
8   *
9   * @param request      dp request
10  * @param parserResponse parsed data
11  * @return list of errors or empty (if result was valid)
12  */
13 @Override
14 public List<AnalysisErrorMessage> runAnalysis(DataProviderRequest
15   request, DParserResponse parserResponse) {
16     return analysisSet.stream()
17       .map(x -> x.analyse(request, parserResponse))
18       .filter(Optional::isPresent)
19       .map(Optional::get)
20       .collect(Collectors.toList());
21 }
```

---

## 6.2 Plánování práce

Plánování práce bylo rozděleno do tříd částí, nalezení adres, vytvoření požadavků a samotné odeslání. Tato kapitola se pak 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 popsanych v návrhu. Následně jsou vráceny, ale rozdělené dle způsobu nalezení. Nalezení adres nově počítá i s frekvencí, která je nastavena u aktivní kampaně.

Nalezené adresy všechny typů jsou poté vyfiltrovány o nežádoucí stavy. K tomuto účelu jsem navrhl rozhraní kontrol, rozhodující o tom, zda je požado-

vané adresu použít. Návrh se pak podobá spuštění kontrol analyzátoru.

## 6.3 Oprava komunikace Manager - ProductProvider

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

Celý proces v Managerovi je rozdělen do tří částí a to nalezení adres, vytvoření požadavků a samotné odeslání. Hlavní metoda servisní třídy *DPrequestSenderServiceImpl* 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 }
```

---

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. Nový návrh vytváření požadavků tento proces deleguje do nové třídy *DPrequestBuilderServiceImpl*. Návrátový typ této třídy pak slučuje množinu požadavků obsahující adresy bez produktů a ty v aktivní kampani, jelikož po vytvoření požadavků, již není potřeba odlišné chování k takovým požadavkům. Vzhledem k potřebě pracovat entitou obsahující adresu detailu i při odesílání požadavků, obsahuje *DPrequestProductUrlWrapper* množiny obsahující dvojice požadavek a adresa detailu.

### 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 zpraco-

vání. V případě neočekávané chyby při odesílání, jsou tyto příznaky změněny na údaj obsahující informaci o chybném odeslání.

Samotné odeslání má však 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 6.3: 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  */
19 void send(DPRequestProductUrl requestProductUrl,
20           Consumer<DPRequestProductUrl> persistConsumer,
21           Consumer<DPRequestProductUrl> revertConsumer) {
22     try {
23         persistConsumer.accept(requestProductUrl);
24         providerRequestHandler.send(
25             requestProductUrl.getDataProviderRequest());
26     } catch (MQConnectionException e) {
27         revertConsumer.accept(requestProductUrl);
28         logger.error("Sending dataProviderRequest error.", e);
29         throw new IllegalStateException(e);
30     }
31 }
```

---

Zde je nutné podotknout důvod, proč není po odchytnutí a zpracování výjimky vrácena opět *MQConnectionException*. Důvodem je zvolený způsob iterace nad objekty a samotného volání odesílací metody. Metoda je volána pomocí rozhraní umožňující prvky funkcionálního programování v jazyce Java. Neumožňuje však volat metodu vyhazující výjimku rozšiřující třídu *Exception*.



Tento nedostatek architektury lze pak vyřešit pomocí *IllegalStateException*, která potomkem třídy *Exception* není.

---

Listing 6.4: Volání metody, která odesílá 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

Po úspěšném odeslání je třeba staženou stránku použít pokud existuje. Nejprve bych pak chtěl popsat změnu komunikační třídy *DataProviderRequest*. Tato komunikační třída představuje jeden požadavek odeslaný pomocí front a skládá se z jednotlivých základních atributů a fragmentů. Fragmentem je zde myšleno složitější struktura, například komunikační třída obsahující informace o produktu. Některé tyto atributy či fragmenty však nemusejí být nastaveny, konkrétně stažená stránka, produkt nebo šablona. U těchto atributů a fragmentů jsem proto provedl změnu u *get*, aby vracely kontejner *Optional*. Tím bylo jasně indikovaná možnost, že hodnoty nemusí být obsaženy.

V rámci *DataProvider* pak pouze 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 následující ukázce:

Listing 6.5: Veřejná metoda třídy *DPDownloaderServiceImpl* zajišťující stažení detailu 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)
```

```
13         );  
14     }
```

---

## 6.4 Spojení chyb analyzátoru a vylepšení rozhraní

Oprava více četnosti chyb a samotného rozhraní je posloupnost několika oprav. První byla již zmíněna při odstranění přebytečných výjimek, což zajistilo spouštění všech validací. Další krok je změna komunikačního objektu, který nové musí uchovávat všechny chybné validace a přesnější informace o datech použitých při parsování.

Původní řešení obsahovalo dvě třídy určené pro odpověď. Jedna představovala validní odpověď, *DataProviderResponse*. Druhá chybu, *DataProviderResponseError*.

Z důvodu velkého množství podobných atributů, obzvlášť po přidání vyparsovaných hodnot, jsem změnil návrh těchto objektů. Z *DataProviderResponseError* jsem odstranil společné atributy a jako jeho rodiče jsem zvolil přímo *DataProviderResponse*. Chybná odpověď tak nové obsahovala i získané hodnoty.

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

Dále už pouze stačilo upravit webové rozhraní, aby odpovídalo změně databázové struktury. První změna se týkala samotného zobrazení informací o chybě. Využil jsem nově uložených dat. Administrátor má tak možnost vidět použité hodnoty při analyzování a všechny chyby. Poslední změna už pouze spočívala v zpracování vstupů od administrátora, kdy bylo potřeba uložit všechny možné příznaky pro budoucí analyzování. Přidal jsem pak také možnost přesměrovat administrátora do rozhraní, kde může změnit šablonu pro parsování stránky, jelikož je možné, že analýza je chybná z důvodu změněné struktury stránky.

## 6.5 Monitorování

Na virtuální server jsem nasadil službu DataDog [41], která po jednoduché instalaci umožňuje sledování běžících služeb a vytížení serveru. Data jsou odesílány přímo do služby DataDog. Webové rozhraní poté 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 pak možné sledovat například výsledek sestavení v Jenkins nebo obsah a využití RabbitMQ front.

## 6.6 Získání adres obchodů a příslušných detailů produktů

Vzhledem pouze 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 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 se seznamů internetových obchodů v České republice a ty manuálně vložit do databáze.

Module Finder byl zcela odstraněn a nahrazen modulem novým, nazvaným ProductDetailProvider. Tento modul zajišťuje hledání detailů produktu, což 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 samotné vytvoření požadavku je nutná 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 pro tento proces, bylo vytvořeno již v rámci týmového projektu a bylo tedy využito.

## 6.7 Párování produktu

Byl navržen proces, který se nejprve pokusí produkt spárovat automaticky, pokud nalezne přímou shodu názvu, EANu nebo modelového čísla. Pokud se spárování nepodaří, jsou provedeny heuristiky hledající pravděpodobné shody. Z množiny těchto možností je pak vytvořena chyba, kterou musí zpracovat administrátor.

Pro tuto možnost a zpracování bylo poté nutné vytvořit webové rozhraní, které administrátorovi umožňuje jednoduché přiřazení adresy k produktu nebo všechny možnosti odmítnout.

### 6.8 Detekce neexistující stránky a nenalezeného produktu

V případě použití interního vyhledávání byl běžný stav, kdy pro hledanou hodnotu nebyl nalezený žádný produkt, což systém zpracoval jako chybnou adresu, jelikož se nepodařilo získat žádnou adresu detailu. Podobný případ byl při nalezení adresy detailu, která ač je obchodem vrácena jako výsledek, tak reálně neexistuje. I pro tuto možnost byla vytvořena chyba šablony, akorát pro jiného typu, jelikož stránka neobsahovala žádné hodnoty.

Jako řešení jsem nejprve zvolil řešení, kdy jsem se pokoušel stránku, kde není žádný výsledek porovnat se stránkou, která byla na obchodě vrácena pro náhodný a nesmyslný řetězec dlouhé délky. Myšlenka byla, že pokud produkt opravdu neexistuje bude pro nesmyslné hledání vrácena podobná stránka.

Problém toho řešení se ukázala přílišná odlišnost HTML stažených stránek a specifčnost obchodů, kde tento algoritmus fungoval. Proto jsem zvolil řešení, kdy administrátor má při řešení chyby možnost zadat řetězec značící neexistenci produktu (popř. neexistující stránky detailu produktu). Tento řetězec je poté pro každý obchod specifický a před vytvořením příslušné chyby šablony je nejdříve zkontrolováno, zde pouze se produkt pouze na obchodě nenachází (popř. stránka neexistuje).

### 6.9 Více šablon detailů produktu

Původní návrh počítal s možností, kdy každá stránka pro detail produktu obsahuje stejnou strukturu napříč celým obchodem. Tento předpoklad se však ukázal jako chybný, kdy například v případě slevy je element obsahující cenu odlišný.

Jiná struktura pak způsobila chybu šablony. Z toho důvodu jsem implementoval 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.

### 6.10 Více stejných chyb

Systém se i po změnách potýkal se stavem, kdy se v administrátorském rozhraní objevilo více chyb šablon nebo analyzátoru. Tento stav nastával v případě, kdy neexistovala šablona pro detail nebo vyhledávání na obchodě, což v případě více požadavků týkajícího se stejného obchodu, způsobilo vytvoření pro každý požadavek chybu.

Chyby lze však predikovat a v případě neexistující šablony odeslat pouze jeden takový požadavek. 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.

Nejprve jsem upravil webové rozhraní, aby chyby týkající se stejné stránky zobrazovala v seznamu chyb pouze jednou. Po zpracování jsem poté přidal algoritmus, který nastaví všechny chyby stejného typu jako vyřešené, čímž odpadne nutnost administrátora všechny vyřešit.

## 6.11 Skladem

V rámci získávání dat u detailu produktu jsem přidal možnost získávat, zda je produkt skladem.

Pro úpravu jsem upravil nejdříve rozhraní pro vytváření šablony a strukturu databáze, aby uchovávala atribut u šablony a výsledku parsování. Poté jsem pouze zajistil, aby byl tento atribut zahrnut v rámci komunikace modulu Manager a DataProvider, kde bylo potřeba zajistit správné uložení do komunikační třídy, samotné získání hodnoty a korektní uložení do databáze.

## 6.12 Ostatní

Při realizaci vylepšení jsem také provedl několik menších změn a oprav, které jsem objevil při vývoji. Jedna ze základních a triviálních změn byla například oprava *jar* souboru sestavené aplikace, která byla zbytečně velká. Velikost byla způsobena externími knihovnami, které byly obsaženy v *jar* vícekrát a to pro každý modul. Oprava pak spočívala pouze v přidání jednoho řádku do *build.gradle*, který specifikuje strukturu projektu a nastavení sestavení pro gradle.

Listing 6.6: Nastavení sestavení *jar* souboru, aby neduplikoval knihovny na kterých jsou moduly závislé.

---

```
1 jar.duplicatesStrategy = DuplicatesStrategy.EXCLUDE
```

---

## 6. REALIZACE VYLEPŠENÍ

Chyba analyzátoru

**Doručujeme i v sobotu a v neděli. Až k vám domů, na kteroukoliv prodejnu a do Alzaboxů.**


Vážení zákazníci, dne **1.5.** jsou otevřeny všechny naše prodejny kromě **Příbrami**.

**alza media** Zkuste nové Alza Premium | Moje Alza | Přihlásit | J





Co hledáte? např. kniha Jo Nesbo...

Alza.cz Media a zábava Hračky Parfémy a hodinky MAXI drogerie

Zobrazit katalog > > **Filmy** > **Krimi** > Bílý Elephant



**Bílý Elephant**

Elephant White    

VŠECHNY FILMY Jak přehrát

Google Chrome umožňuje sledovat začátek filmů  
Film je možné přehrávat na území České republiky

USA | 2011 | 91 min | SD (576p)

Informace o erroru

Info	Vyparsovaný produkt	Informace o produktu
<ul style="list-style-type: none"><li>Timestamp: 2017-04-28 21:36:14</li><li>Info:</li><li>ProductId: 1</li></ul>	<ul style="list-style-type: none"><li>Timestamp: 2017-04-28 21:10:32</li><li>Description:</li><li>Ean:</li><li>Name: Bílý Elephant</li><li>PN:</li><li>Cena: .-</li><li>Cena s dph: .-</li><li>Skladem info:</li><li>URL: <a href="https://www.alza.cz/media/bily-elephant-d1527386.htm?o=1">https://www.alza.cz/media/bily-elephant-d1527386.htm?o=1</a></li></ul>	<ul style="list-style-type: none"><li>Description:</li><li>Ean: 5038483539941</li><li>Name: AIR BAG - přírodní zdravotní pantofle</li><li>PN: F215311016 (43)</li></ul>

Nalezené errorry

DIFFERENT\_EAN  
*Parsed ean value[111] doesn't match known ean value[123]*

DIFFERENT\_NAME  
*Parsed name[111] doesn't match known name[123]*

☐ Reject  
☒ Ok  
☐ Create new parse template

Obrázek 6.1: Ukázka webového rozhraní administrátora po provedených vylepšení. Chyba byla vytvořena uměle za účelem prezentace změn.

## Zhodnocení provedených vylepšení

V této kapitole se budu nejprve věnovat konečné funkcionalitě systému. V návaznosti na výsledný stav poté popíšu jaké úpravy by byly vhodné, aby tento stav řešily.

### 7.1 Funkcionalita

Nejpodstatnější rozdíl při porovnání starého řešení a nového je samotná funkcionalita, která byla značně rozšířena. Především je možné reálné použití celého systému. Ačkoliv není schopný nalézt obchody samotné, poskytuje rozhraní pro jejich manuální přidání.

Pokud je známo jaké obchody chceme sledovat, systém umožňuje reálný a kontinuální běh, který poskytuje výjimky. Nevýhodou a hlavní nedostatkem je zde však velký důraz na administrátora při prvotním nastavení.

I po provedení nutných úkonů jako samotné nastavení šablon parsování detailu, zbývá párování nalezených adres s produkty. Při testování bylo ověřeno, že velké množství obchodů neobsahuje správné nebo odlišené identifikátory produktů. Nejčastější je výskyt pouze jména, které neodpovídá tomu uloženému.


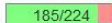
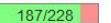
Z toho důvodu je nutné párování provést ve velké množstvím případů manuálně. Systém se manuální párování snaží ulehčit rozhraním a návrhům, nicméně i při malém množství produktů hledaných na více obchodem je počet těchto chyb velmi rozsáhlých.

Problém párování by bylo možné pak vylepšit, pokud by systém uchovával více identifikátorů. Především více uložených jmen by pak zátěž na administrátora mohla klesnout.

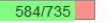
## 7.2 Návrh a testy

Zásadní refactoring a změna návrhu komunikace tříd velmi pozměnila interní část. Interní část kromě toho, že vyhovuje více nárokům na požadovanou funkcionalitu, je složena z daleko přehlednějšího a lépe udržitelného kódu.

Lepší udržitelnost byla výrazná především při přidávání vylepšení, kdy tyto změny často byly triviální záležitostí a po jejich provedení bylo zachována původní funkcionalita. Toto dávám za důsledek především lepšího pokrytí testů a dekompozice tříd. Oproti původní řešení narostl počet jednotkových testů ze **170** na **492**. Pokrytí pak demonstruje následující vizualizace.

Name	Packages	Files	Classes
Cobertura Coverage Report	80% 	83% 	82% 

Methods	Lines	Conditionals
79% 	79% 	66% 

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

Oproti původnímu řešení, vzrostlo pokrytí zhruba o 20%. Základní funkcionalita servisních tříd je, až na výjimky testy pokrytá celá. Zbylé neotestované třídy jsou pak především inicializační třídy front a jejich komunikační rozhraní, které je závislé na jiné běžící službě.

I po provedení těchto změn zůstalo omezení nemožnosti vytvoření více instancí jednotlivých modulů, nově ProductDetailProvider a DataProvider. Tato změna je však vzhledem k návrhu poměrně triviální, kdy pouze stačí přidat instanční rozhraní jednotlivých modulů, jelikož momentálně jsou spouštěny přímo modulem Manager. Následující komunikace pak probíhá pouze přes fronty, tudíž není oddělení nezpůsobí další komplikace, kromě větší zátěže na nastavení infrastruktury nasazení aplikace na serveru.

## 7.3 Rozhraní administrátora

Systém se stal z hlediska pro administrátora uživatelsky přívětivější. Při řešení chyb je rozhraní přehlednější a informace o chybách obsáhlejší, což umožňuje rozhodnutí založené na více faktech. Dále byla také podstatně zrychleno celkové zpracování po odpadnutí nutnosti řešit každý požadavek, ať už se jedná šablona pro stejný obchod či více chyb analyzátoru.



## **Závěr**



---

## Literatura

- [1] Extensible Markup Language (XML) 1.0 (Fifth Edition). 2008. Dostupné z: <https://www.w3.org/TR/2008/REC-xml-20081126/#sec-intro>
- [2] Virginia Tech - College of engineering: Department of computer science. 2002. Dostupné z: <http://courses.cs.vt.edu/~cs1204/XML/htmlVxml.html>
- [3] HTML5: A vocabulary and associated APIs for HTML and XHTML. 2014. 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. 2011. Dostupné z: <https://www.w3.org/TR/CSS21/selector.html#q5.0>
- [5] Rozhovor s Jiřím Hunkou, nar. 12.5.1985, provozovatel eshopů. 28-11-2016 2016.
- [6] Heuréka. Dostupné z: <http://www.heureka.cz>
- [7] Zboží. Dostupné z: <http://www.zbozi.cz>
- [8] Heuréka - Sortiment Report. Dostupné z: <https://sluzby.heureka.cz/napoveda/sortiment-report/>
- [9] Price checking. Dostupné z: <http://www.price-checking.cz/>
- [10] Pricing intelligence. Dostupné z: <http://pricingintelligence.cz/>
- [11] Sledování trhu. Dostupné z: <http://www.sledovanitrhu.cz/>
- [12] Pricebot. Dostupné z: <http://www.pricebot.cz>
- [13] Screen scraper. Dostupné z: <http://www.screen-scraper.com>
- [14] Web extractor. Dostupné z: <http://www.webextractor.com>

- [15] The Java™ Tutorials. 2015. Dostupné z: <https://docs.oracle.com/javase/tutorial/jdbc/overview/database>
- [16] Git –fast-version-control. 2017. Dostupné z: <https://git-scm.com/>
- [17] Huizinga, D.; Kolawa, A.: *Automated defect prevention: best practices in software management*. IEEE Computer Society, c2007, ISBN 9780470042120.
- [18] Continuous Integration. 2006. Dostupné z: <https://www.martinfowler.com/articles/continuousIntegration.html>
- [19] RabbitMQ by Pivotal. 2011. Dostupné z: <https://www.rabbitmq.com/>
- [20] Redmine. 2017. Dostupné z: <https://redmine.org/>
- [21] GitLab. 2017. Dostupné z: <https://gitlab.com/>
- [22] Jenkins. 2017. Dostupné z: <https://jenkins.io/>
- [23] SonarQube. 2017. Dostupné z: <https://www.sonarqube.org/>
- [24] Nette. 2017. Dostupné z: <https://nette.org/>
- [25] Composer. 2017. Dostupné z: <https://getcomposer.org/>
- [26] Gradle. 2017. Dostupné z: <https://www.gradle.org/>
- [27] Maven Repository. 2017. Dostupné z: <https://mvnrepository.com/>
- [28] Cobertura. 2017. Dostupné z: <http://cobertura.github.io/cobertura/>
- [29] The Java™ Tutorials. 2015. Dostupné z: <https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html>
- [30] Google Guice. 2017. Dostupné z: <https://github.com/google/guice/>
- [31] Hibernate. 2017. Dostupné z: <http://hibernate.org/>
- [32] Apache Commons. 2017. Dostupné z: <https://commons.apache.org/>
- [33] Standard - ECMA - 404: The JSON Data Interchange Format. 2013: s. 1–14. Dostupné z: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>
- [34] Introducing JSON. Dostupné z: <http://www.json.org/>
- [35] The Java™ Tutorials: Exceptions. 2015. Dostupné z: <https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html>

- [36] Beck, K.: *Test-driven development: by example*. Addison-Wesley, c2003, ISBN 0321146530.
- [37] Fowler, M.: *Refactoring: zlepšení existujícího kódu*. Moderní programování, Grada, 2003, ISBN 8024702991.
- [38] Scott, M. L.: *Programming language pragmatics*. Morgan Kaufmann Pub., druhé vydání, c2006, ISBN 0126339511.
- [39] Java™ PlatformStandard Ed. 8. 2016. Dostupné z: <https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>
- [40] Java™ PlatformStandard Ed. 8. 2016. Dostupné z: <https://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html>
- [41] DataDog Docs. 2017. Dostupné z: [https://docs.datadoghq.com/guides/basic\\_agent\\_usage/](https://docs.datadoghq.com/guides/basic_agent_usage/)



## Seznam použitých zkratek

**EAN** European Article Number

**XML** Extensible markup language

**HTML** Hypertext Markup Language

**CSS** Cascading style sheets

**JSON** JavaScript Object Notation

**HTTP** Hypertext Transfer Protocol

**DAO** Data Access Object

**URL** Uniform Resource Locator





## Obsah přiloženého CD

	readme.txt.....	stručný popis obsahu CD
	exe .....	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis .....	zdrojová forma práce ve formátu L <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	thesis.ps .....	text práce ve formátu PS