

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

22. 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 22. 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

Úvod	1
1 Cíl práce	3
1.1 Analytické cíle	3
1.2 Praktické cíle	3
2 Popis problematiky získávání informací z webů	5
2.1 Problematika	5
2.2 Výběr dat	6
2.3 XML Path Language	6
2.4 CSS Selector	6
2.5 Současný stav řešení potřeb internetových obchodů	7
3 Analýza týmového projektu	11
3.1 Cíl týmového projektu	11
3.2 Webové rozhraní	11
3.3 Interní část	12
4 Vývoj a implementace týmového projektu	15
4.1 Pojmy	15
4.2 Vývoj	16
4.3 Implementace	17
4.4 Má role	18
5 Zhodnocení týmového projektu	19
5.1 Pojmy	19
5.2 Stav	19
5.3 Nedostatky	20
6 Návrh na vylepšení	25

6.1	Pojmy	25
6.2	Refaktorování stávajícího řešení	26
6.3	Oprava komunikace Manager - ProductProvider	28
6.4	Plánování práce	28
6.5	Spojení chyb analyzátoru	29
7	Realizace vylepšení	31
8	Zhodnocení provedených vylepšení	33
	Závěr	35
	Literatura	37
A	Seznam použitých zkratk	39
B	Obsah přiloženého CD	41

Seznam obrázků

3.1	DataProvider diagram	13
5.1	DataProvider diagram	23

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 a možná řešení. 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.

Cíl práce

1.1 Analytické cíle

1. Rešerše aktuálního stavu získávání dat pro potřeby internetových obchodů
2. Analýza vzniklého řešení týmového projektu vzniklého v prostředí ČVUT FIT, včetně důrazu na použité postupy při softwarovém vývoji
3. Návrh a zhodnocení implementovaných vylepšení

1.2 Praktické cíle

1. Implementace vylepšení systému

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, je nejprve nutné služby zhodnotit a zanalyzovat jejich funkcionalitu.

2.1 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 dále kvůli unikátnosti webových stránek není možné jednoznačně určit jednotný a zcela automatizovaný postup, jak data získat v požadovaném formátu.

2.2 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 pomocí dvou možností:

1. XPath
2. CSS Selector

2.3 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 prohlížeč rozumí. Díky této vlastnosti lze použít XPath pro definování cesty k prvku (a jeho obsahu), který uchovává potřebnou informaci na webové stránce.

2.4 CSS Selector

Jazyk CSS je používán pro vizuální popis prezentace webové stránky definované v HTML. Jazyk k určení prvků se kterými chce pracovat používá selektory, které označují prvek v HTML. Jako selektor může být použit jak samotný název prvku, tak vlastní definované třídy.[4]

Pomocí řetězení těchto selektorů jsme schopni jednoznačně získat element v HTML.

2.5 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 hlavních atributů:

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

2.5.1 Srovnávače cen

Data lze získat pomocí srovnávačů cen jako *zbozi.cz*[5] nebo *heureka.cz*[6]. Problém u těchto spočívá v určení pro koncové zákazníky, kterým umožňuje pro nalezení nejlepší ceny na trhu pro hledaný produkt. S tím souvisí to, že největší srovnávače cen neposkytují 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 obchodům. [7]

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 je, že data neobsahují žádná jména konkurenčních obchodů [?]. Vzhledem k povaze struktury a splatnosti generovaných dat je také nemožné ceny sledovat v časovém období. Ostatní srovnávače mají údajně výstup velmi podobný nebo jako bylo výše řečeno data neposkytují. Díky tomu se ukázali srovnávače jako nedostatečný zdroj dat.[7]

2.5.2 Existující služby

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

Služby mají v zásadě velmi podobnou povahu služeb. Rámcově se jedná o porovnávání cen včetně historie na různých internetových obchodem či na srovnávacích cen. 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 srovnávacích. 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.

2.5.2.1 Price checking

2.5.2.2 Pricing intelligence

2.5.2.3 Sledování trhu

2.5.2.4 Pricebot

2.5.2.5 Zahraniční nástroje

Tyto nástroje jsou obecněji zaměřené a obvykle požadují od uživatele techničtější zaměření, jelikož je nutné přesně specifikovat kde, co a jak chce sledovat. Vzhledem k tomuto omezení nejsou přímo pro provozovatele e-shopů vhodné kvůli nedostatečným technickým kapacitám a pro tuto práci důležité.

Bodový seznam zahraničních nástrojů:

1. Screen scraper [8]
 - 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 [9]
 - Windows Aplikace
 - procházení zadaných stránek
 - hledání stránek pomocí klíčových slov
 - export do csv formátu
 - cena: \$99 - \$199 jednorázově
3. Web Scraper [10]

Analýza týmového projektu

V kapitole analýza týmového se budu věnovat řešení vytvořeného v rámci školní výuky na ČVUT FIT v akademickém roce 2015/16. Nejprve popíšu cíl, který měl projekt za úkol řešit, jaké byly použité postupy při vývoji a mou roli v tomto projektu.

3.1 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.

Hlavní funkcionalita systému spočívá v maximální možné míře automatizace získávání informací o produktech prodávaných konkurencí. Důraz je především kladem na optimalizaci počtu nutných lidských úkonů v rukách administrátora u kterého se předpokládá minimální technické vzdělání. Jediná nutná problematika, co musí administrátor znát je parsování HTML stránek.

Návrh popisuje rozdělení aplikace na část poskytující veškeré webový rozhraní a na část zpracovávající všechny interní procesy. Vzhledem k požadavkům na škálovatelnost aplikace, druhá část se skládá z více samostatných menších služeb - modulů komunikující spolu pomocí front. Díky tomu, že každý modul zajišťuje určitou funkcionalitu umožňující vytvářet více jeho instancí, je možné procesy zpracovávat paralelně. Uživatelská a interní část spolu sdílí data pomocí relační databáze[11].

3.2 Webové rozhraní

Webové rozhraní lze rozdělit na dvě části. Uživatelskou část obsahující množinu podstránek určených pouze pro konečné uživatele služby. Uživatelská část umožňuje vytvořit kampaň. Kampaň je proces trvající určitý časový úsek, který sleduje vložené produkty u konkurence. V těchto běžících kampaních

má poté uživatel možnost uživatel vidět vizualizaci získaných dat, případně je umožněn export dat do formátu csv nebo xlsx. Získaná data obsahují, kde se sledované produkty nacházejí a za jakou cenu se prodávají.

Druhá část je určena pouze pro administrátory a slouží k monitorování kampaní uživatelů a řešení chyb, které systém není schopný vyřešit. Chyby jsou typicky problémy s parsováním webových stránek, párování produktu ke stránce nebo potvrzení zda jsou získaná data validní.

3.3 Interní část

Interní část je rozdělena do samostatných modulů, které spolu komunikují pomocí front. Moduly jsou detailně popsány v následujících podsekcích.

3.3.1 Manager

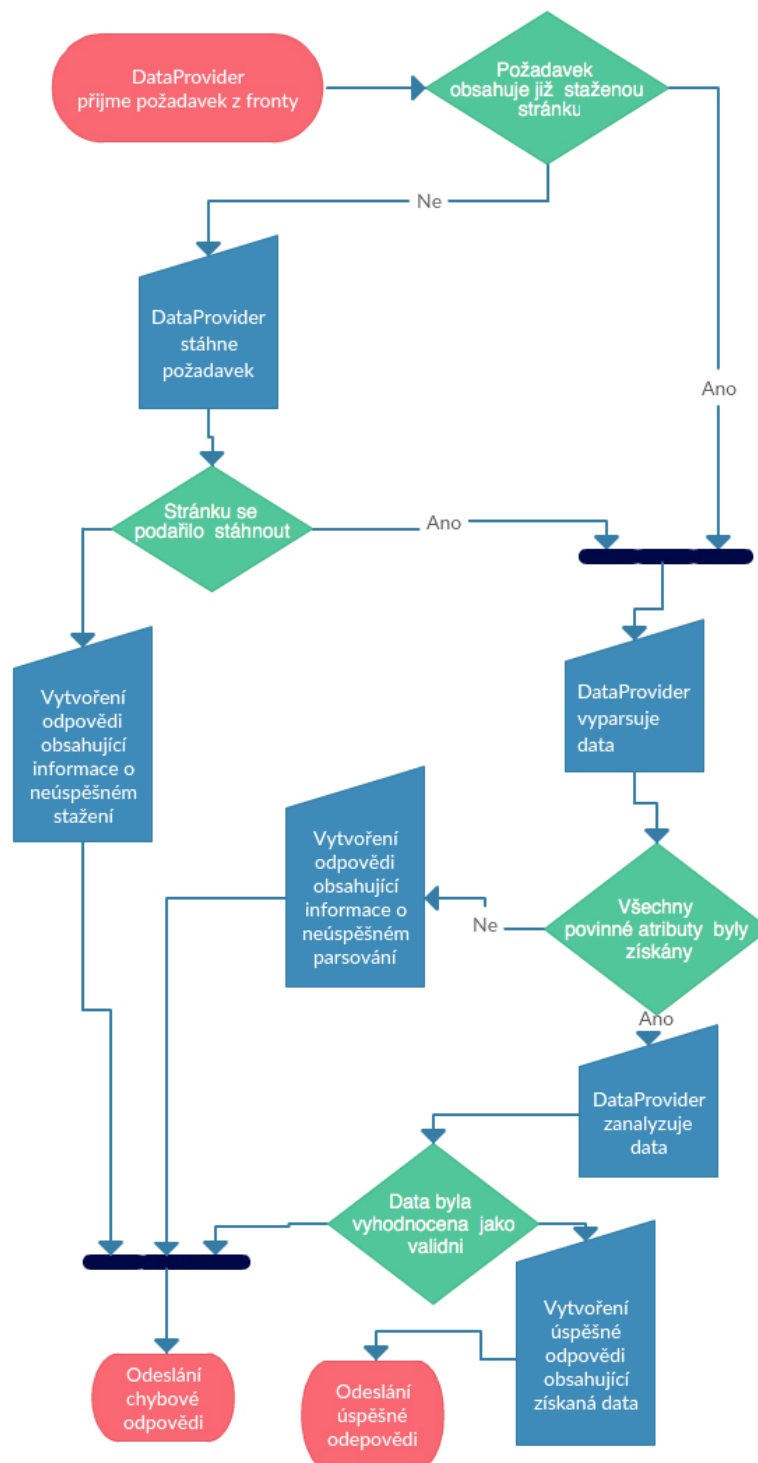
Manager je hlavní modul, který jako jediný má možnost připojení přímo do sdílené databáze a jeho instance může existovat pouze jednou. Manager má za úkol plánování práce pro ostatní části systému a zpracování vstupů a výstupů z front.

3.3.2 Finder

Finder je modul, který má za úkol získávat URL adresy internetových obchodů a na nich vyhledávat URL adresy vedoucí na požadované detaily produktů. Detailem produktu je myšlena webová stránka, kde jsou obsaženy podrobné informace o prodávaném produktu. Typicky je na takové stránce pouze jeden produkt, případně odkazy na jiné detaily prodáváného zboží.

3.3.2.1 DataProvider

DataProvider je module, který zpracovává adresy vedoucí na detaily produktu. Zde existují čtyři hlavní větve možností zpracování požadavků. Po stažení stránky, se z ní pokusí získat požadované hodnoty. V případě neúspěchu odešle chybovou hlášku, v opačném případě data zanalyzuje korektnost získaných cen vůči historickým datům pokud existují. Výsledek je poté odeslán k zpracování. „Managerem“



Obrázek 3.1: Diagram zobrazující aktivity v DataProvider modulu

Vývoj a implementace týmového projektu

Nakonec rozeberu výslednou funkcionalitu oproti nárokům na plně funkční systém.

4.1 Pojmy

4.1.1 Verzovací systém Git

Git je pro sdílení jednotlivých verzí každého vývojáře, umožňující jednoduchý přehled nad rozpracovanými částmi každého vývojáře. Úložiště systému se nazývá repositář, který obsahuje veškerý kód. Základní jednotkou tvoří verze, které jsou postupně vytvářeny vývojáře 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.

4.1.2 Jednotkové a integrační testy

Jednotkovými testy se rozumí sada kladných a záporných testů ověřujících funkcionalitu jedné třídy. Integrační testy pokrývají komunikaci více tříd.

4.1.3 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.

4.1.4 Průběžná integrace

Průběžnou integrací se rozumí sada nástrojů sloužící k rychlému nalezení případných chyb. Základní součástí je vzdálený repozitář, kde na základě každé jeho změny se vytváří nový build. Při buildu jsou poté spuštěny jednotkové a integrační testy. Na jejich základě je poté možné zjistit případné chyby.

4.2 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 vyučujícímu. Každý sprint se skládal z jednotlivých úkolů, které byly přiřazeny členům týmu. Stav úkolů byl uchovávan na systému Redmine, ten umožňuje přehledné řízení projektu a možnost sledování objevených chyb.

Jako verzovací systém byl zvolen systém Git, zajišťován službou Gitlab. Gitlab umožňuje možnost uchovávat repozitář na vzdáleném serveru a poskytuje webové rozhraní pro snadnou správu. Projekt byl v rámci repozitáře rozdělen na 4 části (větve):

- Master - hlavní větev uchovávající verze určené k nasazení na produkční server
- Develop - vývojová větev obsahující aktuální stav vývoje
- Feature - vedlejší větev vytvořená pro konkrétní úkol přidávající novou funkcionalitu
- Fix - vedlejší větev určená pro úkoly opravující naleznou 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 a Gitlab. Jenkins aplikaci zkompileoval, spustil testy a statickou analýzu kódu zajištěnou systémem SonarQube. Výsledky poté publikoval ve svém webovém rozhraní a zároveň v rozhraní Gitlab.

4.3 Implementace

4.3.1 Webové rozhraní

Webové rozhraní je implementováno v jazyce PHP verze 7. Základní kamenem aplikace zajišťuje aplikační rámec Nette. 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 vynucuje MVC architekturu, která odděluje prezentační a logickou vrstvu.

TODO = OBRAZEK MVC

Snadnou správu závislostí umožňuje balíčkovací systém Composer. Na základě definujícího souboru, kde jsou uloženy názvy požadovaných knihoven a jejich verze jsou automaticky stáhnuty z centrálního repositáře. Tím je mimo automatického stažení zajištěno, že každý člen týmu pracuje se stejnými verzemi knihoven.

4.3.2 Interní část

Interní část je implementována v jazyce Java verze 8. Kompilaci, spouštění testů a správu závislostí zajišťuje Gradle. Gradle je nástroj sloužící k automatickému sestavení aplikace. Umožňuje správu závislostí, které stahuje z centrálního repositáře. V rámci sestavení lze pustit testy, včetně přídatných doplňků. Projekt používá doplněk Cobertura, který na základě spuštěné testovací sady vytváří zprávu obsahující pokrytí větví 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 je poté možné poslat jako zpráva. Vzhledem k sdílené podobě objektu je poté možné na druhé straně objekt deserializovat zpět do Java objektu.

Použité knihovny:

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

4.4 Má role

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.

5.1 Pojmy

5.1.1 JSON

JSON označuje specifikaci formátu pro výměnu dat[12]. Jedná se o formát, který je čitelný jak pro lidské oko tak pro stroj[12] a jeho zpracování je implementováno pro velké množství programovacích jazyků[13]. Základní stavební jednotkou tvoří kolekce párů klíč a hodnotu spolu se seřazeným polem hodnot.

5.1.2 Web API

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

5.2 Stav

Ačkoliv stav projektu odpovídal nárokům na úspěšné odevzdání nebyla dosažena implementace všech procesů, aby byla umožněno reálně 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, byla odeslána chyba zpět do Manageru. Manager výsledek korektně a 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.

5.3 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.

5.3.1 Vytváření požadavků pro ProductProvider

Následující nedostatky jsou úzce spjaté s tím, jak aplikace plánovala práci. Plánováním 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.

5.3.2 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.

5.3.3 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. 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.

Informace byly následně odeslány a manager na jejich základě vytvořil chybu pro administrátora k vyřešení. Administrátor mohl chybu označit jako chybnou. Na základě označení bylo poté vytvořeno nastavení, že daná chyba se má na celém obchodě ignorovat. V opačném případě se adresa přestala používat. 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.

5.3.4 Vytváření chyb šablon

Další problém se ukázalo plánování práce založené na kritériu, kdy jsou k vytvoření požadavku vybrány všechny adresy, které neobsahují šablony. Systém odeslal požadavek pro všechny uložené adresy na obchod. To vyústilo ve vytvoření mnoho chyb, které musel administrátor všechny postupně vyřešit.

5.3.5 Modul Finder

Module Finder nebyl zapojený do systému. Existovala pouze hlavní implementace interních procesů, jejichž funkčnost byla pouze ověřena jednotkovými testy. Neexistují 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 a na něm pomocí interního vyhledávání nalezne detaily produktů, na kterých je velká pravděpodobnost, že patří hledanému produktu. To je však třeba ověřit. Po získání hodnot ze stránky je třeba nežádoucí odkazy vyloučit a ostatní se pokusit spárovat s produktem.

5.3.6 Plánování práce

Poslední nedostatek se skládal z absence 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.

5.3.7 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 implementace 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 přídatné 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 [14]. Pro uchování přídatných 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, vytváření požadavků pro DataProvider, hlavní servisní třída DataProvideru nebo validace dat analyzátořem. Z tohoto důvodu jakákoliv oprava nebo implementace no-

5.3. Nedostatky

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 5.1: Pokrytí testy vytvořeného řešení týmového projektu

vých požadavků mohla narušit stávající funkcionalitu 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.

Návrh 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.

6.1 Pojmy

6.1.1 Mock

V objektově orientovaném programování se Mock objekt používá pro simulování chování konkrétní třídy[15]. 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.

6.1.2 Refaktorování kódu

Refaktorování v softwarovém vývoji chápeme jako proces restrukturalizace existující kódu, aniž by byla pozměněna funkcionalita. Refaktorování se provádí za účelem dosáhnout průhlednějšího a čitelnějšího kódu, který se lépe udržuje a rozšiřuje [?]. Refaktorování kódu se především provádí při existenci konstrukcí značící špatný návrh aplikace.

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

- Dlouhá metoda

- Velká třída
- Dlouhý seznam parametrů
- Složité struktury podmínek

6.2 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 jsem se rozhodl přepsat téměř celou interní část, tak aby bylo možné kód lépe udržovat. Zároveň se pokusím z důvodů efektivních, zachovat co nejvíce původních částí. Obzvlášť takové, které jsou ověřeny že fungují pomocí testů a testování. Při přepsání se tedy budu především věnovat nastavení komunikace jednotlivých tříd mezi sebou. Kvůli tomu bude potřeba změnit jejich komunikační rozhraní, nicméně vnitřní logická funkcionalita by měla ve většina případů zůstat stejná. Přepsání nejprve popíšu obecně, což se týká všech interních částí aplikace.

6.2.1 Servisní třídy

Pro větší přehlednost bylo třeba přesunout všechny servisní třídy vykonávající do samostatného balíku, které leží v nadřazeném balíčku „service“. Servisní třídou jsou takové, které nespádají do ani jedné z těchto skupin:

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

6.2.2 Řízení aplikace

Aplikace, především v modulu ProductProvider, byla řízena pomocí chytání výjimek obsahující informace o chybě. Výjimky jsem se rozhodl odstranit a návratové hodnoty změnit na objekt obalující celkový výsledek. Ten obsahuje, zda proces skončil validně a na základě tohoto příznaku, obsahuje buď data, která byla vrácena původně nebo informace o chybě či chybách.

Příklad takového objektu, lze demonstrovat na následující struktuře, která byla redakčně zkrácena oproti originálu.

```
/**
 * Entity to keep parsed response. Almost every
 * attribute can be null,
 * so getters return {@link Optional} of nullable type.
 *
 * @author Jakub Tucek
 * @created 24.1.2017
 */
public class DParserResponse {

    /**
     * Flag for keeping result of parsing
     */
    boolean finishedProperly;

    /**
     * Parsed name of the product
     */
    private String name;

    public boolean isFinishedProperly() {
        return finishedProperly;
    }

    public void setFinishedProperly(
        boolean finishedProperly) {

        this.finishedProperly = finishedProperly;
    }

    public Optional<String> getName() {
        return Optional.ofNullable(name);
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Tato struktura je použita jako návratový typ pro rozhraní a implementaci 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 možné pouze pomocí *get* a *set* metod. To zajišťuje odstínění tříd, které k datům přistupují od implementačních detailů a odstínění od irelevantních detailů implementace [16]. 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 [17]. Před přístupem k hodnotě se proto musí nejdříve programátor objektu zeptat, zda obsahuje hodnotu. Mechanismus vynucení ověření prázdnosti hodnoty pak zamezuje nežádoucím výjimkám, především v tomto případě častých *NullPointerException* [18].

6.3 Oprava komunikace Manager - ProductProvider

Jak bylo zmíněno výše, bylo nejprve třeba komunikaci existujících dvou modulů optimalizovat. Neefektivní chování je velký problém při zpracování velkého počtu dat, především při 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 nestabilitu systému. Tento problém se úzce spojoval s tím, že je velmi obtížné pokrýt jednotkovými testy. Navíc tyto testy v kritických částech vůbec neexistují a proto každá změna může vyústit v neočekávané chování.

6.4 Plánování práce

Přepsání částí se také úzce týká otázky, jak pojmout plánování práce, určující jaké požadavky budou odeslány při spuštění kódu.

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 bez duplicit bylo také potřeba vyřadit ty, 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

Stav je tedy potřeba implementovat ideálně takovým způsobem, kdy bude velmi jednoduché kdykoliv kontrolu přidat nebo odebrat. K tomuto účelu jsem navrhl rozhraní, které na základě adresy vrátí příznak, zda adresa splňuje nebo nesplňuje prochází kontrolou. Příslušná třída poté kolekci těchto rozhraní, aniž by znala jejich implementaci použije, tak že iteruje těmito kontrolami a pokud splňuje adresa všechny kontroly adresu použije a vytvoří pomocí ní požadavek.

6.5 Spojení chyb analyzátoru

V případě detekce možné chyby při analyzování získaných požadavků jsou vytvářeny chyby, které jsou určeny pro vyřešení administrátorem.

Realizace vylepšení

Zhodnocení provedených vylepšení

TODO

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] Heuréka. Dostupné z: <http://www.heureka.cz>
- [6] Zboží. Dostupné z: <http://www.zbozi.cz>
- [7] Rozhovor s Jiřím Hunkou, nar. 12.5.1985, provozovatel eshopů. 28-11-2016 2016.
- [8] Screen scraper. Dostupné z: <http://www.screen-scraper.com>
- [9] Web extractor. Dostupné z: <http://www.webextractor.com>
- [10] Web scraper. Dostupné z: <http://www.webscraper.io>
- [11] The Java™ Tutorials. 2015. Dostupné z: <https://docs.oracle.com/javase/tutorial/jdbc/overview/database>
- [12] 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>

- [13] Introducing JSON. Dostupné z: <http://www.json.org/>
- [14] The Java™ Tutorials: Exceptions. 2015. Dostupné z: <https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html>
- [15] Beck, K.: *Test-driven development: by example*. Addison-Wesley, c2003, ISBN 0321146530.
- [16] Scott, M. L.: *Programming language pragmatics*. Morgan Kaufmann Pub., druhé vydání, c2006, ISBN 0126339511.
- [17] Java™ PlatformStandard Ed. 8. 2016. Dostupné z: <https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>
- [18] Java™ PlatformStandard Ed. 8. 2016. Dostupné z: <https://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html>

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

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 ^A T _E X
	text	text práce
	thesis.pdf	text práce ve formátu PDF
	thesis.ps	text práce ve formátu PS