Sem vložte zadání Vaší práce.

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

Bakalářská práce

InfoWeb - Nástroj získávání informací z webů

Vedoucí práce: Ing. Jiří Hunka

27. 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 27. dubna 2017 .....

Odkaz na tuto práci

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

Abstrakt

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 Nahraďte seznamem klíčových slov v češtině oddělených čárkou.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

Obsah

Úvod **1**

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

2 Analýza týmového projektu **13**
2.1 Cíl týmového projektu 13
2.2 Webové rozhraní 13
2.3 Interní část 14

3 Vývoj a implementace týmového projektu **17**
3.1 Pojmy 17
3.2 Vývoj 19
3.3 Implementace 19
3.4 Má role 21

4 Zhodnocení týmového projektu **23**
4.1 Pojmy 23
4.2 Stav 23
4.3 Nedostatky 24

5 Návrhy na vylepšení **29**
5.1 Pojmy 29
5.2 Refaktorování stávajícího řešení 30
5.3 Oprava komunikace Manager - ProductProvider 30

5.4 Plánování práce 31
5.5 Spojení chyb analyzátoru 32
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

6 Realizace vylepšení **35**
6.1 Refaktorování stávajícího řešení 35
6.2 Oprava komunikace Manager - ProductProvider 40
6.3 Plánování práce 40
6.4 Spojení chyb analyzátoru 41
6.5 Monitorování 41
6.6 Získání adres obchodů a příslušných detailů produktů 41
6.7 Párování produktu 41

7 Zhodnocení provedených vylepšení **43**
7.1 Více identifikátorů 43
7.2 Pokrytí testy 43

Závěr **45**

Literatura **47**

A Seznam použitých zkratek **51**

B Obsah přiloženého CD **53**

Seznam obrázků

1.1 Price checking 8

2.1 DataProvider diagram 15

3.1 Větve v Git repozitáři 18
3.2 MVC 20

4.1 Pokrytí testy vytvořeného řešení 27

7.1 Pokrytí testy po provedených vylepšení 43

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 prodávaných 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.

Kapitola 1

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, 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 specificitě 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.

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

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 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 zajímají obchody prodávající stejné zboží. Potřebné informace o prodávaných produktů 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. [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 pro 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 práce jsem měl možnost nahlédnout do dat, které *heuréka* poskytuje některým obchodům. [5]

Data obsahují následující informace:

- Informace o produktu - Segment, Kategorie, Jméno, ID, Výrobce, EAN, Item ID
- Url na vlastním obchodu
- Url na heuréce
- Počet konkurence a popularita na trhu
- Vlastní cena a pozice dle 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 neobsahují 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 jako bylo výše řečeno, data neposkytují. Díky tomu se ukázali 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 obchodem či na srovnávačích. 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 srovnávačích. 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

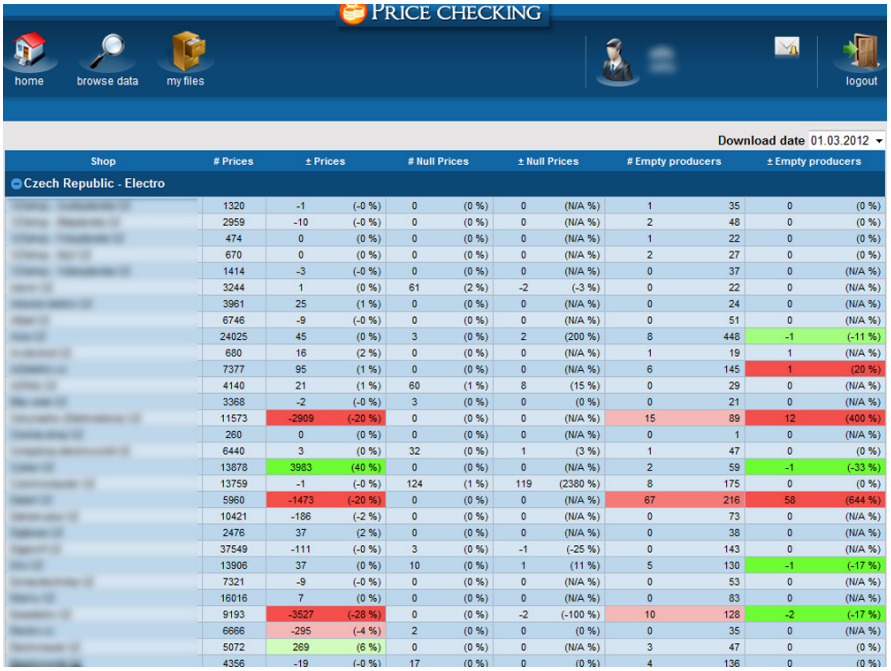
- webové rozhraní

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



Obrázek 1.1: Ukázka služby price checking

1.6.2 Pricing intelligence[10]

Hlavní funkce

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

Výstup

- formát xsl nebo pdf

Prostředí

- webové rozhraní

Data

- nespecifikované data a zaměřený trh

Cena

- 599 až 4999 Kč měsíčně

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

1.6.3 Sledování trhu[11]

Hlavní funkce

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

Vstup

- xml, xsl nebo manuálně

Výstup

- xsl nebo webový

Prostředí

- webové rozhraní

Data

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

Cena

- platba za každé vyhledání
- individuální cena

1.6.4 Pricebot [12]

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

Hlavní funkce

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

Vstup

- produkty ke sledování

Výstup

- pdf na email

Prostředí

- webové rozhraní

Data

- srovnávač cen Heureka.cz

Cena

- dle počtů produktů
- od 299 do 1299 Kč

1.6.5 Zahraniční nástroje

Tyto nástroje jsou obecněji zaměřené a obvykle požadují od uživatele 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.

Příklad zahraničních nástrojů:

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

Kapitola 2

Analýza týmového projektu

V 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.

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 prodávaných 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.

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 sdílejí data pomocí relační databáze[15].

2.2 Webové rozhraní

Webové rozhraní lze rozdělit na dvě části. Uživatelskou část, obsahující množinu podstránek 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 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 automaticky vyřešit. Chyby jsou typicky problémy s parsováním webových stránek, párování produktu ke stránce nebo potvrzení zda jsou získaná data validní.

2.3 Interní část

Interní část je rozdělena do samostatných modulů, které spolu komunikují pomocí front. Moduly je poté možné spustit jako služby ve více instancích. Vzhledem k možnostem front, je pak možné práci distribuovat na více serverech, aniž by byla ohrožena bezpečnost databáze, jelikož k té je možný přístup pouze lokální. Moduly jsou detailně popsány v následujících podsekcích.

2.3.1 Manager

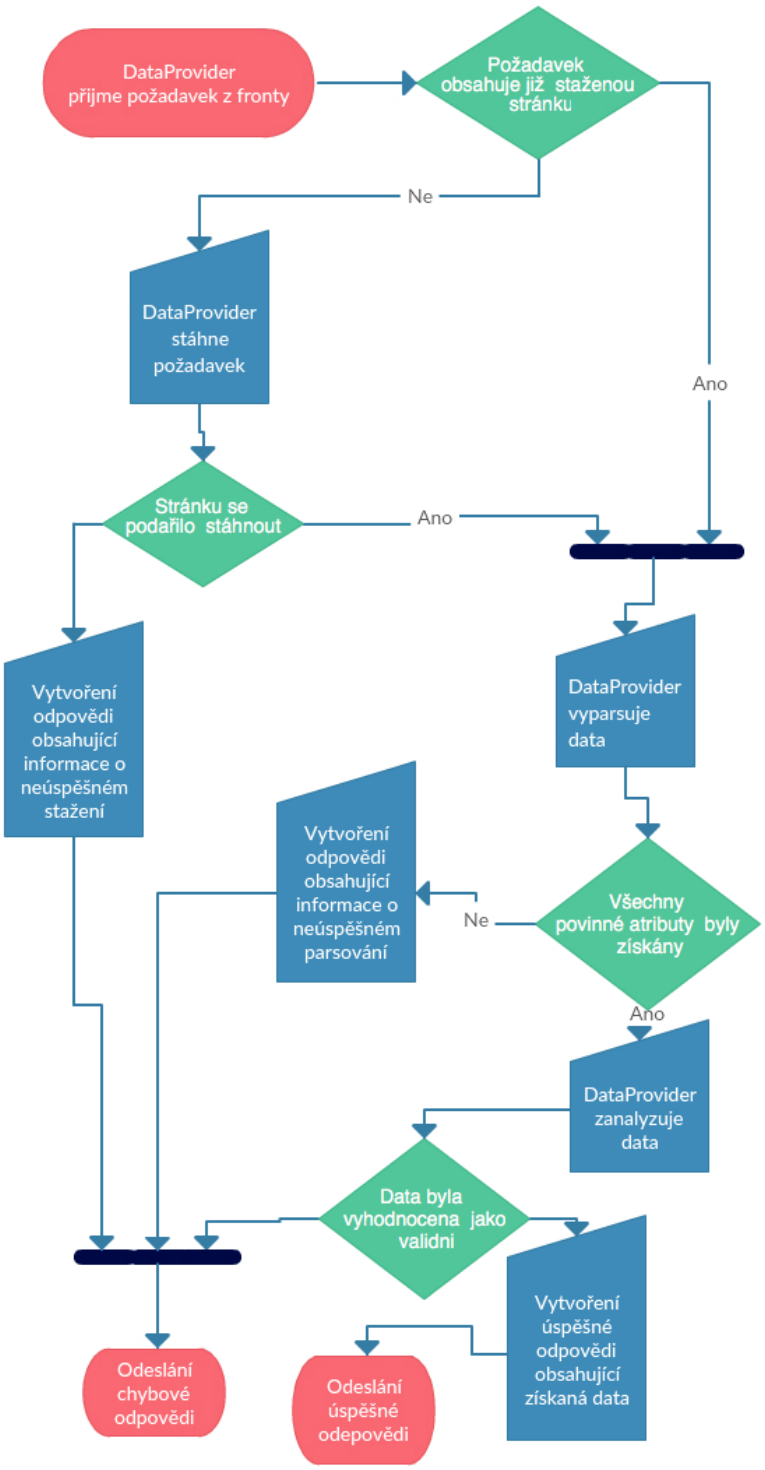
Manager je hlavní modul, který má jako jediná interní část možnost připojení do databáze a jeho běžící instance může existovat pouze jednou. Manager má za úkol plánování práce pro ostatní části systému a zpracování vstupů a výstupů z front.

2.3.2 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 obsažený podrobné informace o prodávaném produktu. Obchody samotné jsou získávány pomocí vyhledávání na internetových srovnávačích. K vyhledání je použit název produktu.

2.3.3 DataProvider

DataProvider je modul, 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 příslušnou chybu, v opačném případě zanalyzuje korektnost získaných cen vůči historickým datům pokud existují. Výsledek je poté odeslán k zpracování „Managerem“.



Obrázek 2.1: Diagram zobrazující aktivity v DataProvider modulu

Kapitola 3

Vývoj a implementace týmového projektu

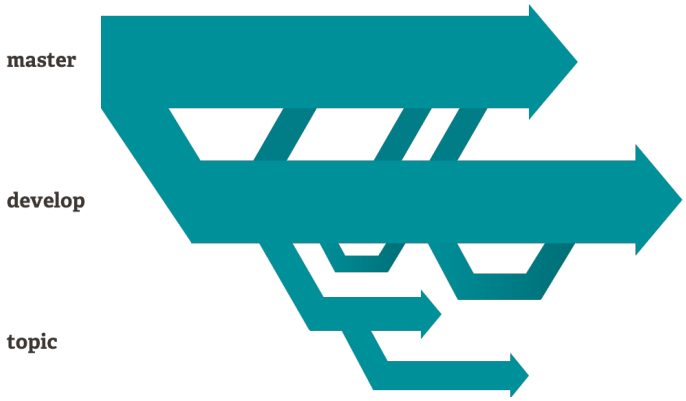
V této kapitole se věnuji průběhu vývoje týmového projektu a vytvořenému řešení. Poslední část rozebírá mou roli v tomto projektu, jelikož jsem věděl, že budu téma dále rozvíjet v rámci bakalářské práce. Z tohoto důvodu jsem se projektu věnoval nad rámec předmětu.

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 verzi a zároveň serverové. Sdílený repozitář pak zajišťuje distributivitu mezí všemi potřebnými členy. Pro lepší správu pak existují nadstavby nad serverovou částí repozitáře, které umožňují jednoduchou správu nad kóde a spouštění úkolů v závislosti na definované akce. 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 větví 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 slučovaní 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 udržovatelnost 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 vyučujícímu. 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 uchovávající 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 větve vytvořen požadavek o zařazení. Po kontrole vedoucím byl požadavek zařazen nebo vrácen k opravě.

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

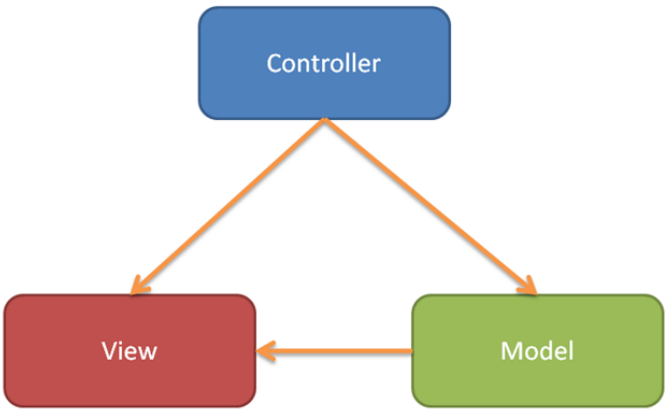
Pro vývoj se využil princip průběžné integrace. Každá verze byla zkompilována, otestována a zanalyzována na vzdáleném serveru. Tyto činnosti zajišťovaly systém Jenkins[22] a Gitlab. Jenkins aplikaci zkompiloval, spustil testy a statickou analýzu kódu zajištěnou systémem SonarQube[23]. Výsledky poté publikoval 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 repositáře. To zajišťuje jednotné verze knihovna 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 repositář. [27]. Na něm jsou uložené poté všechny knihovny, která jsou v rámci toho projektu použity.

V rámci sestavení lze pustit testy, včetně přídavných doplňků. Projekt používá doplněk Cobertura[28], 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 jsou poslány 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 řetězcemi 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. Doplňky 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.

Kapitola 4

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 částí, která byla zmíněna výše. Na žádost jiného týmu jsme také vytvořili REST rozhraní, poskytující získané ceny pro daný produktu.

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 informacích 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 chyba.

4.3.1 Vytváření požadavků pro ProductProvider

Následující nedostatky jsou úzce spjaty s tím, jak aplikace plánovala práci. Plánováním práce je myšleno proces ve kterém jsou vybrány požadováné 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 předefinovaném intervalu poté zajistilo, že požadavky byly vytvořeny pro všechny požadované adresy.

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 nalezlo 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 označená, že jsou pravděpodobně chybná. Dále probíhala kontrola získané ceny s a bez DPH oproti cenám získané na konkrétní stránce dříve. Kontrola zde porovnala 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 uložené informace o chybě. Tento způsob řízení programu však způsoboval, že celá validace skončila při první chybě.

Informace byly následné 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.

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. Myšlenka byla taková, že pro vytvoření samotné šablony, je nutné nejdříve stránky stáhnout, nechat vyhodit chybu parsování a nálsdně 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, jejíchž 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 staženy. 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 podotknou, 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 přepsaní 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 částek 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 analyzátorem. Z tohoto důvodu jakákoliv oprava nebo implementace nových požadavků mohla narušit stávající funkcionalitu bez možnosti rychlého ověření. Tím by mohla být jakákoliv změna velmi časově náročná 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 týmového projektu

Kapitola 5

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 požívá 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 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 testovaní. 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.

Dále pro větší přehlednost by bylo vhodné přesunou 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 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.

Pro korektní opravu, která bude odpovídat čistého objektového návrhu, je nutné nejprve předělat celý proces vytváření požadavků. Jelikož důvodů pro vytvoření požadavků z adresy může být více, je žádoucí chovat se při vytváření požadavků k adresám dle typů..

Výsledek hledání požadovaných adres, by měl mít striktně oddělené množiny adres dle toho jak byly nalezeny. Na základně typů poté požadavky vytvářet a doplnit o požadované atributy.

5.4 Plánování práce

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

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

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

5.5 Spojení chyb analyzátoru

V případě detekce možné chyby při analyzování získaných požadavků jsou vytvářeny chyby, které jsou určeny pro vyřešení administrátorem. Zde je nutné požadováno chyby spojit do jedné, tak aby administrátor mohl vyřešit všechny možné chyby analyzátoru najednou. Po úpravě samotného DataProvideru, tak aby nechal proběhnout všechny kontroly a neskončil při první chybě, je výsledek uložen do databáze Managerem. Úprava se týká i webového rozhraní, které tuto možnost neposkytuje.

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 srovnávačích a na nich pomocí vyhledávaní 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 existující 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 přesně shody některého z identifikátoru, 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 uchovávaní 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.

Kapitola 6

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 a zmenšení počtu parametrů metod.

6.1.1 Servisní třídy

Pro větší přehlednost byly všechny servisní třídy do nadřazeném balíčku „service“. Servisní třídou jsou takové, které nespadají 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 byl program řízen pomocí výjimek, které představovaly 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, což jsou data validní odpovědi, či informace o zjištěné chybě, například v získaných datech.

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

```
/**
 * 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 DPParserResponse {

    /**
     * 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);
```

```
33      }
34
35      public void setName(String name) {
36          this.name = name;
37      }
38  }
```

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 uchovávající data a přístup k ním 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 [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 přístup k hodnotě se proto musí nejdříve programátor objektu zeptat, zda obsahuje hodnotu. Mechanismus nutného ověření hodnoty, pak zamezuje nežádoucím výjimkám, především *NullPointerException* [40].

Výjimky jsou tak 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 obsahuje změna rozhraní, kdy bylo potřeba zmenšit počet parametrů a odstranit výjimku, která byla vyhozena v případě nalezení chyby.

Třetí změna je poté samotné pouštění validací. V původním řešení byla třída závislá na všech příslušných validací, které spouštěla a navíc skončila při první chybě, což je jedna z příčin chování, které jsem popsal v kapitole poukazující na možnosti spojit chyby analyzátoru section 5.5

Vytvořil jsem nový návrh, který je pak použit i na ostatních místech interní části v závislosti na prováděných vylepšení. Servisní třídě jsem odebral jednotlivé závislosti na 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 změněn návratový typ na *Optional* případné chyby nebo prázdnou hodnotu *Optional*. Implementace těchto rozhraní pak zahrnovala kontrolu jednotlivých hodnot. Při úpravě validací, aby odpovídaly novému rozhraní jsem 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 skupin pak obsahují třídy *AbstractStringAnalysis* a *AbstractPriceAnalysis*. Výsledná základní validace kontrolující, zda získaná hodnota odpovídá nějaké historické hodnotě vypadá následovně:

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

```
38      @Override
39      AnalysisErrorMessage generateAnalysisErrorMessage(String
            comProductValue, String parsedValue) {
40          return new AnalysisErrorMessage()
41                  .withErrorMessage(
42                          String.format("Parsed name value[%s] doesn't
                                match known name value[%s]",
43                                  parsedValue, comProductValue)
44                  )
45                  .withErrorType(AnalysisErrorType.DIFFERENT_NAME);
46      }
47  }
```

Konečné spuštění validací bylo ve výsledku zkráceno na metodu obsahující jeden řádek kódu, ačkoliv tento řádek obsahuje více zřetězených 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
             AnalyserException when analysing fails, contains error type
8        * @param analyserFlags
9        */
10      @Override
11      public void analyse(ComProduct newInfo,
12                          ComProductHistory newData,
13                          ComProduct oldInfo,
14                          List<ComProductHistory> productHistory,
15                          ComAnalyserFlags analyserFlags) throws
                                AnalyserException {
16          ValidatorData data = new ValidatorData(newInfo, newData,
                oldInfo, productHistory, analyserFlags);
17          try {
18              eanValidator.validate(data);
19              partNumberValidator.validate(data);
20              priceValidator.validate(data);
21              namesValidator.validate(data);
22          } catch (AnalyserException e) {
23              logger.info("Analysis failed for product id: {}",
                    newInfo.getProductId(), e);
24              throw e;
25          }
26          if (!data.getWarnings().isEmpty()) {
27              //No handling neeeded at the moment
```

```
28          }
29      }
```

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

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

6.2 Oprava komunikace Manager - ProductProvider

6.3 Plánování práce

Implementoval jsem 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.

6.4 Spojení chyb analyzátoru

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. Službu je však možné rozšířit o velký počet doplňků. Pomocí těch je pak možné sledovat například stav buildu 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. Myšlenkou rozdělení bylo oddělení části, která vyhledává na internetovém obchodu a části, která samotné obchody, které by mohly produkt nabízet hledá.

Implementována byla pouze první část, jelikož hledání samotných obchodů lze nahradit manuálním přidání obchodů na kterých chceme vyhledávat, případně využít některý se seznamů internetových obchodů v České republice a ty manuálně vložit.

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 nutné kritérium existence pouze částečné šablony, která obsahuje informace jak na obchodě vyhledávat. Tento požadavek poté vytvoří chybu pro administrátora, aby specifikoval jak na stránce vyhledávat detaily adres. Webové rozhraní pro tento proces, bylo vytvořeno již v rámci týmového projektu a lze tedy použít.

6.7 Párování produktu

Byl navržen proce, 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.

Kapitola 7

Zhodnocení provedených vylepšení

7.1 Více identifikátorů

//TODO - popuze příprava

7.2 Pokrytí testy

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

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

Obrázek 7.1: Pokrytí testy vytvořeného řešení týmového projektu

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.: *Refaktoring: 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/`

Příloha **A**

Seznam použitých zkratek

EAN European Article Number

XML Extensible markup language

HTML Hypertext Markup Language

CSS Cascading style sheets

JSON JavaScript Object Notation

HTTP Hypertext Transfer Protocol

DAO Data Access Object

URL Uniform Resource Locator

Příloha **B**

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 LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS