VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Petr Dihel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Dihel**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Shopsys s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Janoušek**

Konzultant bakalářské práce: Jan Kroček

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA *

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 18. dubna 2019 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 18. dubna 2019

Rád bych na tomto místě poděkoval všem kolegům společnosti Shopsys s.r.o., kteří mi pomohli při absolvování této odborné praxe.

Abstrakt

Tato bakalářská práce popisuje průběh odborné praxe, kterou jsem absolvoval ve firmě Shopsys s.r.o. Zaměřuji se na popis úkolů, které mi byly přiřazeny konzultantem a zejména na postup při jejich řešení. Mým primárním úkolem, který je zároveň hlavním tématem této bakalářské práce, bylo zpracování interní aplikace nazvané *Všeaplikace* a její součásti, agregátoru služeb.

Klíčová slova: Agregátor služeb, Všeaplikace, PHP, Odborná praxe

Abstract

This bachelor thesis describes the course of professional practice, which I completed at Shopsys s.r.o. I focus on describing the tasks assigned to me by the consultant and, in particular, on how to deal with them. My primary task, which is also the main topic of this bachelor thesis, was the processing of an internal application called *Všeaplikace* and its component, a service aggregator.

Key Words: Service aggregator, PHP, Professional Practice

Obsah

Seznam použitých zkratek a symbolů

PHP	–	Hypertext Preprocessor
HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
LESS	–	Leaner Style Sheets, preprocessor style sheet language
XML	–	eXtensible Markup Language
JSON	–	JavaScript Object Notation
CSV	–	comma-separated values
CRUD	–	Create, read, update and delete
API	–	Application programming interface
URL	–	Uniform Resource Locator
AJAX	–	Asynchronous JavaScript and XML
RDSMS	–	relational data stream management system
RDBMS	–	relational database management system
DOM	–	Document Object Model

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Praxi ve firmě Shopsys jsem si vybral, protože jsem už měl nějaké zkušenosti s vývojem webových aplikací se stejnými technologiemi, které tato firma používá. Také jsem chtěl nabrat zkušenosti pracovního postupu v týmu na větších projektech. Ve firmě jsem pracoval na více projektech. V tomto dokumentu popisuji práci na interní aplikaci tzv. *Všeaplikaci* a její součásti, agregátoru služeb. Nejdříve popíšu služby typu porovnávačů zboží. Následně popíšu architekturu aplikace a proč je tato interní aplikace vůbec potřeba. Poté sepíšu úkoly, které mi byly přiřazeny a postupy řešení těchto úkolů. V poslední části budu probírat jaké nové znalosti jsem se naučil, jaké znalosti mi chyběly při řešení úkolu a celkové zhodnocení mé praxe.

2 Popis firmy a zařazení studenta

2.1 O firmě

Shopsys s.r.o. je firma, která se soustředí na technologický vývoj velkých e-shopů na míru. Shopsys založil v roce 2003 Petr Svoboda ve svých 16 letech. Ve firmě Shopsys již vzniklo více než 800 e-shopů. Jejím momentálním hlavním cílem je produkt *Shopsys Framework*. Jde o *open source* platformu pro stavění velkých e-commerce řešení.

2.2 Pracovní zařazení studenta

Ve firmě jsem pracoval jako programátor, který se zaměřoval hlavně na *backend* webových stránek. Tedy ne o vzhled stránek, ale funkčnost, zobrazování dat a jejich zpracování.

2.3 Poskytované služby

Firma nabízí e-commerce řešení na míru pro velké e-shopy. Také nabízí své *open source* řešení pro elektronické obchodování pro podniky, které mají svůj vlastní vývojářský tým.

3 Použité technologie

Při odborné praxi jsem pracoval hlavně s programovacím skriptovacím jazykem PHP.

3.1 PHP

Populární skriptovací jazyk pro vývoj webových aplikací.

3.2 Javascript, Jquery

3.2.1 Javascript

Javascript je skriptovací jazyk, používaný na straně klienta, lze ho však také vidět i na straně serveru. V této praxi se používal hlavně pro interakci s uživateli a grafickými prvky na stránce.

3.2.2 Jquery

Jquery je Javascriptová knihovna, která slouží pro snadnější manipulaci s HTML DOM. Lze jednoduše vybrat požadované prvky z dokumentu a upravovat jejich vlastnosti. Dále lze snadněji vytvářet animace, zachycovat události, používat AJAX. [1]

3.3 HTML

Značkovací jazyk sloužící pro vytváření webových stránek a webových aplikací. Používá se s kombinací CSS a Javascriptu.

3.4 CSS

Jazyk, ve kterém se popisuje, jak se má zobrazovat dokument, který je napsaný ve značkovacím jazyce.

3.5 LESS

Je preprocesor pro CSS. Rozšiřuje CSS dynamickým chováním, jako například proměnnými, operacemi a funkcemi. Používá se pro jednodušší stylování než jen s použitím CSS.

3.6 SQL, MariaDB

3.6.1 SQL

Je dotazovací jazyk pro správu dat uložených v RDSMS nebo v RDBMS.

3.6.2 MariaDB

MariaDb je relační databáze vyvíjená komunitou. Vznikla z MySQL.

3.7 GIT

GIT je verzovací systém. Usnadňuje práci v týmu na jednom projektu, kde se pracuje na více úpravách najednou.

4 Porovnávače zboží

Porovnávače zboží slouží jejich uživatelům ke srovnání nabídek zboží od různých e-shopů. Mohou zde vidět a porovnávat například cenu zboží, dopravy a služby k produktu, které e-shop nabízí. E-shopům zase nabízí zviditelnění, a tím tedy reklamu a možné vyšší prodeje. Pro zviditelnění vybraných produktů v porovnávači je nutné k nim dodat informace. Ve většině případů se jedná o datový soubor, takzvaný XML feed.

XML feed je datový soubor ve formátu XML. Tento soubor by měl obsahovat nabídku produktů e-shopu, které chce aby byly na porovnávači viditelné. U některých případů lze poslat i ty, které nemají být viditelné. Měl by obsahovat důležité informace o produktech, jako je cena produktu, jeho dostupnost, název a další. Některé informace u produktu mohou být povinné.

Mezi porovnávače například patří:

- *zboží.cz*
- *heuréka.cz*
- *heuréka.sk*
- *google*

4.1 XML soubor

XML je značkovací jazyk, který slouží hlavně k uložení a přenosu dat. Tento jazyk je podobný značkovacímu jazyku HTML. Na rozdíl od HTML však nemá přesně definované značky, které může používat. Tento jazyk byl navržen tak, že je i člověkem čitelné, jaké informace jsou v něm uloženy. Značky mohou mít nějaké atributy, ty však musí být správně ohraničeny znakem ″. [2]

4.2 XML feed *zboží.cz*

V příkladu XML feedu je zobrazen feed pro *zboží.cz*. Nejdříve vše obaluje značka *SHOP*. V ní jsou zanořeny značky *SHOPITEM*, které označují produkty. Značka *SHOPITEM* obsahuje dále například značku *ITEM_ID*, která obsahuje unikátní identifikátor v rámci nabídky e-shopu. Díky této značce lze zboží identifikovat od sebe a lze sledovat jeho historii v této službě. Značka *PRODUCTNAME* musí být v následujícím formátu. Musí nejdříve obsahovat výrobce produktu. Dále musí obsahovat produktovou řadu produktu. Poté musí obsahovat produktové označení a nakonec variantu produktu. Značka *CATEGORYTEXT* musí obsahovat kategorii produktu, kterou poskytuje služba *zboží.cz*, nikoli však kategorii, ve které je zařazena v e-shopu. [3]

```
<?xml version="1.0" encoding="utf-8"?>
 <SHOP xmlns="http://www.zbozi.cz/ns/offer/1.0">
 <SHOPITEM>
 <ITEM_ID>62448</ITEM_ID>
```

```
<PRODUCTNAME>Solartent MC234CZ/A premium Beige</PRODUCTNAME>
<PRODUCT>Stínítko z laminátových prutů Solartent MC234CZ/A premium Berige </
    PRODUCT>
<DESCRIPTION>Velmi praktické stínítko s lehkou konstrukcí z laminátových prutů
    .</DESCRIPTION>
<CATEGORYTEXT>Dům, byt a zahrada | Zahrada | Stínící technika | Zahradní slune
    čníky</CATEGORYTEXT>
<EAN>8594061743744</EAN>
<PRODUCTNO>MC234CZ/A</PRODUCTNO>
<MANUFACTURER>Solartent</MANUFACTURER>
<URL>http://example.com/slunecniky/solartent123</URL>
<DELIVERY_DATE>0</DELIVERY_DATE>
<DELIVERY>
<DELIVERY_ID>DPD_PICKUP</DELIVERY_ID>
<DELIVERY_PRICE>100</DELIVERY_PRICE>
<DELIVERY_PRICE_COD>149</DELIVERY_PRICE_COD>
</DELIVERY>
<EXTRA_MESSAGE>free_gift</EXTRA_MESSAGE>
<FREE_GIFT_TEXT>Powerbanka ADATA s kapacitou 12500 mAh</FREE_GIFT_TEXT>
<EXTRA_MESSAGE>extended_warranty</EXTRA_MESSAGE>
<PARAM>
<PARAM_NAME>barva</PARAM_NAME>
<VAL>Béžová</VAL>
</PARAM>
<IMGURL>http://example.com/obrazky/slunecniky/solartent123.jpg</IMGURL>
<PRICE_VAT>1290</PRICE_VAT>
<MAX_CPC>6,50</MAX_CPC>
<MAX_CPC_SEARCH>5,80</MAX_CPC_SEARCH>
</SHOPITEM>
</SHOP>
```

Výpis 1: Příklad xml feedu

4.3 Feed kategorizace

Porovnávače zařazují produkty do kategorií. E-shop ale může mít produkty do kategorií jinak zařazené, nebo mít jiné kategorie. Některé porovnávače dokážou produkt správně spárovat jen podle názvu produktu. V takovém případě většinou musí mít název produktu přesně požadovaný formát. Tedy nastává problém správného spárování se správným zbožím na straně porovnávače. Dokud není zboží správně spárováno, e-shop je penalizován, a to například tím, že se produkt

nezobrazí ve vyhledávači nebo v dané kategorii. Proto některé vyhledávače poskytují datový soubor, který obsahuje jejich vlastní stromovou strukturu kategorií. Správci e-shopu mají tedy možnost své kategorie spárovat s kategorií od porovnávače. Zvolenou celou cestu kategorie je možné dodat v datovém souboru s produkty. Datové soubory s feed kategorizací jsou většinou ve formátech XML, JSON, CSV.

Kategorie může určovat cenu za proklik nebo například formát názvu. Proto je důležité, aby produkty byly správně zařazeny.

4.4 JSON

JSON neboli Javascriptový objektový zápis je způsob zápisu dat. Slouží hlavně k přenosu zapsaných dat. Vstupem může být jakékoliv pole či objekt a složitost hierarchie není nějak limitována. Nevýhoda je, že u něho není možnost definovat znakovou sadu. Výstupem je vždy řetězec.

4.5 Kategorizace *zboží.cz*

V příkladu feedu kategorizace od *zboží.cz* je ve formátu JSON. Z feedu vyplývá, že jedna z hlavních kategorií má jméno *Foto* a má podkategorii *Foto doplňky a příslušenství*. Ta má zase další podkategorie například kategorii *Blesk*, která má atribut *categoryText*, což je celá cesta k té dané kategorii. V tomto případě *Foto | Fotodoplňky a příslušenství | Blesky*. Také má atribut *id* 10.

Některé kategorie nemají *id* a *categoryText*. K těmto kategoriím nelze přiřazovat produkty.

```
{
 "name": "Foto", "children":
    [
       {
          "name": "Foto dopl\u0148ky a p\u0159\u00edslu\u0161enstv\u00ed",
          "children": [{
             "categoryText": "Foto | Foto dopl\u0148ky a p\u0159\u00edslu\
                u0161enstv\u00ed | Blesky",
             "name": "Blesky",
             "id": 10
          }, {
             "categoryText": "Foto | Foto dopl\u0148ky a p\u0159\u00edslu\
                u0161enstv\u00ed | Objektivy",
             "name": "Objektivy",
             "id": 11
          }, {
```

```
    "categoryText": "Foto | Foto dopl\u0148ky a p\u0159\u00edslu\
       u0161enstv\u00ed | P\u0159eds\u00e1dky, filtry a krytky",
    "name": "P\u0159eds\u00e1dky, filtry a krytky",
    "id": 12
}...
```


Výpis 2: Příklad datového souboru pro feed kategorizaci od služby *zboží.cz*

5 *Všeaplikace* - interní aplikace a agregátor služeb

Cílem je vytvořit modulární agregátor služeb, který bude obsahovat různé funkce a nástroje, které ve svém důsledku usnadní práci zaměstnancům společnosti Shopsys. Problémem je, že stejná data služeb se zpracovávají na každém e-shopu zvlášť, tím se zbytečně zvedá zatížení serverů. Dalším problémem je, že když služba změní strukturu dat, nebo úplně způsob napojení, je potřeba tuto změnu provést na všech e-shopech, které službu využívají.

Primárním účelem této aplikace je sjednotit všechny úkony na jednom místě. Na jednotlivé e-shopy již tedy půjdou data v jednotném formátu. *Všeaplikace* bude obsahovat *diff tool* pro webové služby, nástroj pro načtení kategorizací různých služeb například od *google*, *seznam*, *heureka.cz* a *heureka.sk*. Kategorizace bude také multijazyková. Součástí aplikace bude správa seznamů povolených e-shopů pro synchronizaci těchto kategorií, včetně různých verzí exportů.

Dále bude obsahovat zpracování dat typů doprav pro Heuréku a napojení dat na zásilkovny a uloženky. Jeden z požadavků bylo implementovat jádro aplikace tak, aby bylo snadno rozšiřitelné o případné moduly a funkce.

5.1 Architektura *Všeaplikace*

Architektura *Všeaplikace* je založená na stabilní verzi e-shopu, která byla určená pro nižší verzi PHP. Tato verze byla upravena tak, aby byla spustitelná na verzi PHP 7.2. a nevyužívala takzvané *deprecated* funkce, jako tomu bylo u původní verze. Aplikace bude funkční na více doménách a bude se zobrazovat ve více jazycích. Na každé doméně budou jiné funkce a nástroje.

Na projektu se využívá objektově relační mapování. Jsou použity návrhové vzory *Dependency injection*, *Data mapper pattern*, *Repository pattern*, *Factory pattern* a *Model View Controller pattern*. *Všeaplikace* využívá návrhového vzoru *Model-View-Controller*. Tato architektura rozděluje aplikaci do třech nezávislých částí. Datového modelu aplikace, uživatelského rozhraní a řídící logiku. [4]

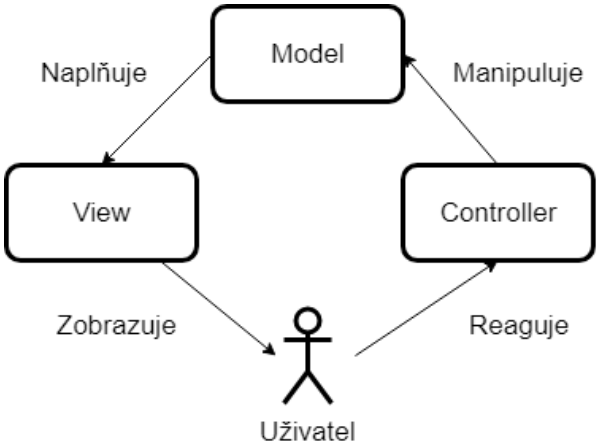
Model Tato vrstva obsahuje logiku, pravidla a spravuje data aplikace. Na *Všeaplikaci* je tato vrstva reprezentována například soubory ve složce *domain_agregator/model*.

Controller Reaguje na vstupy od uživatele a převádí je do modelu, nebo do *View* vrstvy. Reprezentováno na projektu jako soubory ve složce *domain_agregator/pages*.

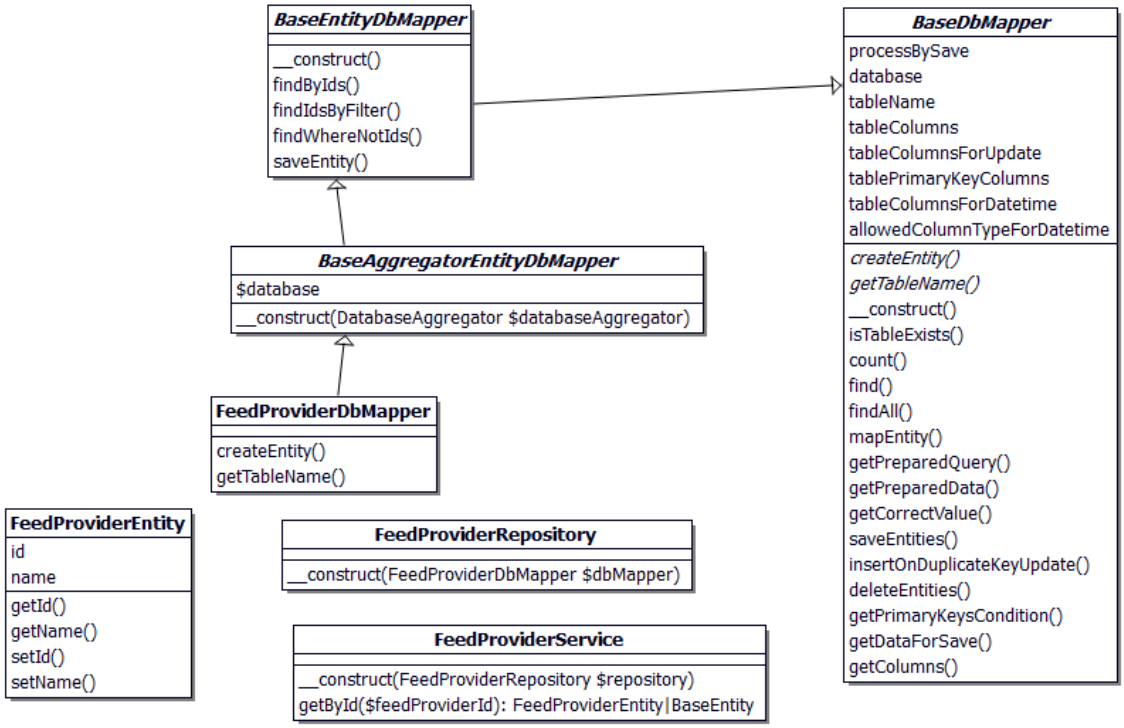
View Slouží pro prezentaci dat uživateli. Na projektu můžeme vidět tuto vrstvu například v *domain_agregator/templates*.

Přístup a práce s entitami je rozdělena do následujících vrstev

- *Entita*
- *Service*
- *Repository*
- *Mapper*



Obrázek 1: Model - View - Controller



Obrázek 2: Model *Feed provider*

Tabulka 1: Tabulka *feed provider value translate*

Název sloupečku	Datový typ	Popis
provider_value_id	int	unikátní identifikátor poskytovatele služby kategorizace
language_id	int	unikátní identifikátor jazyka
name	varchar	název kategorie
full_path	varchar	celá cesta ke kategorii
tis_visible	tinyint	je kategorie viditelná, 0 - ne 1 ano
last_update	int	timestamp poslední aktualizace záznamu

5.2 Návrhový vzor *Dependency Injection*

Na projektu se tento vzor používá k využití vkládání závislostí. Instance objektu se nestará o závislosti, které potřebuje, ale stará se o ně takzvaný *Dependency injection kontejner*. Kontejner je prvotní továrna, která poskytuje všechny potřebné služby pro chod aplikace. Služby jsou v tomto případě všechny instance objektů. Aplikace využívá injekci konstruktorem, kdy při zrodu objektu jsou závislosti předány pomocí parametrů konstruktoru. V příkladu se tento návrhový vzor používá pro předání správné třídy *FeedProviderDbMapper* jako parametr konstruktoru ve třídě *FeedProviderRepository*.

Entity jsou třídy, které reprezentují nějaký objekt doménové logiky. Třída má jako atributy vlastnosti těchto objektů. Například unikátní identifikátor, nebo název. Tyto třídy slouží pouze k dočasnému uložení těchto informací a dále se s nimi pracuje v dalších vrstvách.

V tomto případě třída *FeedProviderEntity* reprezentuje službu, která poskytuje feed kategorizaci. Tedy reálně zastupují například služby *zboží.cz*, *heureka.cz* a další poskytovatele služeb.

Service třídy slouží pro práci s entitami. V názorném příkladě ji zastupuje třída *FeedProviderService*.

Repository vrstva pracuje s třídami, které využívají perzistentní úložiště. V příkladě jako *FeedProviderRepository*.

5.3 Návrhový vzor *Data Mapper*

Jedná se o návrhový vzor poskytující datovou vrstvu, který přenáší data mezi perzistentním úložištěm a dočasným úložištěm v doménové vrstvě. Tedy doménový objekt neobsahuje CRUD operace. V příkladu lze vidět objekt *FeedProviderEntity*, který metody pro tyto operace nemá. *FeedProviderDbMapper* je třída, která slouží pro práci s databází, ale pouze pro entitu *FeedProviderEntity*. Místo toho je pokrývá metoda *saveEntity()* ve třídě *FeedProviderDbMapper*.

Po domluvě s kolegy se na projektu používá přesné znázornění třídy v implementaci se zobrazením do databáze. Tedy všechny atributy třídy entity jsou uloženy v databázi.

Při porovnání tabulky *Feed provider value translate* s příkladem kódu z implementace lze vidět, že všechny atributy tabulky jsou zobrazeny i ve třídě, která tabulku představuje.

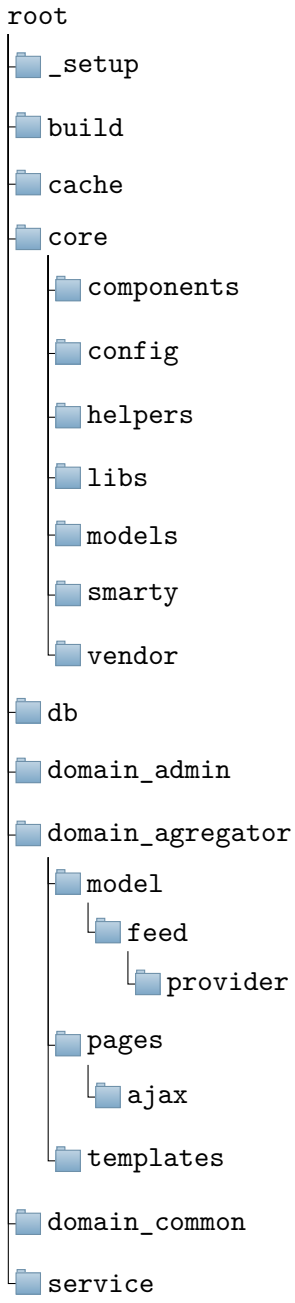
```php
class FeedProviderValueTranslateEntity extends BaseEntity {

   use TLanguageId;
   use TName;
   use TLastUpdate;

   /**
    * @var int
    */
   private $providerValueId = 0;

   /**
    * @var string
    */
   private $fullPath = '';

   /**
    * @var bool
    */
   private $isVisible = false;

   /**
    * @return int
    */
   public function getProviderValueId() {
      return $this->providerValueId;
   }

   /**
    * @param int $providerValueId
    * @return $this
    */
   public function setProviderValueId($providerValueId) {
      $this->providerValueId = (int)$providerValueId;
      return $this;
   }

   /**
```

```
     * @return string
     */
    public function getFullPath() {
        return $this->fullPath;
    }

    /**
     * @param string $fullPath
     * @return $this
     */
    public function setFullPath($fullPath) {
        $this->fullPath = (string)$fullPath;
        return $this;
    }

    /**
     * @return bool
     */
    public function getIsVisible() {
        return $this->isVisible;
    }

    /**
     * @param bool $isVisible
     * @return $this
     */
    public function setIsVisible($isVisible) {
        $this->isVisible = (bool)$isVisible;
        return $this;
    }

}
```

Výpis 3: FeedProviderValueTranslateEntity

5.4 Adresářová struktura aplikace

5.4.1 Úryvek stromu struktury aplikace



5.4.2 Složka *setup*

Ve složce setup se nachází soubory s logikou pro inicializování projektu, vytvoření struktury databáze a její naplnění daty, instalace nových modulů, nastavení tzv. *master e-mailu*. Na tento e-mail se posílají všechny odchozí e-maily. Tato funkce se používá v testovacím a vývojovém prostředí aplikace. Do složky *setup* není v produkčním režimu přístup.

Zde jsem například přidal možnost přiřadit k uživatelům roli *superadmina*. Dále jsem zde přidal model generátor. Tomu lze zadat následující parametry.

- Doména
- Název modelu
- Třída, kterou má model rozšiřovat

Tento skript poté vygeneruje adresářovou strukturu podle názvu modelu a vytvoří 4 soubory (*Mapper*, *Repository*, *Service*, *Entity*).

5.4.3 Složka *build*

Obsahuje vygenerované výsledné zprávy z automatických kontrol kódu.

5.4.4 Složka *cache*

Ve složce cache se ukládají dočasné zkompilované šablony pro rychlejší načtení stránek.

5.4.5 Složka *core*

Ve složce core se nachází komponenty, modely, pomocné třídy, vendor, konfigurační soubory a jádro aplikace.

5.4.6 Složky *domain_*

Pro každou doménu je zvlášť složka *domain_* a název domény. Ty obsahují šablony pro jednotlivé stránky (Views), php soubory, které obstarávají akce uživatele (*Controllery*) a modely, které se na dané doméně používají.

Ke složkám domény je ještě jedna složka *domain_common*. V této složce se nachází modely, které se používají na více doménách.

Aplikace si vytváří seznam šablon. Ze složky *domain_common* se do seznamu načtou všechny šablony. Poté se podle domény, na které se uživatel aplikace zrovna nachází, přepíšou ty šablony, které existují ve složce s touto doménou.

5.4.7 Složka model

Obsahuje modely, které se používají na této doméně aplikace. Tedy například následující třídy.

- *FeedProviderEntity*
- *ServiceFeedProviderRepository*
- *FeedProviderMapper*

- *FeedProviderService*

Třídy jsou rozděleny do složek podle svého názvu. Takže například třída *FeedProviderEntity* je zanořena ve složkách *Feed* a *Provider*.

5.4.8 Složka *service*

Obsahuje skripty, které jsou spouštěny službou *CRON* v předem definovaných intervalech. Například aktualizaci dat z feed kategorií.

6 Přiřazené úkoly

6.1 Přihlašování pomocí *Google Sign-in*

6.1.1 Úkol

Měl jsem za úkol implementovat přihlášení pomocí *Google Sign-In*.

6.1.2 Odhad úpravy

Odhad doby práce na tomto úkolu se pohyboval okolo 15 hodin.

6.1.3 Postup řešení

Přihlašování přes *google* jsme vybrali, protože měl výhodu jednotného přihlašování do více aplikací. Další výhodou je, že při zablokování tohoto účtu se zamezí uživateli přístup do všech aplikací, ve kterých se používá, a také je již vyřešené zabezpečené přihlašování.

Před započetím implementace jsem musel založit projekt v *Google API Console*, a získat tak svoje *client ID*. Základní, nejjednodušší implementace podle dokumentace google si tlačítko pro přihlášení zobrazuje a generuje samotné API od *google*. Je potřeba mít HTML element s třídou *g-signin2*, ke kterému se přivážou akce pro přihlášení. Také je v této verzi potřeba přidat *Client id* do hlavičky stránky pomocí *meta tagu*. [5]

Zvolil jsem implementaci takovou, která mi dovolila připojit funkčnost *Sign-in* na vlastní HTML element. Dostal jsem tak větší kontrolu i nad tím, kdy se skript pro přihlášení volá, a jestli má být uživatel vždy vybídnut vybrat účet, ze kterého se chce přihlásit.

Uživatelské přihlašování a autentizace přes *google* API probíhá pouze na jedné stránce. Jakmile se uživatel přihlásí ke svému google účtu, tak google API vrátí informace o uživateli včetně jeho vygenerovaného tokenu. Poté, kdy skript obdrží tyto informace, zavolá se na pozadí pomocí AJAX php skript, který zase ověří správnost tokenu tak, že se zavolá zpátky *google* API s tím daným tokenem a ten zpátky vrátí informace o uživateli, kterému tento token patří.

Druhá varianta řešení by bylo použít externí knihovny pro ověření tokenu. Zde byl problém, že knihovna pro toto ověření byla rozsáhlá. Také se ověření tokenu vráceného z *google* API nepoužívá na každé stránce. Proto jsme se rozhodli pro první popsanou variantu.

Pokud je vše v pořádku a e-mail je z domény *Shopsys*, tak si vygeneruji svůj vlastní token, který se skládá z *google* tokenu, user agentu, *SessionId* a e-mailu, který se uloží do databáze. Token se uloží uživateli do jeho Session a je přihlášený. Poté se na každé stránce místo *google* API kontroluje vygenerovaný token v databázi s informací v Session uživatele. Takhle pro každou doménu aplikace existuje zvlášť záznam přihlášení. A také zvlášť pro každý *SessionId*. Na aplikaci lze pracovat i offline, což byl jeden z požadavků.

```
/**
```

```
 * @param {element} element
 */
function attachSignin(element) {
   auth2.attachClickHandler(element, {},
      function(googleUser) {
         let id_token = googleUser.getAuthResponse().id_token;
         googleUserData = googleUser;
         if (id_token !== "") {
            window.ajaxVerifyLoginRequest = $.ajax({
               type: 'POST',
               url: shop_url + 'ajax.php?page=verify_login',
               data: {id_token : id_token},
               beforeSend : function() {
                  if (window.ajaxVerifyLoginRequest != null) {
                     window.ajaxVerifyLoginRequest.abort();
                  }
               },
               success: function(data) {
                  handleSigInSuccess(data);
               },
               error: function(error) {
                  appendSigInError(JSON.stringify(error, undefined, 2));
               }
            });
         }

      }, function(error) {
         appendSigInError(JSON.stringify(error, undefined, 2));
      });
}
```

Výpis 4: Úryvek kódu který se volá po přijmutí odpovědi z API

Token má nastavenou dobu expirace, do jaké doby se uživatel musí znovu přihlásit přes *google* API. Každým požadavkem na server na dané doméně se doba expirace prodlouží. Tedy uživatel se odhlásí až po nějaké době jeho neaktivity.

Úryvek kódu znázorňuje akci, která se provede poté, kdy *google* API pro přihlášení vrátí zpět odpověď.

S tímto úkolem tedy souvisí vytvoření modelu uživatelů. Konkrétně se jednalo o třídy *UserEntity*, *UserRepository*, *UserMapper*, *UserService*. Tenhle model jsem přidal do adresářové

Tabulka 2: Tabulka *User Email*

Název sloupečku	Datový typ	Popis
user_id	int	unikátní identifikátor uživatele z tabulky user
email	string	e-mail uživatele
is_default_for_login	tinyint	je to přihlašovací e-mail? - nabírá hodnoty 1 jinak 0
is_contact_email	tinyint	je to kontaktní e-mail? - nabírá hodnoty 1 jinak 0

struktury *domain_common* kvůli tomu, že se používá na více doménách.

Poté se objevil požadavek na to, aby mohl mít uživatel více e-mailů. Jen jeden z nich slouží k přihlašování. Vytvořil jsem tedy další model, a to *UserEmail*. Následně bylo potřeba vytvořit model *UserLoginActivity* pro záznamy přihlášení uživatelů.

V rámci této úpravy jsem také napsal skript pro vytváření modelů v *_setup* složce, který vytvořil podle parametrů odpovídající adresářovou strukturu a třídy s konstruktory s odpovídajícími parametry. Například třídu *UserRepository*.

6.1.4 Zhodnocení

Povedlo se mi implementovat *google* API pro přihlašování a naučit se ho využívat. Také jsem vytvořil model *User*, který se využije i v dalších úkolech přes celou interní aplikaci. S tímto úkolem bylo potřeba i poupravit jádro aplikace a upravit kód po jeho kontrole od kolegy. Z těchto důvodu se implementace úpravy prodloužila. Zabrala asi 25 hodin práce.

6.2 Uživatelské role

6.2.1 Úkol

Mým dalším úkolem bylo vyřešit uživatelské role.

6.2.2 Odhad úpravy

Odhad doby práce na tomto úkolu se pohyboval okolo 15 hodin.

6.2.3 Postup

Jednalo se o uživatelské role. Tyto role určují, jaký uživatel má přístup na jakou doménu. Na aplikace jsou následující role.

1. Nepřihlášený uživatel aplikace nebo uživatel, který nemá na dané doméně žádnou roli je pouze vybídnut k přihlášení na jiný účet, který toto právo má.

2. Uživatel má přístup na tuto doménu.

3. Administrátor má všechna práva jako uživatel na dané doméně a navíc přístup do některých kategorií.

4. Vývojář má všechna práva na dané doméně, ale zobrazí se mu zde navíc nástroje pro ladění aplikace.

5. Na aplikaci je také skrytá role, speciální role *superadmin*. Když má uživatel tuto roli, dostane automaticky všechny práva na všechny domény. Tato role přednostně slouží k usnadnění práce programátorům, kteří si spouští aplikaci na svém zařízení.

Před zobrazením jakékoliv informace, se nejdříve zkontroluje, zda je uživatel správně přihlášen. Tedy porovnají se jeho informace uložené v Session s tím, co je v databázi. Pokud je to provedeno v pořádku, zkontroluje se, zda má na té určité doméně nějakou z uživatelských rolí. Když tomu tak není, zadá se mu role *nepřihlášený uživatel*. Když má roli *developer*, tak se mu zobrazí i nástroje pro ladění aplikace.

U této úpravy jsem tedy vytvořil modely *UserRole* a *UserRoleType*. S ohledem na uživatelskou roli *superadmin* jsem vytvořil ve složce *_setup* formulář, kde lze vybrat jakéhokoliv uživatele a přiřadit mu roli *superadmin* pro zjednodušení inicializování projektu a práci na projektu pro vývojáře.

6.2.4 Zhodnocení

Implementace zabrala více než 20 hodin práce, a to z důvodu úpravy architektury jádra pro snazší další vývoj.

6.3 Přidání formulářů pro upravování Uživatelů

6.3.1 Úkol

Následující úkol byl zobrazit seznam uživatelů a vytvořit formulář pro jejich upravení. Na to navazovaly další formuláře, například formulář pro vytváření a upravování pracovních pozic.

6.3.2 Odhad úpravy

Úprava měla odhadem zabrat 20 hodin.

6.3.3 Postup

Díky již vytvořeným modelům stačilo udělat jen stránku pro výpis uživatelů a poté stránku pro jejich upravení. Vytvořil jsem tedy šablony a *Controllery* pro tyto 2 stránky. V dalším kroku jsem je musel přidat do kategorií již přes administrátorské rozhraní pro upravování a přidávání kategorií a do souboru *_setup/layout.xml*, podle kterého se do databáze generují URL, ze kterých pak lze ke stránkám přistoupit. Na stránce se seznamem uživatelů jsou zobrazeny pouze některé

základní informace. Lze se odtud dostat skrze odkaz na stránku, kde je možné uživatele upravit. Formulář je rozdělený na několik částí.

1. Uživatelská data
2. Pracovní pozice
3. Uživatelské role
4. Seznam e-mailů
5. Firemní údaje

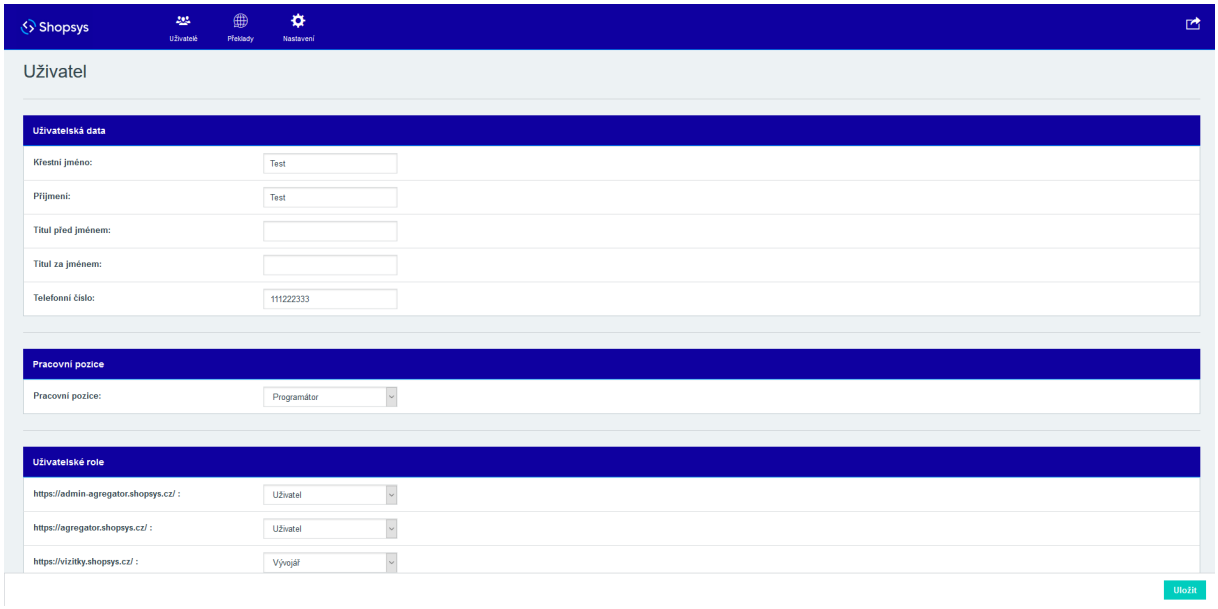
V sekci uživatelská data jsou základní údaje. Ty se v šabloně plní do HTML elementů z Modelu předaného z Controlleru. Dále je zde vidět pracovní pozice, na které jsem si podle stejného postupu vytvořil zvlášť stránku na upravování a přidávání. Formulář využívá konkrétně tyto modely.

- *User*
- *WorkPosition*
- *UserEmail*
- *UserDomainRole*
- *UserRoleType*

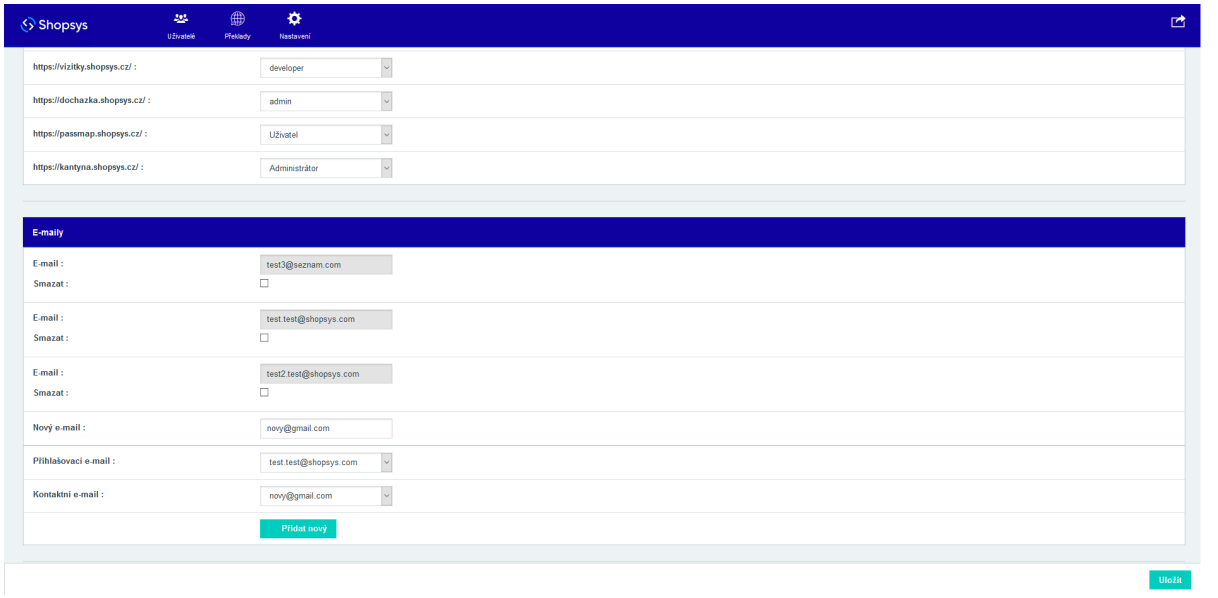
Dalším prvkem ve formuláři lze vidět uživatelské role. Pro každou doménu lze uživateli nastavit jinou roli. Nejsložitějším prvkem ve formuláři byla správa e-mailů. Přihlašovací e-mail může být pouze z domény *shopsys*. Tedy ve vyběru přihlašovacího e-mailu bylo potřeba omezit výběr pouze na tyto e-maily. Poté bylo potřeba, aby se v žádném výběru nezobrazoval e-mail, který má zaškrtnutý příznak smazat. Dále se muselo dynamicky přidávat do tohoto výběru zadaný nový e-mail. Tlačítko *Přidat nový* přidalo další HTML element pro zadání dalšího nového e-mailu. Tuto funkčnost jsem vyřešil pomocí technologií Jasvasciptu a JQuery. Po odeslání formuláře jsou odeslané informace stejnými pravidly validovány na straně serveru. Poté se změny uloží do databáze, pokud bylo vše správně.

6.3.4 Zhodnocení

Úkol se mi podařilo dokončit za přibližně 25 hodin kvůli tomu, že to byly jedny z prvních formulářů na této aplikaci a některé funkce v jádru aplikace pro generování formulářových prvků bylo třeba také upravit.



Obrázek 3: Formulář uživatele první část



Obrázek 4: Formulář uživatele druhá část

6.4 Kategorizace

6.4.1 Úkol

Každý feed je v jiném formátu a s jinými vlastnostmi. Je třeba je načíst do agregátoru a v takovém stavu, aby data bylo možné použít nezávisle na platformě e-shopu.

6.4.2 Odhad úpravy

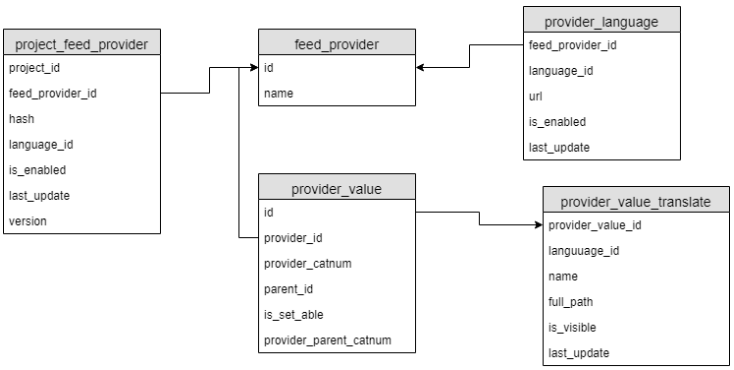
Úprava hrubým odhadem zabere 35 hodin práce.

6.4.3 Postup

Požadovanou funkci lze rozdělit do dvou hlavních kroků.

- Stažení a zpracování datových souborů od poskytovatelů kategorizací
- Exportování těchto dat pro různé e-shopy

V rámci předimplementační konzultace jsme se s kolegou dohodli na následující struktuře uložení dat. Podle této struktury dat jsem tedy vytvořil odpovídající modely *FeedProvider*, *FeedProvi-*



Obrázek 5: ER model struktury dat

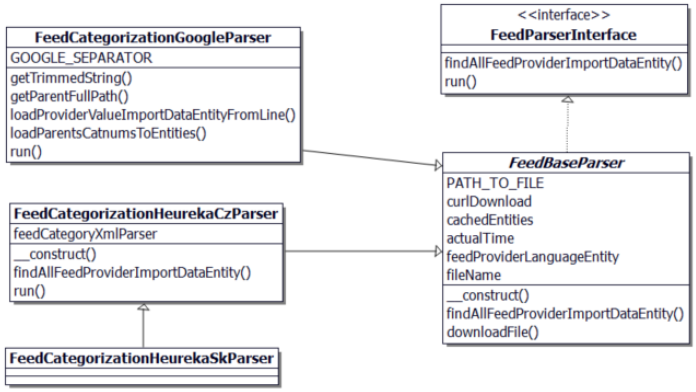
derLanguage, *FeedProviderValue*, *FeedProviderTranslate*. Poté jsem vytvořil skript umístěný v *service/modules/feedcategorization.php*, který po spuštění vybere jednu *FeedProviderLanguage-Entitu*. Vybere *entitu* s nejstarším datem poslední aktualizace a povolením spouštět. Poté se zkontroluje, jestli existuje *Parser* pro zvoleného poskytovatele. Například třída *FeedCategorizationGoogleParser*.

Tato třída se stará o zpracování dat. Musí implementovat metodu *findAllFeedProviderImportDataEntity*, která vrací pole *FeedProviderImpoDataEntity*. Tento model jsem musel vytvořit jako mezičlánek mezi importovanými daty a daty, která se budou ukládat do databáze. Díky tomu mohu sjednotit například výpočet struktury hierarchie kategorií a tím cestu ke kategorii, protože některé služby tyto informace neposkytují.

Byly přidány nové kategorie v administraci pro výpisy a přehled těchto informací a modelů.

V této úpravě bylo potřeba udělat nějaké formuláře pro přidání a editaci modelů *FeedProvider*, *FeedProviderLanguage*.

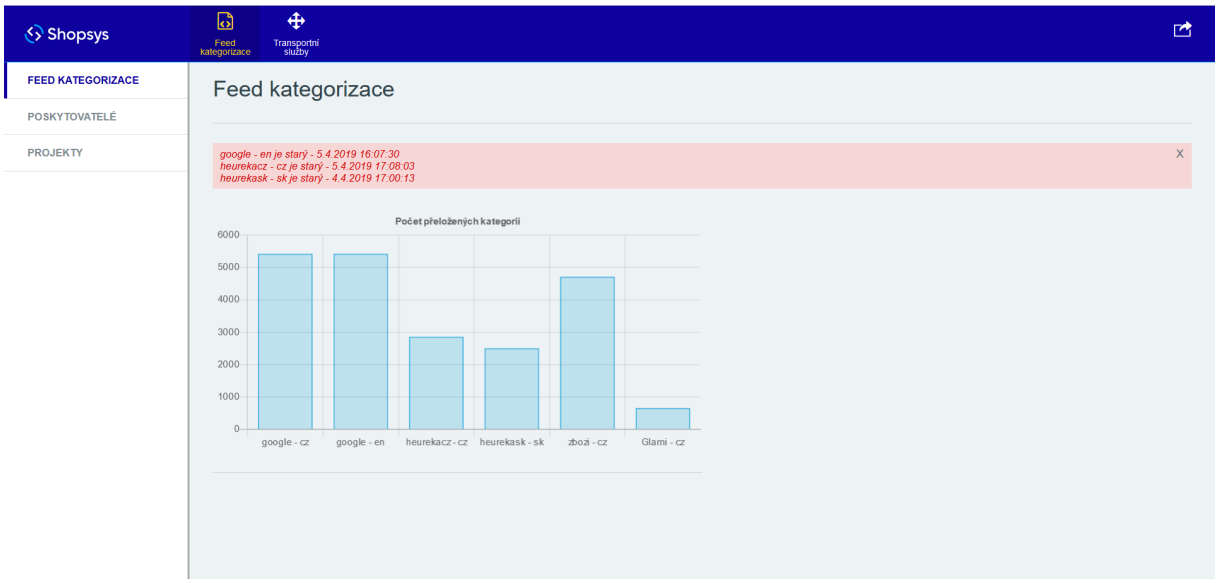
V dalším kroku bylo potřeba udělat vazbu na projekt - e-shop a poskytovatele, od kterého může brát uložená data. Vytvořil jsem další formulář pro editaci a úpravu těchto vztahů. Tento formulář má na rozdíl od zatím implementovaných formulářů prvky, do kterých se dynamicky načítají data pomocí AJAX. Po výběru poskytovatele služby kategorizace se uživateli přenačtou



Obrázek 6: Třídní diagram Parserů

jazyky dle nabídky poskytovatele. Také zde lze vygenerovat hash pro pozdější použití napojení na e-shop.

Na úvodní stránce této kategorie pro kategorizaci jsem přidal základní výpis upozornění. Výpis obsahuje například podezřelá stará data posledních aktualizací. Přidal jsem také graf s výpisem počtu hodnot přeložených kategorií, seskupených podle jazyka a poskytovatele. Graf je vykreslen pomocí *open source* javascript knihovny *Chart.js*.



Obrázek 7: Přehled feed kategorizace

6.4.4 Zhodnocení

Úkol se povedl a v databázi jsou uložena data v jednotném formátu a připravena pro export. Formuláře jsem již díky předchozím úkolům zvládl implementovat mnohem rychleji. Úprava zabrala 40 hodin práce.

6.5 Transportní služby

6.5.1 Úkol

Stejně jako u kategorizace je potřeba data od více služeb dát do jednoho formátu. Data se poté mohou exportovat na e-shopy. Úkolem je získat data o pobočkách služeb *Zásilkovna* a *Uloženka*.

6.5.2 Odhad úpravy

Hrubý odhad úpravy je 35 hodin práce.

6.5.3 Postup

Postup byl velmi podobný minulému úkolu. Problém byl sjednotit data od služeb *Zásilkovna* a *Uloženka*, protože mají odlišnou struktury dat a jinak nazvané podobné prvky poboček. Ne všechny informace jsou zpracovávány, jelikož nejsou na e-shopech využity. Nejprve jsem si vytvořil modely *TransportServiceProvider*, *TransportServiceBranch*.

Dalším krokem bylo vytvoření skriptu *service/modules/transportservicebranches.php* a obdobným způsobem jako skript *service/modules/feedcategorization.php* vezme službu, která má nejstarší datum poslední aktualizace a zkontroluje, zda existuje třída pro zpracování dat. V případě služby *Zásilkovna* je to třída *TransportServiceZasilkovnaParser*. Ta pomocí API služby stáhne všechny pobočky a případné změny uloží do databáze. Pro získání dat od služby *Uloženka* je potřeba využívat její API verze 3. Výměna dat probíhá ve formátu *application/json* s kódováním *UTF-8*. [6]

Pro získání dat od služby *Zásilkovna* jsem využil její API verze 3. Díky jednoduchosti implementace jsem použil metodu *Rest/XML* komunikace. [7] Poté se jednotlivé pobočky uloží do databáze.

K tomuto úkolu patřila také možnost přidávat poskytovatele přes formuláře v administraci. Struktura byla tedy velmi podobná, nejdříve byl výpis, ze kterého se dá dostat na upravení nebo přidání nového poskytovatele. Využil jsem toho, že jsem si v minulém úkolu připravil graf a přidal jednoduché statistiky ohledně počtu poboček podle poskytovatele služby. Také jsem přidal varování ohledně podezřele velkého počtu mizejících poboček od jednoho dodavatele.

6.5.4 Zhodnocení

Úkol se mi povedl implementovat za 45 hodin práce. Naučil jsem se pracovat s rozdílnými API od různých služeb.

7 Závěr

7.1 Získané zkušenosti a dovednosti

Při implementaci úkolů jsem hodně musel komunikovat s kolegy, kteří na projektu také pracovali. Naučil jsem se pracovní postupy. Nejdříve probíhala předimplementační konzultace ke každé větší úpravě se zkušenějším kolegou. Po implementaci musela úprava projít automatickou kontrolou standardů a kontrolou duplicitního kódu. Následně byla předána na kolegu, který se mohl vyjádřit k napsanému kódu, který jsem případně dle jeho doporučení upravil. Naučil jsem se více pracovat s použitými technologiemi a s návrhovými vzory a rovněž zpracovat dokumentaci pro své kolegy.

7.2 Chybějící zkušenosti a dovednosti

Nejdříve jsem měl problém s dodržováním standardů kódu, který jsem ale vyřešil například správným nastavením vývojářského prostředí. Dále jsem se musel naučit, jak implementovat použité návrhové vzory.

7.3 Dosažené výsledky a zhodnocení

Úkoly se mi podařilo dokončit a několika svými návrhy jsem přispěl k vylepšení aplikace. Práce s kolegy mě bavila a na všem jsme se nakonec dokázali domluvit. Při práci na tomto projektu jsem se hodně naučil a získal cenné zkušenosti, které v budoucnosti využiji nejen pro vývoj webových aplikací.

Literatura

[1] JQuery. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, c2001-2019 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/JQuery

[2] XML introduction. W3schools [online]. Sandnes: Refsnes Data, c1999-2019 [cit. 2019-03-09]. Dostupné z: https://www.w3schools.com/xml/xml_whatis.asp

[3] Specifikace XML feedu | Seznam Nápověda. W3schools [online]. Prague: Seznam.cz, 2019 [cit. 2019-03-10]. Dostupné z: https://napoveda.seznam.cz/cz/specifikace-xml-feedu/

[4] BÖHMER, Marian. Návrhové vzory v PHP: [23 vzorových postupů pro rychlejší vývoj]. Brno: Computer Press, 2012. ISBN 978-80-251-3338-5.

[5] Integrating Google Sign-In into your web app. Integrating Google Sign-In into your web app [online]. California: Google, c2019 [cit. 2019-03-16]. Dostupné z: https://developers.google.com/identity/sign-in/web/sign-in

[6] Uloženka.cz API v3 · Apiary. Ulozenkav3.docs.apiary.io [online]. Uloženka.cz [cit. 2019-04-04]. Dostupné z: https://ulozenkav3.docs.apiary.io/

[7] API pro podání zásilek. Zasilkovna.cz [online]. Zásilkovna, c2013 [cit. 2019-04-04]. Dostupné z: http://www.zasilkovna.cz/popis-api/