VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE UMOŽŇUJÍCÍ SPRÁVU PRAVIDEL PRO ROZPOZNÁVÁNÍ SOFTWARU V ASSET MANAGEMENT

WEB APPLICATION FOR MANAGING SOFTWARE DETECTION RULES IN SOFTWARE ASSET MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DAVID DRTIL**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

145529

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Drtil David**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Webová aplikace umožnující správu pravidel pro rozpoznávání softwaru v Asset Management**
Kategorie:	Informační systémy
Akademický rok:	2022/23

Zadání:

1. Seznamte se s problematikou správy softwarových prostředků dle metodiky ITIL (Software Asset Management, zkráceně SAM). Prozkoumejte a porovnejte existující řešení pro SAM, open-source i dostupné komerční produkty. Zaměřte se především na nástroje pro sledování spouštěných procesů a detekci instalace programů na počítačích, dále také na formát zápisu pravidel a způsoby jejich využití pro detekci a rozpoznávání nainstalovaných programů.
2. Analyzujte požadavky na webovou aplikaci pro příjem dat z dostupných nástrojů pro sledování spouštěných procesů a detekci nainstalovaných programů. Aplikace bude umožňovat správu, testování, ladění a nasazení použitých pravidel nad daty přijatými z uvedených nástrojů.
3. Po konzultaci s vedoucím aplikaci dle požadavků navrhněte. Zaměřte se na architekturu aplikace, nasazení a škálování, vrstvu uživatelského rozhraní, aplikační logiky a databáze, atp. Navrhněte také způsob integrace aplikace do existujících řešení pro SAM, včetně konverze vstupních a výstupních formátů a adaptace rozhraní integrovaných služeb.
4. Implementujte navrženou aplikaci tak, aby ji bylo možné provozovat jako službu v prostředí Azure App Service. Navrhněte a implementujte jednotkové, integrační i uživatelské testy a implementovanou aplikaci důkladně otestujte.
5. Výsledek zdokumentujte, vyhodnoťte vlastní přínos i použitelnost výsledné aplikace a navrhněte možná rozšíření.

Literatura:

- Vernon Lloyd, Colin Rudd. ITIL Service Design. The Stationery Office, 2007. ISBN 978-0-11-331047-0. 2021
- Shirley Lacy, Ivor Macfarlane. ITIL Service Transition. The Stationery Office, 2007. ISBN 978-0-11-331048-7. 2021
- David Cannon, David Wheeldon. ITIL Service Operation. The Stationery Office, 2007. ISBN 978-0-11-331046-3.

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	18.10.2022

Abstrakt

Náplní práce je vytvoření aplikace pro správu a rozvíjení knihovny pravidel, která slouží pro rozpoznávání softwaru na zařízeních. Knihovna softwarových produktů a pravidel má zvláštní význam pro systém Software Asset Management a jsou na ní závislé další moduly. Rozpoznávání softwaru jako takové probíhá aplikací pravidel vůči přítomným informacím o instalaci programů. Formáty zápisu pravidel a metodologie mohou být různé, vychází se však z původního již osvědčeného postupu společnosti Alvao. Práce je zaměřena především na zjednodušení a automatizaci procesu vytváření či ladění pravidel vhodným navrhováním samotných pravidel.

Abstract

The content of this thesis is the management of a library of rules used for software recognition on devices. The application has special purpose in the Software Asset Management system and other components depend on it. Software product recognition is performed by applying rules to the information present about the installation of the programs. Formats for writing rules and methodology may be different, but they are based on the original Alvao best practice. The work is primarily focused on simplifying and automating the process of creating or adjusting rules by means of appropriate suggesting of the resulting rules, categorization or other related actions.

Klíčová slova

rozpoznání software, rozpoznávací pravidla, detekce instalace software, knihovna softwarových produktů, katalog software, Naivní Bayesova klasifikace, Software Asset Management, SAM, ASP.NET Core MVC

Keywords

software recognition, recognition rule, detection of software installation, software product library, software catalog, Naive Bayes classification, Software Asset Management, SAM, ASP.NET Core MVC

Citace

DRTIL, David. *Webová aplikace umožňující správu pravidel pro rozpoznávání softwaru v Asset Management*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Webová aplikace umožňující správu pravidel pro rozpoznávání softwaru v Asset Management

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením RNDr. Marka Rychlého, Ph.D. Další informace mi poskytla společnost Alvao, s.r.o. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Drtil
9. května 2023

Poděkování

Tímto bych chtěl poděkovat vedoucímu bakalářské práce RNDr. Markovi Rychlému, Ph.D. za nevídaně rychlé odepisování na zprávy a za předávání cenných rad na konzultacích při realizaci této práce. Dále bych chtěl poděkovat Ing. Janu Minaříkovi ze společnosti Alvao, s.r.o. za poskytnutí materiálů a nespočet odborných konzultací, při kterých jsem měl tu možnost získat zpětnou vazbu k řešení a tím práci zdokonalit.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V organizacích je klíčové mít přehled a kontrolu nad tím, jaký software je nainstalován a provozován na firemních zařízeních. Vytváření přehledů má za cíl zajistit zejména to, že organizace jsou v plném souladu s licenčními podmínkami používaného softwaru, že je používán právě ten správný počet těch požadovaných licencí tak, aby nevznikaly zbytečné náklady. Je především důležité, aby se daly tyto přehledy vytvářet automatizovaně a bylo tak možné provádět časté kontroly pro informování správce v reálném čase o aktuálním stavu licencí a případných nedostatcích, jako jsou nevyužité či nesprávné licence. Přínosná je také možnost ověření, zda se nenachází na některém ze zařízení zakázaný software či zranitelná verze softwaru, která by mohla způsobit bezpečnostní riziko.

Prerekvizitou k automatickému vytváření těchto přehledů je sběr dat o nainstalovaných programech a jejich následné rozpoznávání pomocí knihovny pravidel. Rozpoznávací pravidla jsou založená na informacích, jež programy při své instalaci zaznamenávají do registrových záznamů a metadat spustitelných souborů. Identifikace programů následně probíhá aplikací pravidel vůči přítomným informacím na zařízení.

Knihovnu s těmito pravidly je nutné stále rozvíjet a udržovat aktuální, jelikož organizace vydávají velké množství nových produktů či jejich verzí a edicí. Největší komplikací je nejednotnost ve způsobu zaznamenávání informací o instalaci programů, a to napříč vydavateli softwaru, ale i v rámci různých produktů konkrétního vydavatele. Navíc názvy produktů a jejich edic jsou v různých zemí distribuovány pod odlišnými, lokalizovanými názvy a mohou se lišit, i když se jedná o stejný produkt. V průběhu času také často dochází k přejmenování produktů, a dokonce i samotných organizací.

Všechny výše uvedené případy musí aplikace podporovat a umožňovat správci knihovny je efektivně řešit. Důraz je kladen především na částečnou automatizaci a zjednodušení vytváření nových pravidel pomocí vhodného navrhování výsledných pravidel, kategorizace či jiných s tím spjatých akcí.

Práce je členěna do 4 částí. V první části jsou popsány teoretické aspekty práce k pochopení problematiky a také jsou představeny metodiky existujících řešení software katalogů. Druhá část se věnuje specifikaci požadavků a návrhu aplikace včetně zvolení technologií, na kterých bude celý systém postaven. Třetí část se zaobírá implementačními detaily webové aplikace a programů provádějících zpracování záznamů a klasifikaci. Čtvrtá část se zabývá tvorbou testovacích scénářů a testováním aplikace jednotkovými, integračními i uživatelskými testy. V závěru se nachází celkové zhodnocení práce, jsou zde shrnuty dosažené výsledky a uvedeny návrhy pro další možná rozšíření práce.

Kapitola 2

Studium problematiky

Tato kapitola se zabývá stručným popisem současných řešení. Obsahuje informace potřebné k pochopení problematiky a zasazení samotné knihovny softwarových produktů do kontextu s ostatními moduly.

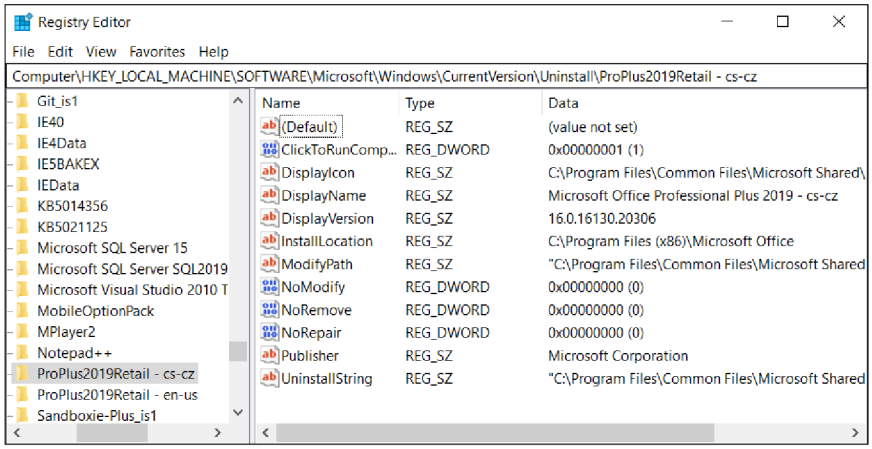
2.1 Obsah softwarové knihovny

Správa softwarových licencí a aktiv není úspěšná, pokud jsou produkována data a výsledky, které jsou plné chyb či mezer. V Software Asset Management (SAM) tvoří velkou část práce sbírání a zpracování různorodých dat, která jsou navíc na sobě závislá. Knihovna nazývaná také často jako softwarový katalog, obsahuje pravidla k rozpoznání, identifikaci softwaru a také informace o softwarové licenci. Vývoj a údržba této knihovny je dále delegována na odpovědné osoby, které pravidelně tuto knihovnu rozvíjí a přizpůsobují, protože neexistuje žádná průmyslová norma, která by definovala, jak je software po instalaci licencován nebo identifikován. [6]

Správce knihovny navíc určuje i další informace, které jsou zjistitelné z načtených záznamů, jako je typ licence produktu, kategorii softwaru. Za účelem zjištění těchto informací vyhledává název produktu na webovém vyhledávači, a především z oficiálních stránek organizace analyzuje typ licence.

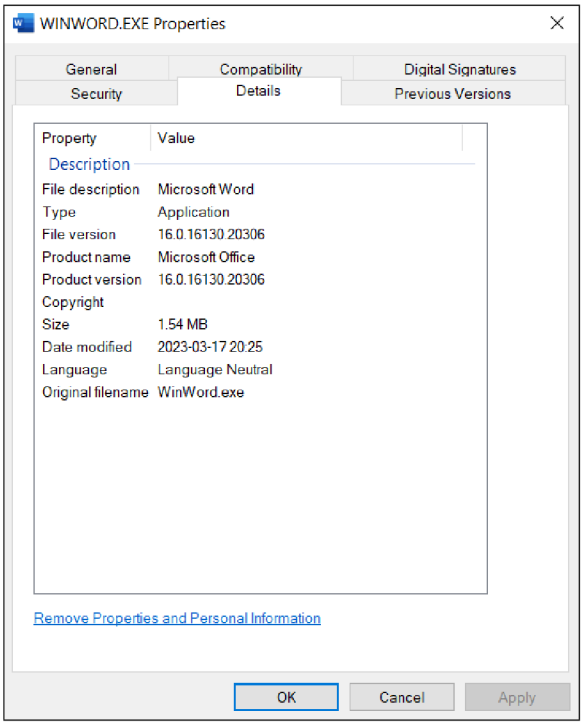
2.1.1 Typy záznamů k identifikaci programů

Na Windows proti jiným operačním systémům jsou data o přítomnosti programu zaznamenána do registrů, konkrétně do sekce „uninstall“. Zde se nacházejí stejné informace jako ty zobrazované v ovládacích panelech na záložce „Přidat nebo odebrat programy“. Tyto záznamy mají při rozpoznávání nejvyšší váhu, jelikož licencované, komerčně prodávané produkty mají tyto záznamy velmi často kvalitně zpracované a obsahují kompletní informace pro rozpoznání tohoto produktu, a tak není potřeba vytvářet další komplexnější pravidla. Na následujícím obrázku 2.1 je ukázáno, jaká data bývají typicky obsažena v registrovém záznamu.



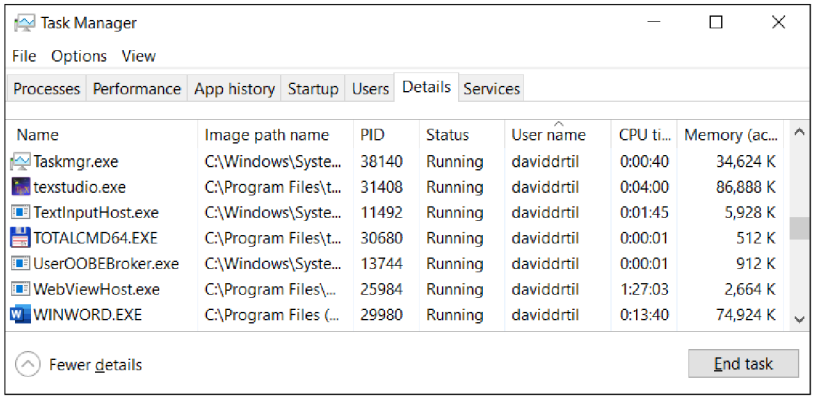
Obrázek 2.1: Ukázka struktury a obsahu registrového záznamu

Další významnou položkou k detekci software neboli ke zjištění, zda je určitý software přítomný na zařízení, jsou metadata spustitelných souborů. Pro nalezení takových souborů dochází k prohledávání disku, obzvlášť důležitá je složka „Program Files“. Ta obsahuje největší množství relevantních programů pro Software Asset Management, ale bývají prohledávány i další složky. U nalezených spustitelných souborů s příponami exe ale i dll, které často úzce souvisí s těmito spustitelnými soubory, jsou načítána ze záhlaví souborů potřebná metadata. Formát dat záhlaví souboru a příklad ukázkových dat konkrétní aplikace Microsoft Word 2016 je uveden na obrázku 2.2.



Obrázek 2.2: Příklad dat obsažených v záhlaví spustitelného souboru

Informace o nainstalovaných produktech jsou dále získávány z monitorování aktivních procesů, ty mohou vést k nalezení netradičně uložených souborů. Navíc se z těchto dat vytváří statistiky o používání určitých programů, jenž organizace využívají k vyšší kontrole nad firemními zařízeními či zhodnocují počty potřebných licencí při dalších nákupech. Data o aktivně běžících procesech jsou programově získávána a zapisována do souboru, ale lze tyto informace zobrazit vestavěným programem „Správce úloh“, jež jsou pro názornost uvedena na obrázku 2.3.



Obrázek 2.3: Ukázka aktivních procesů

2.2 Sběr informací o nainstalovaném software

Při získávání dat o nainstalovaných softwarových produktech velmi záleží na operačním systému daného stroje. Na různých operačních systémech jsou data o instalacích programů zaznamenávána odlišně, a proto také musí být použity odlišné metody či alespoň modifikovaná nastavení.

Na koncových zařízeních je provozována služba tzv. *agent*, který provádí periodicky skenování. Ke spouštění dochází typicky každý den v určitou stanovenou hodinu tak, aby nedocházelo v nevhodný čas k vytěžování zdrojů stroje nebo zahlcování sítě. Výstupem je sada nálezů, které jsou automaticky odeslány na nakonfigurovaný *kolektor*. Tyto nálezy se vyhodnotí pravidly ze softwarové knihovny a vznikne seznam nainstalovaných produktů z této knihovny. Některé záznamy mohou být nerozpoznané a je potřeba je odeslat na zpracování. Nerozpoznané záznamy jsou odeslány jako soubor na webovou službu SwLib, ta je přijme a uloží do úložiště.

Pro zajištění správnosti rozpoznávání je navíc potřeba knihovnu pravidelně aktualizovat a mít staženou nejnovější verzi. Kontrola se opět běžně provádí automaticky v určitých intervalech, ale lze aktualizaci také vynutit a spustit ji manuálně. Aktualizují se pouze nově přidané či modifikované záznamy, na základě porovnání časových značek u každé položky.

Jak již bylo zmíněno, způsoby uložení dat o přítomnosti programů se liší v závislosti na operačním systému stroje, a proto se také liší i nástroje pro sběr informací. Na platformě s jádrem Linux jsou data o softwaru získávána z balíkovacích nástrojů jako jsou například *RPM* či *YUM* nebo také prostřednictvím protokolu *SNMP*.

2.3 Průzkum alternativních řešení

Vzhledem k důležitosti Software Asset Managementu pro organizace existuje velké množství komerčních i nekomerčních řešení, které řeší problém s knihovnami produktů a rozpoznávacích pravidel.

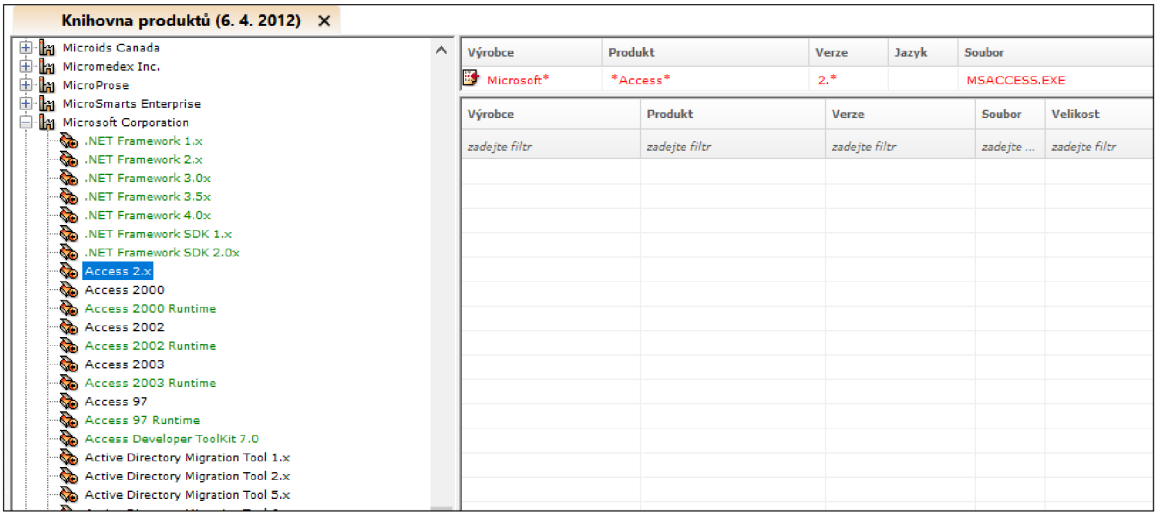
K použití se nabízí dokonce i online katalogy softwaru, které navíc vystavují API, neboli aplikační programové rozhraní. Tyto služby jsou většinou placené. Neplacené alternativy nejsou úplné, protože údržba a rozvíjení katalogů o nové produkty jsou nákladné procesy. Příkladem online katalogu pro komerční použití je Capterra[1] nebo G2, které nabízejí navíc srovnávání produktů podle určitých vlastností, kategorií, cen za licenci atd. Jedná se o přímou konkurenci firmy Alvao a z hlediska závislosti na produktech 3. stran nebo dokonce nutnosti platit za tyto služby není možné využít. Volně zveřejněné softwarové katalogy by také nebyly spolehlivé z důvodu, že zákazníky firmy Alvao jsou nejčastěji české a slovenské firmy, které používají tuzemské, lokalizované produkty. Takovými produkty mohou být různé podnikové systémy na míru ERP nebo produkty s lokalizovaných názvy, které by nebyly rozpoznány.

Z nejpoužívanějších nekomerčních alternativ systému *SAM* jsem vyzkoušel open–source *GLPI*[2], u které se bohužel nijak nezpracovávají detekované nainstalované programy pomocí knihovny produktů a je tato část práce přenesena na IT správce, proto nebylo možné se inspirovat.

Z komerčních produktů jsem vyzkoušel ukázkovou aplikaci *AuditPro*[3], řešení *LogMeIn Central*[4] a nahlédnul jsem na alternativu *Asset Management For ESET*. Komerční řešení byla zpracována mnohem kvalitněji, bohužel ale firmy nezveřejňují interní části svých systémů, kterou je právě detekce a rozpoznání programů, proto ani zde nebylo příliš z čeho čerpat.

Ačkoliv jsem u těchto aplikací neměl možnost nahlédnout, jak řeší problém s detekcí programů a jaké používají formáty pravidel pro rozpoznání, tak jsem si tímto vyzkoušením aplikací prohloubil znalosti o používání knihovny produktů. Konkrétně v aplikaci *AuditPro* jsem narazil na to, že knihovnu produktů mají tvořenou strukturovaně ve stromu, kde na nejvyšší úrovni se nachází výrobce a teprve pod ním se nacházejí jednotlivé produkty, viz 2.4.

[1] Webová stránka produktu Capterra *GLPI*: https://www.capterra.com/
[2] Hlavní stránka produktu *GLPI*: https://glpi-project.org/
[3] Demo verze produktu *AuditPro* dostupná na: http://www.auditpro.cz
[4] Produkt *LogMeIn Central* dostupný na stránce: https://www.logmein.com/central



Obrázek 2.4: Knihovny produktů z aplikace AuditPro

Právě knihovna produktů z aplikace AuditPro byla podnětem pro návrh nového formátu pravidel, formou stromové struktury. Pravidla by se skládala z více úrovní, je zde předpoklad, že by se tímto redukoval počet pravidel i nutnost vytvářet zbytečně kompletně nová pravidla. Z nerozpoznaných záznamů by se také nejspíše jednodušeji generovaly nové produkty pro nabízení, jaký produkt se má vytvořit. Knihovna produktů a pravidel pak může mít následující stromovou strukturu:

```
vydavatel
└─ obecný název produktu
   ├─ verze - rok
   ├─ edice
   ├─ platforma
   └─ vývojová větev
```

Pro úplnost jsou zde uvedeny příklady obsahů logických částí názvu neboli jednotlivých uzlů ve stromě:

- vydavatel – Microsoft, Adobe
- obecný název produktu – Word, Excel, Access, Skype
- rok vydání či verze – pro rok vydání 2 nebo 4 číslice nebo pro verze až 4 čísla
- edice – Home, Standard, Professional, Enterprise
- platforma – x86, x64, 64-bit, for Linux, for Windows
- cílový jazyk – en, cs-cz, Czech, English
- vývojová větev – release candidate, release, preview, alpha, beta

2.4 Normy pro SAM

Procesy SAM, mezi které patří správa knihovny produktů a pravidel, se řídí normami ITIL a ISO/IEC 19770. Norma ITIL, anglicky celým názvem Information Technology Infrastructure Library, poskytuje rámec pro správu IT služeb, včetně metodiky pro správu softwarových aktiv nazvané Software Asset Management (SAM). ITIL nabízí doporučené postupy pro implementaci SAM v organizaci, což zahrnuje správu licencí, správu nákupů, správu produktů a správu životního cyklu softwaru [5]. Norma ISO/IEC 19770 je mezinárodní standard pro správu softwarových aktiv, který definuje procesy a postupy pro správu licencí, správu nákupů a správu životního cyklu softwaru.

Tyto normy jsou kompatibilní a mohou být vzájemně propojeny při implementaci správy softwarových aktiv v organizaci. Spolu poskytují různé nástroje a postupy, které organizacím umožňují účinně spravovat svá softwarová aktiv a snižovat rizika spojená s nesprávnou správou a nelegálním používáním softwaru.

2.5 Klasifikace produktů

Nerozpoznané záznamy, které byly načteny nově skenerem 2.2, je potřebné očistit o určité nezajímavé, nedůležité produkty pro licenční záležitosti a další účely *SAM*. Takovými produkty typicky bývají ovladače, hry ale i některé méně známé bezplatné produkty. V případě, že se jedná o tuto kategorii produktů, jsou pro registrové a souborové záznamy vytvářena tzv. „vyškrkávací“ pravidla, jinak také nazývaná jako obecná nebo neklíčová pravidla. V textu je dále používán pojem vyškrkávací pravidlo, protože firma Alvao je zvyklá na tuto terminologii. Význam vyškrkávacího pravidla spočívá v tom, aby bylo možné nepodstatné záznamy rozlišit při další detekci a dále tak nepřekážely. Pro komerční produkty se vytváří klíčová pravidla, ta na rozdíl od pravidel vyškrkávacích se již vážou na produkt a pro vytvoření tohoto pravidla je tedy nutné buď vytvořit nový produkt nebo přiřadit již nějaký existující.

Databáze od firmy Alvao obsahuje velké množství již zpracovaných a správně označených dat o registrových, souborových záznamech, pravidlech a náležících produktech, proto se zde přímo nabízelo použít tato data k napomáhání s klasifikací produktů. Před samotným vytvářením pravidla je totiž nutné rozhodnout, zda je produkt klíčový nebo neklíčový a podle toho samozřejmě, jaký typ pravidla má vzniknout.

2.5.1 Základní přístup klasifikace

Ke klasifikaci produktů se nabízí použít statistické metody, jelikož vynikají svou přesností a rychlostí i při aplikaci na velké datové sady.

Nejintuitivnější statistická metoda uvažuje četnosti jednotlivých slov k určení, zda produkt má být zařazen jako klíčový nebo naopak jako neklíčový. Metoda provede tokenizaci názvů produktů na jednotlivá slova. U tokenů je zjištěn počet výskytů patřících do uvedených kategorií a následně určena pravděpodobnost, se kterou slovo přísluší do jedné z těchto kategorií. Tato pravděpodobnost je následně porovnána se zadanou prahovou hodnotou, která může být nastavena například na 90 %, a při překročení této prachové hodnoty je produkt označen jako klasifikovaný. Tímto způsobem je možné odhalit v názvech i na první pohled méně jednoznačná klíčová slova, která jsou však rozhodující pro určení klasifikace. Příkladem slov pro určení klíčových produktů mohou být slova *Professional*, *Photoshop*

nebo *Autodesk*. Zatímco mezi slova pro určení neklíčových produktů patří například *USB*, *Printer* nebo *Driver*.

Kategorizace podle jednotlivých slov nemusí být v určitých případech příliš spolehlivá, pokud dojde k velkému průniku slov, která patří do obou tříd zároveň, a nebude možné na základě těchto slov přesně klasifikovat. Řešením je rozdělit názvy pomocí metody *N-Gram* do sledů n po sobě jdoucích slov z dané posloupnosti neboli do n-tic slov, čímž je možné zachytit více kontextu. Na rozdíl od použití vyšších řádů metody *N-Gram* pro analýzu textu přirozeného jazyka, mohou být názvy kvůli kratšímu charakteru rozdělovány pouze po dvojicích slov, maximálně trojicích slov. Používá se tedy metoda *Bi-Gram* nazývaná jako „First order Markov model“ a *Tri-Gram* metoda jinak taky pojmenovaná jako „Second order Markov model“.

Při klasifikaci produktů by měli být také zohledněni výrobci produktů, protože je osvědčené, že většina výrobců jako například *AMD*, *Dell* a další vydávají primárně pouze ovladače nebo jiné produkty, které jsou téměř výhradně řazeny jako neklíčové. U výrobců není však kvůli nižší variabilitě nutné název rozkládat na jednotlivá slova jako u názvu produktů a je možné používat pro klasifikaci celý název.

Vlastnosti vedoucí ke klasifikaci produktů by měly být při výpočtu uvažovány všechny zároveň, nikoliv jednotlivě. Získané pravděpodobnosti lze průměrovat a poté je teprve srovnávat s prahovou hodnotou, která bude umocněna počtem průměrovaných pravděpodobností.

2.5.2 Bayesovská klasifikace

Mezi nejpoužívanější statistické metody pro klasifikaci či kategorizaci textových dat do daných tříd patří Bayesovské klasifikátory. Bayesovská klasifikace je založena na Bayesově teorému, jinak také nazývaným jako Bayesova věta. Ta je ukázána na (2.1) a má následující znění: Mějme vzorek dat X, též nazývaný jako evidence, a hypotézu H označující třídu, do které vzorek dat patří. *Apriorní* pravděpodobnost hypotézy $P(H)$ odpovídá znalostem o zastoupení jednotlivých tříd bez ohledu na jakékoliv další informace. Klasifikací se snažíme určit podmíněnou pravděpodobnost $P(H|X)$, což je tzv. *aposteriorní* pravděpodobnost neboli pravděpodobnost, že vzorek dat splňuje hypotézu za podmínky, že již víme, že nastalo X. $P(X)$ vyjadřuje pravděpodobnost evidence [3].

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{2.1}$$

Na použití Bayesovy věty jsou založené Naivní Bayesovské klasifikátory, jež předpokládají, že vliv hodnoty atributu na určitou třídu je nezávislý na hodnotách ostatních atributů. Kvůli této vlastnosti, že nezávisí na pořadí jednotlivých slov a různý slovosled tak neovlivní výsledek klasifikace, jsou klasifikátory také pojmenované jako naivní. Klasifikace pomocí Naivního Bayesovského klasifikátoru probíhá na základě modelu zvaného pytel slov, anglicky bag of words. Při tomto přístupu je provedena frekvenční analýza slov a jsou zjištěny četnosti výskytů jednotlivých slov. Následně je na základě těchto četností vypočítávána pravděpodobnost, zda testovaný vzorek přísluší do dané kategorie.

Vlastnost nezávislosti na pořadí slov Naivních Bayesovských klasifikátorů je považována za výhodnou pro klasifikaci produktů, protože mezi různými verzemi a edicemi produktu bývají totiž slova v názvech různě přeuspořádána a nejsou mezi jednotlivými slovy tak pevné vazby jako v přirozených jazycích.

Existují různé druhy Naivních Bayesovských klasifikátorů, mezi něž patří Gaussův, Bernoulliho a Multinomiální Naivní Bayesovské klasifikátory, které se používají v různých aplikacích. Multinomiální Naivní Bayesův klasifikátor se používá pro textovou klasifikaci dat, například k odfiltrování nevyžádaných zpráv, jimiž jsou různé reklamní či podvodné nabídky. Na následujícím příkladu vycházejícího z článku [8] je představen postup pro klasifikaci, zda je zpráva normální nebo nevyžádaná. Tento postup je aplikovatelný pro jakoukoliv textovou klasifikaci dat, včetně klasifikace produktů na klíčové a neklíčové produkty.

Metoda uvažuje četnosti výskytů slov, se kterou se nacházejí slova v normálních zprávách a se kterou v nevyžádaných zprávách. Na základě těchto četností lze získat pravděpodobnost pro každé slovo, s jakou se vyskytuje v jedné z kategorií. Tato pravděpodobnost se vypočítá jako podíl počtu výskytů slova patřícího do dané kategorie z celkového počtu výskytů. Po zpracování četností jednotlivých slov z trénovací sady dat lze rozhodovat o klasifikaci zpráv z testovací sady. Pro rozhodnutí o klasifikaci nové zprávy se začíná počátečním odhadem pravděpodobnosti, že jakákoli zpráva, bez ohledu na její obsah, je normální zprávou. Počáteční odhad pravděpodobnosti, nazývaný jako *apriorní* pravděpodobnost, je získán jako podíl vypozorovaných normálních zpráv z celkového počtu zpráv. Tento počáteční odhad pravděpodobnosti je vynásoben pravděpodobnostmi jednotlivých slov vzhledem k tomu, že se jedná o normální zprávu. Tímto způsobem je získáno skóre, které zpráva získá, pokud se jedná o normální zprávu. Stejný postup se provede pro výpočet skóre, zda zpráva patří do druhé kategorie. Klasifikace zprávy je následně rozhodnuta podle nejvyššího skóre. V některých aplikacích se neuvažuje apriorní pravděpodobnost a bývá zcela vynechána z výpočtu, čímž je zanedbána předchozí pravděpodobnost příslušnosti zpráv do kategorií. Při výpočtu nastává problém s agresivitou 0 při násobení, pokud některé ze slov se nevyskytuje v jedné z kategorií, tak dochází ke zkreslení výsledné klasifikace. Problému se předchází přidáním tzv. pseudo-počtu α všem výskytům slov, kdy $\alpha = 1$. Je zde navíc důležité, aby se tento pseudo-počet nebyl promítnut do apriorní pravděpodobnosti.

V článku [4] jsou ukázány různé techniky pro zlepšení výsledků klasifikace. Například srovnávání podílu pravděpodobností s prahovou hodnotou, které vede k menšímu poměru nesprávně klasifikovaných dat. Dále je v článku ukázáno řešení pro zamezení podtečení čísel s plovoucí desetinnou čárkou, ke kterému může eventuálně dojít při součinu nízkých hodnot pravděpodobností. Jsou zde navíc shrnuty dosažené výsledky experimentů s různými modifikacemi Naivního Bayesovského klasifikátoru, kterými se lze inspirovat.

Ke klasifikaci produktů se z počátku uvažovalo o použití strojového učení, které je používáno v mnoha alespoň částečně podobných aplikacích a existuje tak velké množství již předtrénovaných modelů. Takové modely, které jsou natrénované na různých kolekcích dat, jsou ke stažení z veřejně přístupných databází, příkladem může být stránka Hugging Face[5], která tyto modely seskupuje na jednom místě. Avšak pro potřeby klasifikace produktů by bylo vyžadováno vytvořit a natrénovat vlastní model daty z databáze firmy Alvao. Pro tyto účely byla prozkoumána a vyzkoušena hluboká neuronová síť s více vrstvami, anglicky Multi-Layer Perceptron, kterou se často nahrazují statistické klasifikační metody.

Z hlediska nižší paměťové i výpočetní náročnosti bylo vyhodnoceno, že je pro náš případ klasifikace produktů vhodnější využít statistických metod.

[5]Předtrénované modely dostupné na: https://huggingface.co/datasets

2.6 Vyhledávání a přiřazování produktů

V průběhu času se název produktu mění a s tím také vzniká problém s vyhledáváním a následným přiřazováním produktů. Správce knihovny se snaží najít nejpodobnější produkt či pro kontext nějakou množinu souvisejících produktů. Nejzákladnějším přístupem je prosté vyhledání, filtrování záznamů pomocí podřetězců, na úrovni databáze pomocí operátoru *like* v klauzuli *WHERE*. Pro většinu případů je tento způsob zcela dostačující, ale existují i případy, kdy se produkt touto vyhledávací metodou velmi složitě hledá kvůli nekonzistenci názvů produktů. Může také nastat situace, kdy je nalezeno velkého množství výsledků a produkt se pak správci špatně přiřazuje a je žádoucí tyto výsledky hledání seřadit. Vyhledané produkty by mohly být řazeny podle data vzniku od nejnovějších záznamů po nejstarší, jelikož nejnověji vytvářené produkty by mohly být relevantnější a je zde větší pravděpodobnost, že bude patřit registrový záznam právě k tomuto produktu. V případě tohoto řazení se však jeví list výsledků hledání jako neuspořádaný a špatně se prochází. Vhodnějším způsobem je řazení pomocí vzdálenostních metrik nazývaných také jako podobnostní metriky a následně navíc i podle názvu.

2.6.1 Podobnostní metriky

Zjišťování shody řetězců má význam v mnoha odvětvích. Jedním z nich jsou internetové obchody (anglicky ecommerce), které se potýkají s podobným problémem, a to nalezením nejrelevantnějších produktů pro vyhledávanou frázi. Kvalitní vyhledávání a nabízení podobných produktů je pro internetové obchody zásadní, proto tomu také věnují velké úsilí. Kromě názvu produktu se běžně srovnává více parametrů jako jsou popis, specifikace produktu, výrobce, podobnosti obrázků, cena a také manuálně přidané štítky, které poskytují dodatečné informace, jež by nebyly možné vydedukovat z prostého názvu produktu. Označující štítky následně lépe kategorizují a specifikují produkt.

Ačkoliv tyto štítky mají své uplatnění při vyhledávání a kategorizaci produktů na internetových obchodech, pro využití v knihovně produktů je tento způsob nevyhovující, jelikož by znamenal pro správce knihovny práci navíc. Místo toho je zajímavé se zaměřit na metody, které se používají pro ohodnocení shody zadané fráze s názvy produktů nacházejících se v katalogu internetových obchodů, protože dochází k přesně stejnému problému jako v naší aplikaci, a to nekonzistenci u názvů produktů, kdy stejný produkt je nabízen různými prodejci pod mírně odlišnými názvy, což znemožňuje jejich vyhledávání. Problém se objevuje v menších rozdílech mezi různými názvy produktů a ve velkých rozdílech v názvech stejného produktu.

Ve článku [10] jsou uvedeny různé kombinace názvů stejných produktů a také metody k řešení toto problému. Je možné použít podobnostní metriky založené na tokenech či znacích v závislosti na tom, jaký problém u názvů je třeba řešit.

Jednou z nejznámějších metrik je Levenshteinova editační vzdálenost. Metrika je založená na znacích a měří minimální počet operací sestávajících z vložení, vymazání nebo nahrazení jednoho znaku, které jsou nutné ke změně jednoho slova na druhé. Modifikace Damerau-Levenshtein vzdálenost se od klasické Levenshteinovi vzdálenosti liší tím, že kromě tří klasických editačních operací s jedním znakem zahrnuje mezi své povolené operace také transpozice dvou sousedních znaků. Pomocí editační vzdálenosti lze odhalit překlepy názvech. Podrobněji popsané na [1].

Dalším typem jsou metriky založené na tokenech, tyto metriky se používají, pokud nezáleží na pořadí slov / tokenů. Prvním krokem těchto metod je tokenizace řetězců do

sady tokenů. Tokenizovat lze řetězce jako posloupnost znaků o pevně zadané délce nebo po jednotlivých slovech. Následně se metrikami kvantifikuje podobnost na základě těchto sad tokenů, obvykle pokud jsou řetězce podobné, tak se jejich sady tokenů překrývají. Mezi tyto metriky patří například Kosinová podobnost, Jaccardův koeficient podobnosti, Sorensen–Dice koeficient podobnosti a N-Gram. Každá metoda má jiné vlastnosti, a proto je tedy každá vhodná pro různé aplikace.

Kapitola 3

Analýza požadavků a návrh systému

Vývoj nové aplikace si vyžaduje pečlivou analýzu požadavků a návrh systému, aby byla aplikace přesně navržena a vyvinuta s ohledem na potřeby uživatelů. Tato kapitola se proto zabývá sběrem informací a definováním požadavků na aplikaci, které budou vycházet ze specifikací a potřeb uživatelů. Následně se budeme věnovat návrhu funkcionalit aplikace, které budou plně vyhovovat definovaným požadavkům. Zaměříme se také na výběr technologií a nástrojů pro implementaci navrženého systému. Cílem této kapitoly je tedy vytvořit kompletní návrh a plán pro vývoj nové aplikace, který bude zahrnovat veškeré klíčové prvky a zajištění správného fungování systému.

3.1 Analýza současného stavu

Interní aplikaci začali správci knihovny používat v roce 2008, od té doby se na aplikaci udály pouze menší změny, ale v jádru aplikace se zachovaly funkce, u kterých se ukázalo během let, že nejsou potřebné, a které v důsledku znemožňují úpravy aplikace. Původní aplikace byla navíc napsána ve starší verzi platformy .NET Framework a z hlediska dalšího vývoje bylo žádoucí přejít na novější verzi .NET Core. Proto také bylo navrženo přepsání celé aplikace, která využije již získané zkušenostmi během let.

Na samém začátku bylo klíčové seznámit se s hlavními uživatelskými scénáři a zjistit, jaké operace jsou nejčastěji prováděny a za jakým účelem. Na základě těchto poznatků pak mohly být navržené určité změny v aplikaci, odhalily se repetitivně prováděné procesy, ale také určily části aplikace, které se mají zachovat, protože byly navrženy správně a nejsou problémové.

Správce knihovny pravidelně s určitou periodou, typicky každý týden, prochází seznam s nerozpoznanými záznamy od všech zákazníků a zpracovává je tak, aby se knihovna udržovala aktuální. Jednotlivé nerozpoznané záznamy je výhodné seskupovat, řadit a filtrovat podle nějaké společné vlastnosti, například podle názvu záznamu nebo podle výrobce. Obvykle totiž patří několik záznamů ke stejnému produktu a snadněji se zpracovávají najednou. V původní aplikaci bylo možné seznamy pouze řadit a možnost vyhledávání ve sloupcích chyběla.

Následně se rozhoduje, jaké pravidlo se má pro dané záznamy vytvořit. Pro produkty komerčního charakteru se vytváří „klíčové“ pravidlo a pro ostatní nepodstatné produkty pravidlo „vyškrkávací“, čímž dojde k odfiltrování a očištění těchto záznamů. Při tomto

rozhodování, které záznamy se mají odfiltrovat, chybí nějaká chytřejší logika pro napovídání na základě již dříve vytvořených pravidel. Absence návrhů klasifikace má pak za následek nekonzistenci v knihovně produktů. Samozřejmě vždy zde bude hrát roli lidský faktor, i kdyby byl klasifikátor sebevíc přesný, tak vždy budou bohužel vznikat chyby, proto uživatel musí ověřovat vygenerovaná pravidla.

Co se týče samotného vytváření pravidel, tak bylo vypozorováno z již vytvořených pravidel, že jsou opakovány určité vzory. Tyto vzory naznačují, že by mohl být proces částečně automatizován a uživatel by ideálně pouze schvaloval nově vygenerovaná pravidla.

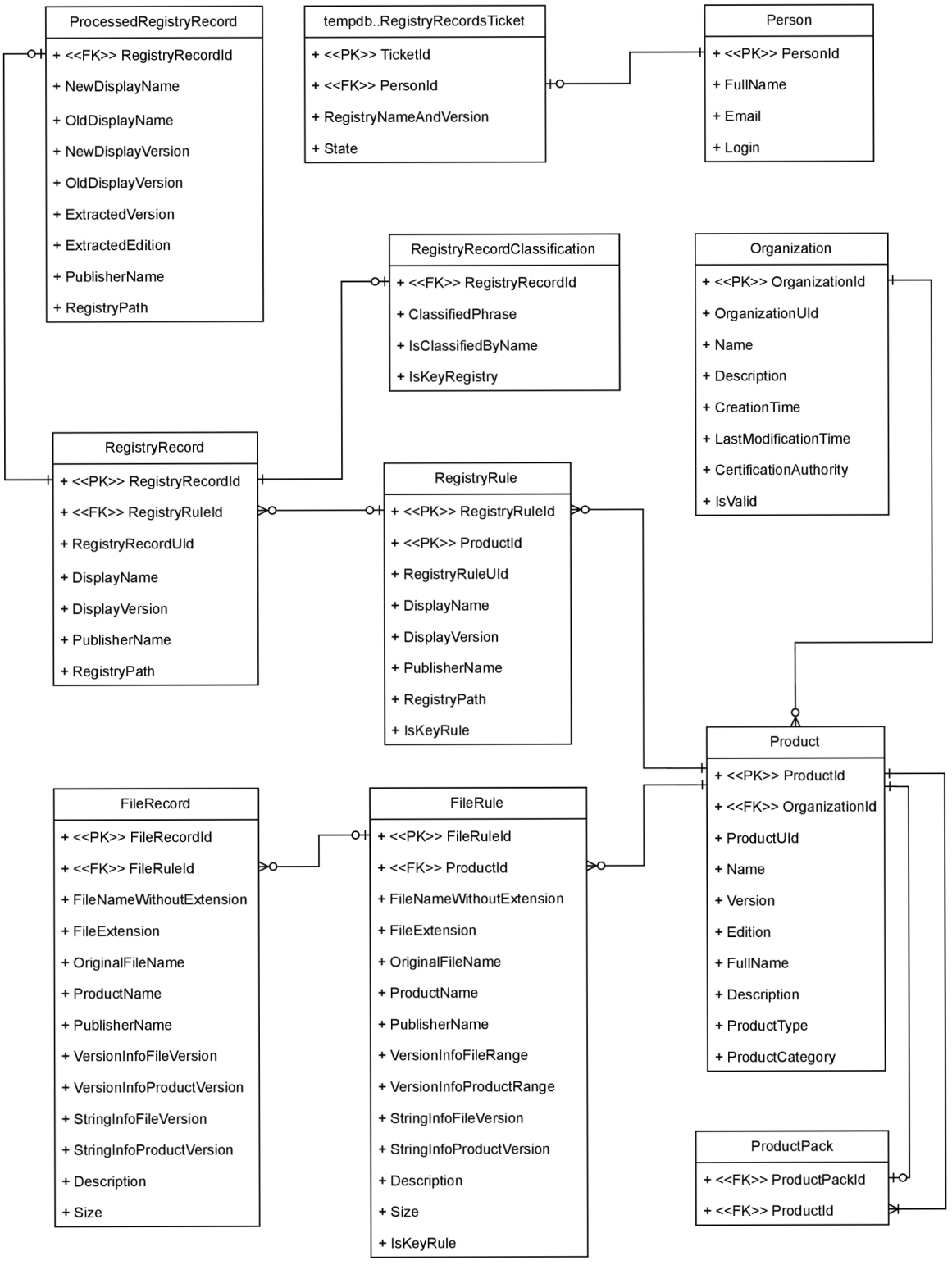
V Service Desk jsou správci adresovány požadavky, jinak také nazývané jako tikety, týkající se knihovny produktů. Příchozí tikety vyřizuje prioritně a závislosti na typu požadavku vykonává určité akce v aplikaci.

3.2 Datové schéma

Vzhledem k návaznostem většího množství modulů na databázi nemohla být hlavní část databázového schématu nijak dotčena tak, aby nedošlo k problémům s integrací při nasazení aplikace. Nová aplikace však měla za úkol zpřehlednit, které části jsou důležité a které nikoliv. Do modelů dat byly načítány pouze potřebné informace a ostatní data byla přehledně oddělena. Sloupce byly navíc náležitě přejmenovány a načítány tzv. aliasy. Tato skutečnost napomůže v dalším vývoji při změně databázového schématu.

V souvislosti s databázovým schématem byl předveden představitelům firmy Alvao navržený formát pravidel popsaný v sekci 2.3. Změna formátu pravidel a produktů knihovny, ačkoliv byla potencionálně přínosná, nemohla být realizována, protože by znamenala příliš velký zásah musely se měnit všechny moduly závislé na knihovně softwaru.

Na obrázku 3.1 je uvedeno datové schéma, na kterém jsou vyznačeny vybrané tabulky používané v aplikaci. Nově přidané tabulky jsou *ProcessedRegistryRecord*, *RegistryRecordClassification* a *RegistryRecordsTicket*. Tabulka *ProcessedRegistryRecord* obsahuje zpracované registrové záznamy, tedy extrahovaná data z názvů těchto záznamů a návrhy nových pravidel. V tabulce *RegistryRecordClassification* se nachází klasifikované registrové záznamy, kde je zaznamenán výstup klasifikace. Dočasná perzistentní tabulka *RegistryRecordsTicket* je vytvořena pro uchování vstupů z formulářů zvlášť pro každého uživatele.



Obrázek 3.1: Datové schéma

3.3 Nároky na novou aplikaci

S uživateli byla provedena v začátcích práce analýza požadavků na aplikaci a průzkum, při kterém byly zjištěny nejčastěji opakované úkony a nedostatky v aplikaci. Na základě tohoto průzkumu byly poté navrženy změny. Hlavním požadavkem bylo zpracování registrových záznamů pro navrhování pravidel, které se dělalo manuálně, a klasifikace těchto záznamů na klíčové a neklíčové.

3.3.1 Volba architektury

Ze zadání práce ani z požadavků firmy Alvao nebylo pevně určeno jaký objektově relační mapovací (ORM) framework má být použit, proto byl nejdříve vyzkoušen pro práci s relačními databázemi Entity Framework. Ten se však ukázal jako méně vyhovující pro přístup založený na databázi, tzv. database first approach, který spočívá v tom, že nejprve se vytvoří databáze a poté se vytváří modely dat na základě struktury a schématu databáze. Pro tento případ je výhodnější použít Micro ORM Dapper, který sice obsahuje minimální počet funkcí, ale za to poskytuje větší flexibilitu v tom, jak přistupovat k databázi. Umožňuje přímé psaní SQL dotazů, což bývá užitečné v situacích, kdy je potřeba provádět složitější operace nebo kdy je třeba optimalizovat pro výkon. [15]

Klíčovou součástí moderního vývoje webových aplikací je tzv. *client-side scripting*. Vykreslování na klientské straně umožňuje dynamicky měnit obsah stránky bez nutnosti kompletního načítání nové stránky ze serveru. Webová aplikace tak může být interaktivní a více uživatelsky přívětivá. Pro tyto potřeby byla vybrána knihovna *jQuery*. Jedná se o jednu z nejrozšířenějších knihoven jazyka JavaScript, která je velmi dobře dokumentovaná knihovna a má velkou komunitou vývojářů. Knihovna výrazně zjednodušuje vývoj webových stránek tím, že nabízí jednoduché a intuitivní API pro manipulaci s prvky *DOM* (Document Object Model). Umožňuje také jednoduše provádět asynchronní volání *AJAX* (Asynchronous JavaScript and XML) na webový server pro načtení nebo odeslání dat.

3.3.2 Návrhové vzory

V projektech pracujících s databází se použije návrhový vzor úložiště (anglicky repository pattern), který slouží k vytvoření abstrakce mezi vrstvou přístupu k datům a vrstvou obchodní logiky aplikace, aby bylo možné lépe oddělit vrstvy a snížit závislosti mezi nimi. Tento návrhový vzor poskytuje jednotné rozhraní *IRepository* pro přístup ke zdrojům dat, přičemž skrývá detaily implementace databáze. To usnadňuje automatizované testování a úpravy aplikace, protože změny v úložišti se promítnou pouze v této vrstvě a neovlivní business logiku aplikace. Vzor úložiště přispívá ke zlepšení organizace kódu a následné údržby aplikace [14].

Při implementaci webové aplikace se použije architektonický vzor *Model-View-Controller* (MVC), který slouží pro oddělení prezentační vrstvy od zpracování dat a obchodní logiky. Konkrétně se jedná o rozdělení aplikace do tří hlavních částí model, pohled a řadič. Komponenta model odpovídá veškeré logice souvisejí s daty, včetně dat souvisejících s obchodní logikou. Pohled je zodpovědný za prezentaci obsahu prostřednictvím uživatelského rozhraní. Řadič funguje jako rozhraní mezi komponentami model a pohled. Zpracovává příchozí požadavky uživatelů, které jsou na řadič směrovány. Interaguje s modelem za účelem provádění uživatelských akcí nebo získávání dat a poté vybírá správné zobrazení, které se má zobrazit, a poskytne mu potřebná data z modelu [11].

Pro všechny modely dat se použije datový typ záznam podle [12], který je zaveden od verze jazyka c# 9.0. U tohoto datového typu jsou kompilátorem syntetizovány metody pro zobrazení záznamů pomocí *Object.ToString()* a dále také metody *Object.Equals()* pro porovnávání záznamů na základě hodnot z veřejně vystavených polí. Použití tohoto typu je užitečné pro pozdější vývoj a ladění aplikace.

Kapitola 4

Implementace

Tato kapitola se zaměřuje na technické detaily implementace a popisuje, jak byly realizovány naplánované funkce aplikace. Zahrnuje také informace o rozvržení aplikace do logických celků. Cílem této kapitoly je poskytnout čtenáři komplexní pohled na samotný vývoj aplikace a umožnit mu pochopit, jak byl výsledný produkt vytvořen.

4.1 Zpracování registrových záznamů

V pravidlech se objevují opakující vzory a je tedy se mohou taková pravidla vygenerovat. Výsledkem je zjednodušení a alespoň částečná automatizace procesu vytváření pravidel. Zpracovávají se především názvy registrových záznamů, jelikož obsahují největší variabilitu a dají se rozdělit do několika logických částí viz 2.3.

Ke zpracování registrových záznamů dochází pouze při načtení nových nerozpoznaných záznamů. Pro potřebu uchování zpracovaných dat o nerozpoznaných záznamů byla vytvořena nová přechodná tabulka pojmenovaná „ProcessedRegistryRecord“, jejíž obsah je k nalezení na 3.1. Přechodnou tabulkou je zajištěno, že nemusí při každém načítání seznamu proběhnout časově náročný dotaz na získání nerozpoznaných záznamů. Obsahuje sloupce *Id* totožné z původní tabulky se všemi registrovými záznamy, *OldDisplayName* a *NewDisplayVersion* jsou nezměněné hodnoty, *NewDisplayName* a *NewDisplayVersion* jsou zpracované hodnoty, které slouží pro vygenerovaní pravidla. Ukládají se také extrahované informace z názvu o verzi a edici do sloupců *ExtractedVersion* a *ExtractedEdition*.

Rozložení názvu probíhá pomocí regulárních výrazů z knihovny *RegularExpressions* ve standardním jmenném prostředí *System.Text*. Regulární výrazy byly separovány do samotného souboru pojmenovaného *ParserRegexes*. Jsou vytvářeny konstruktorem vždy s příznakem *RegexOptions.IgnoreCase*, aby se neuvažovala velká a malá písmena při hledání určitých výskytů. Dále kvůli optimalizaci je použit příznak *RegexOptions.Compiled*, který značně urychlil zpracování, protože regulární výrazy jsou spouštěny nad větším množstvím záznamů. Pro lepší čitelnost komplikovanějších regulárních výrazů je také použit příznak *RegexOptions.IgnorePatternWhitespace*, ten umožňuje připsat k určitým částem komentáře. To je výhodné především pro další vývoj aplikace, kdy se budou regulárních výrazy blíže specifikovat podle nových kombinací, které se vyskytnou.

4.1.1 Zpracování verzí

Z názvu jsou odebírány různé nepodstatné části, například bývá odstraňována verze z zobrazovaného názvu *DisplayName*, pokud je přítomná zobrazovaná verze. Odstranění probíhá náhradou celé verze v názvu za %.

Pokud u registrového záznamu zobrazovaná verze není přítomná, tak v názvu má zůstat pouze hlavní část licence (anglicky major licence), např. z názvu „CifX Device Driver 1.3.0.0“ má vzniknout „CifX Device Driver 1.%“.

Pokud zobrazovaná verze je přítomná, ale v názvu se nachází verze, a navíc uprostřed názvu, tak nemá být celá odstraněna náhradou za %, ale opět pouze zachována hlavní část licence, např. u názvu „BurnInTest v9.0 Standard“ má vzniknout pravidlo „BurnInTest v9.% Standard“.

Pro generování pravidel jsou v názvu dále za % přímo nahrazovány části jako cílová architektura, data, jazyk a číslo KB u produktů společnosti Microsoft.

4.1.2 Extrakce edicí

Pro extrahování edicí bylo nutné nejdříve zjistit, jaké jsou nejpoužívanější edice. U produktů byla provedena analýza edicí a vybrány takové edice, které již byly použity v minulosti více než 5x. Následně byl tento seznam edic nahrán do listu v jazyce c# tak, aby seznam edic dal snadno upravovat, přidávat nové či odebírat nechtěné edice. Extrakce edicí mohla být prováděna různými způsoby, hledáním v názvech podřetězce, který by odpovídal některé z vybraných edic, nebo opět regulárním výrazem.

K extrakci byl použit regulární výraz, protože bylo možné jednodušeji, a hlavně přehledněji specifikovat podmínky. Mezi edicemi se totiž nacházely i takové, které nebyly jednoslovné. Také bylo nutné, aby edice nezasahovala do některé ze slov názvu a před edicí byl začátek názvu nebo mezera a za edicí byla mezera nebo konec názvu. Regulární výraz je vždy vytvořen při spuštění programu, dojde ke zkombinování edicí, nahrazení bílých znaků za `\s` a vytvoření regulárního výrazu.

Při testování byly vyhledány nejčastěji extrahované edice z názvů registrových záznamů a nalezeny edice jako „Windows“, „Update“, „Server“, „Client“ a další, které není vhodné extrahovat. Seznam extrahovaných edic byl následně očištěn o tyto nežádoucí edice.

4.1.3 Testování analyzátoru

Pro testování a validaci zpracovaných registrových záznamů byl založen projekt *RegistryRecordParser* typu konzolová aplikace, na místo typu knihovna tříd. Spuštěním programu dojde ke zpracování všech registrových záznamů a unikátní záznamy ukládány do tabulky. Tento proces umožňuje rychlé a jednoduché ladění, které se zejména využije při úpravách regulárních výrazů v pozdějších fázích životního cyklu aplikace.

V tomto testovacím módu se ukládají pouze unikátní registrové záznamy, protože je důležité analyzovat výsledky zpracování na různých záznamech a ty stejné nejsou v tento okamžik podstatné. Kvůli rychlosti jsou tyto záznamy při zpracovávání ukládány do struktury *HashSet*. Ta má vlastnosti hašovací tabulky, a tedy vyhledání v této struktuře, pokud nedochází ke kolizím, tak probíhá v průměru v konstantním čase. Velice důležitou částí při používání generických kolekcí, struktur je přepsání definic metod *GetHashCode()* a *Equals()*. Metoda *Equals()* se používá ke kontrole, zda jsou si dvě instance daného typu rovné. Metoda *GetHashCode()* se používá ke generování *hash* hodnoty pro danou instanci. Přepsáním definic těchto metod poskytneme novou implementaci metodám zděděných ze základní třídy

a je možné pak používat metodu *Add* struktury *HashSet* pro přidání objektu *ProcessedRegistry*, která interně používá výše uvedené redefinované metody. [13]

Při porovnávání záznamů se nezohledňují velká a malá písmena proto byl u definice metody *GetHashCode()* použit *StringComparer.OrdinalIgnoreCase* a u metody *Equals()* *StringComparison.OrdinalIgnoreCase*. Kvůli jednotnosti v projektech byl pro model *ProcessedRegistry* použit datový typ záznam, který v základě implementuje tyto metody a jejich redefinice není možná. Bylo tedy nutné vytvořit třídu *ProcessedRegistryComparer* odvozenou od rozhraní „IEqualityComparer<ProcessedRegistry>“, v níž byly definovány výše uvedené metody. Struktura *HashSet* následně byla vytvořena konstruktorem s parametrem *ProcessedRegistryComparer*.

4.2 Klasifikátor produktů

Při vytváření rozpoznávacích pravidel je nutné rozhodnout, zda je produkt klíčový nebo neklíčový a v závislosti na této informaci, jaký typ pravidla se má vytvořit. Pro usnadnění určování, do které kategorie produkt má být zařazen, byl založen nový projekt pojmenovaný „ProductClassificator“, který provádí klasifikaci registrových záznamů.

Činnost tohoto programu se dá rozdělit do tří částí. Nejprve dojde k očištění názvů registrových záznamů. Dále jsou pro jednotlivá slova zjištěny četnosti výskytů v klíčových a neklíčových produktech. Nakonec proběhne klasifikace pomocí vybrané klasifikační metody.

Pomocí analyzátoru registrových záznamů 4.1 jsou zobrazované názvy očištěny o nežádoucí informace, které nemají být použity pro klasifikaci. Z názvů jsou odstraňována čísla verzí, cílová architektura, jazyky a data vydání. Speciální nealfanumerické znaky, mezi které patří podtržítka, uvozovky, pomlčky a další, jsou nahrazeny za mezeru a následně odstraněny možné nově vzniklé duplicitní mezery, čímž se zabrání vzniku nekonzistencí v názvech.

Takto očištěné názvy registrových záznamů se postupně prochází, rozdělují se na jednotlivá slova a vkládají se do slovníku, kde se navíc zaznamenává počet výskytů. Slovník je typu *Dictionary<string, WordOccurrence>*, kde klíč je slovo z názvu a hodnota je objekt obsahující počty výskytů v klíčovém a zvlášť v neklíčovém záznamu. Název je také rozdělen na dvojslova a proveden stejný postup. Pro názvy výrobců z registrových záznamů je vytvořen samostatný slovník a opět zjišťovány četnosti výskytů v jednotlivých kategoriích, s rozdílem, že název výrobce není rozkládán na samostatná slova.

Zpracovaná slova názvů včetně počtů výskytů v kategoriích jsou následně uložena do tabulky *ClassificatorRegName* a názvy výrobců do tabulky *ClassificatorRegPublisher*. K uložení všech záznamů se používá hromadné ukládání (anglicky bulk insert), které ukládá záznamy po dávkách o nastavené velikosti. Ukládání dávek probíhá transakčně a v případě chyby dochází k navrácení všech změn provedených v rámci transakce. Hromadné ukládání má za výsledek, že není na SQL server posíláno příliš velké množství dotazů jako při jednotlivém vkládání záznamů, a je dosaženo výrazně vyšší přenosové rychlosti a rychlejšího průběh celého programu.

Pro klasifikaci registrových záznamů se načítá obsah tabulek *ClassificatorRegName* a *ClassificatorRegPublisher* zpět do slovníků typu *Dictionary<string, WordOccurrence>* a následně probíhá klasifikace vybranou metodou.

První klasifikační metoda rozkládá název registrového záznamu na jednotlivá slova a dvojslova. Taková spojení poté zkouší získat ze slovníku, pokud slovník spojení neobsahuje, pokračuje se dále v rozpoznávání registru dalším spojením. V opačném případě se vypočítá pravděpodobnost, s jakou přísluší spojení do dané kategorie, a při pravděpodob-

nosti vyšší, než nastavená prahová hodnota je záznam klasifikován. Stejným způsobem se postupuje při klasifikaci pomocí názvu výrobce.

U Multinomiální Naivní Bayesovy klasifikační metody je nejdříve zjištěna *apriorní* pravděpodobnost jako poměr klíčových a neklíčových záznamů z celkového počtu záznamů, přičemž tyto počty jsou načteny z databáze. Apriorní pravděpodobnost klíčového záznamu je přibližně hodnota 0.37 a pravděpodobnost neklíčového záznamu je doplněk, tedy 0.63. Tyto počáteční odhady pravděpodobnosti jsou následně promítány do klasifikace, protože je tato hodnota nahrána do celkové pravděpodobnosti klíčového a neklíčového záznamu. Při klasifikaci registrových záznamů je rozkládán název na jednotlivá slova, ta se opět pokouší získat ze slovníku. V případě, že slovo se nevyskytuje ve slovníku, tak se předpokládá, že se jedná o zcela nový produkt, a není na základě tohoto slova rozpoznáváno. V opačném případě se vypočítají pravděpodobnosti, s jakými slovo patří do klíčového a neklíčového záznamu, kde je navíc připočten pseudo výskyt $\alpha = 1$, jak je znázorněno na následující rovnici (4.1).

$$P(\text{"}slovo\text{"}|K) = \frac{|K| + 1}{|K| + |N| + 1} \tag{4.1}$$

Celková pravděpodobnost příslušnosti záznamu, zda je klíčový a neklíčový, je vypočtena jako součin všech získaných pravděpodobností slov z názvu registrového záznamu a názvu výrobce. Následně jsou výsledné pravděpodobnosti porovnány a zjištěn výsledek klasifikace.

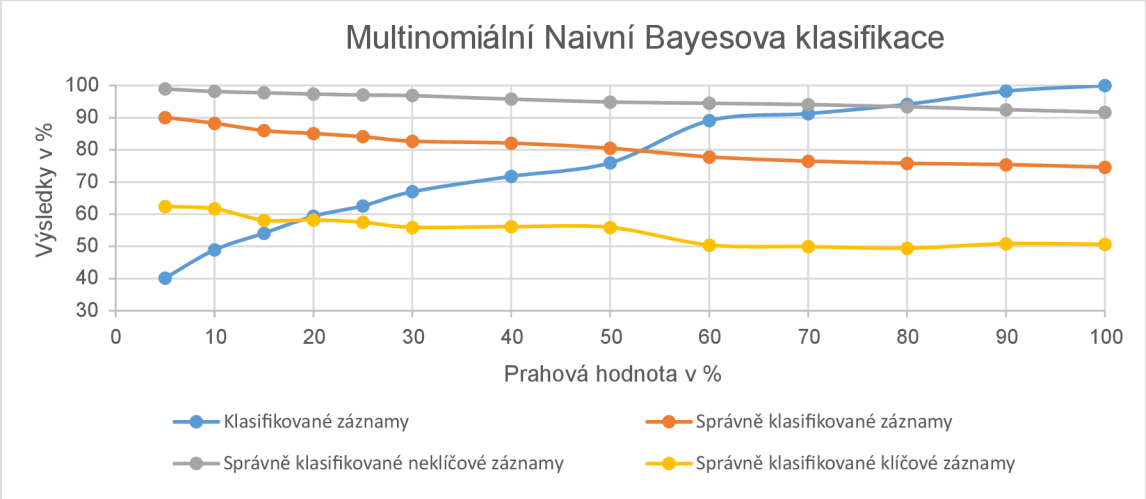
Z projektu *ProductClassificator* je vystavena funkce *ClassifyUnrecognizedRegs*, která volá metody pro zpracování nových nerozpoznaných záznamů a provádí klasifikaci. Výsledek následně je ukládán do tabulek *ProcessedRegistryRecord* a *RegistryRecordClassification*. Tato funkce je poté volána v jiném projektu, který provádí sběr nerozpoznaných záznamů ze zařízení od zákazníků firmy Alvao.

4.2.1 Testování klasifikačních metod

Pro otestování přesnosti zmíněných klasifikačních metod 2.5 byly registrové záznamy rozděleny na trénovací a testovací sady v poměru 80/20. Záznamy byly rozděleny náhodně, aby bylo dosaženo nezávislosti testů. Množina obsahující trénovací data byla zpracována a byly zjištěny četnosti výskytů slov. Poté proběhla klasifikace záznamů z množiny testovacích dat. Klasifikované záznamy byly porovnány se skutečnými klasifikacemi záznamů a zhodnoceny výsledky metod.

Výsledky klasifikace se lišily podle nastavené prahové hodnoty. Na následujícím grafu 4.1 lze vidět naměřené výsledky klasifikací pomocí Multinomiálního Naivního Bayesova klasifikátoru pro různé prahové hodnoty. Tyto hodnoty určují minimální povolený podíl celkové klíčové a neklíčové pravděpodobnosti, při které lze záznam klasifikovat. Podíl pravděpodobností je zde prováděn za účelem dosažení lepších výsledků klasifikace. Při nízkých prahových hodnotách je dosažena vyšší přesnost klasifikace, avšak na úrok nižšího počtu klasifikovaných záznamů z celkového počtu. Při zvyšování prahové hodnoty dochází k vyššímu zastoupení klasifikovaných záznamů, ale zároveň tak k nižším přesnostem klasifikace.

Jak si můžete povšimnout metoda klasifikuje velice přesně neklíčové záznamy. Tento jev je vysvětlen vysokým opakováním určitých slov, která jsou čistě specifická pro tuto kategorii. Zatímco u klíčových záznamů jsou názvy více proměnlivé.

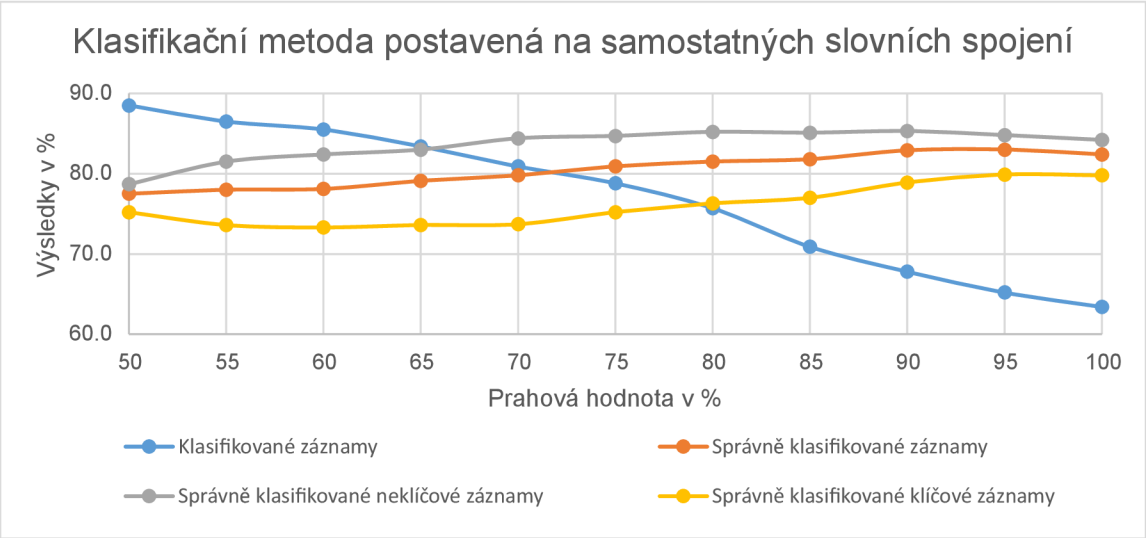


Obrázek 4.1: Graf s naměřenými hodnotami Multinomiálního Naivního Bayesova klasifikátoru

K tomuto grafu je nutné navíc podotknout, že rozložení klasifikovaných záznamů na klíčové a neklíčové odpovídalo přibližně *apriorní* pravděpodobnosti. Mírná odchylka v řádu maximálně ±3 % je způsobena náhodným rozdělením dat na trénovací a testovací sady.

Při klasifikaci záznamů je zásadní přesnost, která se upřednostňuje nad vyšším počtem klasifikovaných záznamů. Z tohoto důvodu je vybrána a 50 %, protože představuje kompromis mezi kvalitou a kvantitou. Při takto nastavené prahové hodnotě Multinomiální Naivní Bayesův klasifikátor klasifikuje přibližně 75 % záznamů, z nichž správně určených je 82 % celkových klasifikovaných záznamů a 95,5 % neklíčových klasifikovaných záznamů.

Při měření výsledků klasifikace Multinomiálního Naivního Bayesova klasifikátoru byly zároveň měřeny výsledky druhé klasifikační metody, která je založená na klasifikaci samostatných slovních spojení. Naměřené výsledky experimentů s různě nastavenými prahovými hodnotami lze vidět na následujícím grafu 4.2. Zde prahová hodnota není srovnávaná s podílem celkových pravděpodobností, ale s pravděpodobností slovního spojení, zda se jedná o klíčový nebo neklíčový záznam. Z tohoto důvodu je pro vysoké prahové hodnoty nižší zastoupení klasifikovaných záznamů s vyššími přesnostmi klasifikací a obráceně pro nízké prahové hodnoty.



Obrázek 4.2: Graf s naměřenými hodnotami klasifikační metody postavená na samostatných slovních spojení

Výsledky této metody liší oproti metodě Multinomiální Naivní Bayesův klasifikátor v poměrech správně klasifikovaných klíčových a neklíčových záznamů. Klíčové záznamy jsou přesněji určeny, ale za to výrazně nižší přesnosti dosahuje klasifikace neklíčových záznamů.

Právě z důvodu výrazně lepších výsledků klasifikace neklíčových záznamů je vybrán do produkčního prostředí Multinomiální Naivní Bayesův klasifikátor. Na základě této klasifikace jsou odděleny v aplikaci klasifikované klíčové a neklíčové záznamy do různých pohledů. V pohledu seskupujícím klasifikované neklíčové záznamy je přizpůsobeno uživatelské rozhraní, aby bylo možné hromadně zpracovávat vybrané záznamy a usnadnila se tak práce s očišťováním záznamů.

4.3 Vyhledávání produktů

Zcela běžně se stává, že stejný produkt je distribuován pod mírně odlišnými názvy anebo se zaměněným pořadím slov v názvu. Bývá také výhodné najít jiné verze a edice právě zpracovávaného produktu, jelikož může uživateli napomoci rozhodnout, zda je produkt nutné evidovat pro licenční záležitosti. Pro snadné vyhledávání a přiřazování produktů byly implementovány 4 různé metody mezi kterými lze přepínat, ukázáno pro názornost na obrázku 4.3. Tyto metody nabízí větší možnosti při vyhledávání. Uživatel pro nalezení problematického produktu nemusí vyzkoušet tolik kombinací nebo vytvářet nové produkty, čímž by se zanášely do databáze nepřesnosti.

Jako výchozí vyhledávací metoda je nastavená na prosté vyhledávání pomocí podřetězců, to probíhá následujícím způsobem. Na začátek a konec vyhledávané fráze se připojí anglicky tzv. *wildcard* % a následně pomocí operátoru *like* v klazuli *where* se hledají odpovídající názvy produktů. Dochází také k prohledání, zda některý z názvů produktů není podřetězcem vyhledávané fráze, a to kvůli pokrytí případů, kdy je vyhledávaná fráze příliš specifická a delší než název produktu v databázi. Alternativně lze použít vestavěnou funkci *charindex* a testovat zda, je výsledek větší než 0. Exekuční plán dotazu je však stejný a bohužel se při vyhledávání podřetězců nedá žádným způsobem vyvarovat prohledávání

celého indexu neboli použití tzv. *index scan*. V databázi existují řádově pouze desetitisíce produktů, takže rychlost metody nečiní žádný problém, nevýhoda spočívá pouze v nutnosti zadat přesný název produktu nebo zadávat do vyhledávané fráze symboly %, který nahradí 0 nebo více znaků.

Dále z důvodu častého přeuspořádání slov v názvu byla implementována metoda, jež testuje shodná slova vyhledávané fráze se slovy z názvů produktů. Produkty s nejvyšším počtem shodných slov jsou brány jako nejpodobnější a tím pádem jsou seřazeny podle tohoto skóre. Metoda se ukázala velmi prospěšná, protože názvy bývají nekonzistentní, a metoda umožňuje najít i produkty, které by vyhledání pomocí podřetězců nenašlo. Dokáže pokrýt i typický případ, kdy je přidán nebo naopak odebrán název vydavatele na začátek názvu produktu. Krása této metody spočívá v její jednoduchosti, což také znamenalo, že mohla být napsána přímo v sql dotazu viz 4.1.

```
1  SELECT TOP(30) [product].[intProductId] AS [Id]
2      ,[product].[intProductUID] AS [UId]
3      ,[product].[txtName] AS [Name]
4      ,[product].[txtVersion] AS [Version]
5      ,[product].[txtEdition] AS [Edition]
6      ,[product].[FullName] AS [FullName]
7      ,[similarity].[rank] AS [SimilarityRank]
8  FROM [dbo].[tblProduct] [product]
9      CROSS APPLY (SELECT SUM(CASE WHEN CHARINDEX(' ' + [words].[value], ' ' +
           [product2].[FullName]) > 0 THEN 1 ELSE 0 END) [rank]
10         FROM (SELECT [product].[FullName] [FullName]) [product2],
               STRING_SPLIT(@searchedPhrase, ' ') [words]) [similarity]
11 WHERE [similarity].[rank] > 0 AND [product].[bolValid] = 1
12 ORDER BY [SimilarityRank] DESC, LEN([product].[FullName]), [product].[FullName]
```

Ukázka kódu 4.1: Sql dotaz pro vyhledávání produktů metodou shodných slov

4.3.1 Využití podobnostních metrik

V názvech produktů se sice žádné chyby nenachází, ale při zadávání vyhledávací fráze může dojít k překlepu, proto byla vytvořena metoda, která používá podobnostní metriky. Po vzoru [10] byla použita metrika zvaná *Sorensen–Dice* koeficient nad slovy, délka nejdelšího společného podřetězce, a navíc metrika uvažující editační vzdálenost, metrika *Damerau-Levenshtein*. Pro dobré výsledky byla zkombinována metoda *Sorensen–Dice* s metrikou *Damerau-Levenshtein* a slova se tak považovala za stejná při editační vzdálenosti menší než 3, kdy se vychází z prahové hodnoty a postupu uvedeném v [9] v 5. kapitole. Podobnost vypočtená metrikami byla sečtena a následně vypočítán aritmetický průměr, čímž byla vypočítána celková podobnost. K nalezení podobnosti mezi 2 názvy byly použity metriky z *nuget* balíčku *F23.StringSimilarity*, které bylo potřeba modifikovat tak, aby pracovaly nad slovy.

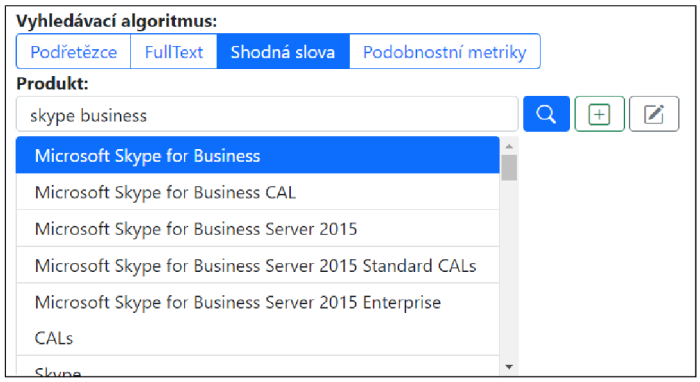
Pro praktické použití těchto podobnostních metrik byly otestovány implementace metrik jako sql funkcí, ale jak se ukázalo byly natolik pomalé, že je nebylo možné používat pro vyhledávání. Funkce byly pomalé, protože sql není na takové výpočty s alokací paměti přímo určený. Pro urychlení se nabízelo načíst do paměti všechny produkty a provádět výpočet v paměti, ale tento způsob by byl paměťově náročný a složitý na správu při *CRUD* operacích nad produkty. Nakonec byl zvolen způsob nasazení databázových objektů *Common Language Runtime* (CLR), čímž je možné používat funkce napsané v jazyce c# přímo v sql. Je tedy možné volat funkci *dbo.CalculateSimilarity(@name1 nvarchar(max), @name2 nvar-*

char(max)) vracející hodnotu podobnosti 2 názvů. V sql byla vytvořena funkce, jejíž obsah je vázán příkazem *external name* na jméno metody v jazyce c#. Pro nasazení databázových objektů byl vytvořen nový projekt typu knihovna tříd. Z tohoto projektu byla následně vystavena metoda pomocí atributu *[SqlFunction(DataAccess = DataAccessKind.None, IsDeterministic = true)]*.

Aby byla zajištěna bezpečnost používání této knihovny tříd, zkompilovanému souboru s příponou .dll, tak byl ze souboru vytvořen hash algoritmem SHA-512 a následně přidán v sql jako důvěryhodné *assembly*. Soubor .dll tak nemůže být přepsán a zneužit.

Z tohoto způsobu použití plyne také mnoho výhod, metoda je rychlá, nenáročná na paměť a z hlediska dalšího vývoje je možné použité metriky jednoduše zaměnit za jiné. Tím je umožněno jednodušší otestování jiných podobnostních metrik nebo jejich kombinací pro nalezení přesnější anebo rychlejší metody.

Byly také otestovány vestavěné funkce v SQL Serveru *DIFFERENCE* a *SOUNDEX*, tyto funkce však nedosahovaly přijatelných výsledků pro vyhledávání podobných produktů. *Full-text*ové vyhledávání s použitím *CONTAINSTABLE* a *ISABOUT* dosahovalo téměř stejných výsledků jako vyhledávání metoda shodných slov 4.1, ale u *full-text*ového vyhledávání je algoritmus pro určování podobnosti tzv. rank zcela interní a nelze jej upravit. U metody nebylo chtěné započítávání, zvyšování skóre podobnosti při nalezení stejného slova vícekrát. Při zadání klíčového slova „business“ měl příliš velkou podobnost název obsahující toto slovo dvakrát, což nebylo žádoucí. Hodnotící systém lze ovlivnit pouze přidáním vyšší váhy pro slova, která se nacházejí blíže začátku vyhledávané fráze. Nesporná výhoda tohoto *full-text*ového vyhledávání však spočívá v rychlosti, protože nad slovy a prefixy slov jsou vytvořené indexy a nemusí se procházet všechny řádky tabulky, čímž se složitost snížila z lineární $\mathcal{O}(n)$ na konstantní $\mathcal{O}(1)$.



Obrázek 4.3: Vyhledávač s přepínačem vyhledávacích metod

4.4 Webová aplikace

Při vývoji webové aplikace založené na frameworku *ASP.NET Core MVC* byla použita kniha [2], ze které byly čerpány nejlepší praktiky. Mezi které patří například znovu-používaní částí kódu pomocí tzv. částečných zobrazení (anglicky partial views). Částečná zobrazení umožňují vyhnout se kopírování a samostatné údržby stejných částí kódu. Ty jsou poté vykresleny na specifikované místo na stránce. Tímto způsobem je možné rozdělit do logických celků.

4.4.1 Registrová pravidla

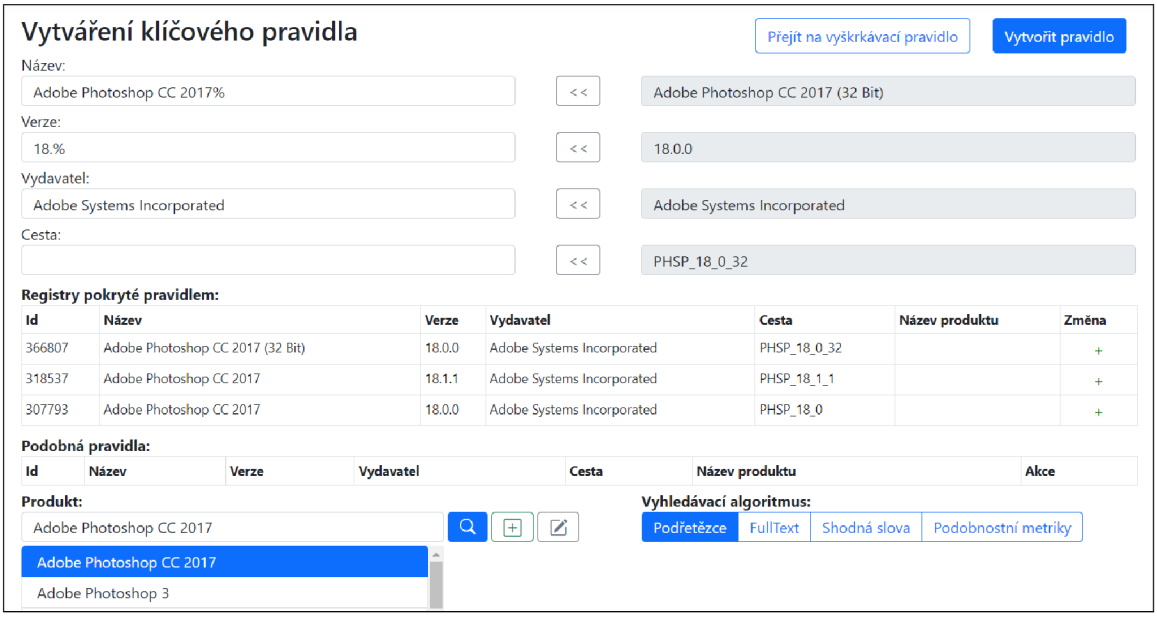
Správce knihovny zpracovává vždy všechny registrové záznamy a je jich nemalé množství. Z tohoto důvodu jsou registrové záznamy zpracovány za pomocí analyzátoru 4.1 a vygenerována základní pravidla, která napomáhají s odstraňováním nejčastěji se objevujících nechtěných částí pravidla.

Registrové záznamy se očišťují o neklíčové záznamy tzv. *vyškrkávacími* pravidly. Tato pravidla nepřiřazují žádný produkt k záznamům, a navíc bylo vypozorováno, že pro drtivou většinu záznamů jsou navržená, vygenerovaná pravidla správná a není třeba dalších modifikací. Z tohoto důvodu bylo přizpůsobeno vytváření těchto pravidel a v tabulce s registrovými záznamy jsou zobrazována navržená pravidla. Pro každý záznam jsou v prvním řádku obsažené původní hodnoty a v druhém řádku jsou zpracované hodnoty s navrženým pravidlem, přičemž jsou zde uvedené pouze hodnoty pro sloupce název a verze. Pokud je název vydavatele přítomný, tak se v pravidle uvádí bez jakýchkoliv úprav a cesta registrového záznamu u toho typu záznamů není podstatná, proto se neuvádí. Navržená pravidla v uvedeném znění lze vytvářet při kliknutí na žluté tlačítko s titulkem „Automaticky vytvořit vyškrkávací pravidlo“. Kliknutím na toto tlačítko se vyvolá *AJAX* požadavek s Id záznamu, pro které má být vytvořeno pravidlo. Vytvořením pravidla se pokryje jeden nebo více registrových záznamů, Id hodnoty pokrytých záznamů jsou poslány v odpovědi a tyto řádky pomocí jazyka *Javacript* smazány z tabulky. Pro odlišení klasifikovaných neklíčových záznamů v této hlavní tabulce jsou navíc záznamy obarveny světle žlutou barvou.

Registrová pravidla lze také vytvářet v samostatné pohledu, jak je ukázáno na následujícím obrázku 4.4. Zde můžete v levé horní části vidět vstupní pole, do kterých jsou načítány hodnoty navrženého, předem vygenerovaného pravidla. Zatímco v pravé horní části jsou načteny původní hodnoty registrového záznamu. Kliknutím na šipky mezi těmito vstupními poli lze přehrát novou hodnotu tou původní, tím dojde ke změně směru šipky a při opětovném kliknutí na tlačítko se obsah vstupního pole vymaže. Tyto akce lze neomezeně opakovat a uživatel tak nemusí kopírovat a vkládat původní hodnoty do vstupních polí nebo je z nich mazat při ladění pravidla.

Ladění pravidla je ukázáno v části pod těmito vstupními poli. V tabulce s nadpisem registry pokryté pravidlem se nacházejí registrové záznamy, které jsou pokryty právě vytvářeným pravidlem. Dále také v tabulce nazvané „Podobná pravidla“, v níž se nachází všechna pravidla, která pokrývají registrové záznamy ve výše uvedené tabulce registry pokryté pravidlem. Pokud tedy dojde ke konfliktu, je možné přejít na existující pravidlo a upravit ho tak, aby pokrývalo i nový nerozpoznaný záznam. Případě lze takto i odhalit redundance pravidel a redundantní podobné pravidlo smazat. Před smazáním pravidla je ukázáno potvrzovací modální okno, ve kterém je uživatel varován, že se jedná o nevratnou akci a zda si ji opravdu přeje provést.

Při změně znění pravidla jsou obě tyto tabulky aktualizovány společně. Ke změnám pravidla dochází dvěma způsoby. Prvním z nich je kliknutí na tlačítko pro nahrání nebo smazání hodnoty, což po vykonání funkcionality zavolá asynchronní funkce pro aktualizování obou tabulek. Druhým způsobem, jak může dojít ke změně právě vytvářeného pravidla, je zápis do vstupních polí. U tohoto způsobu je dbáno, aby se negenerovalo zbytečně velké množství dotazů, především protože aplikace bude provozována v cloudovém prostředí Azure App Service. Na vstupní pole je přidán posluchač události (anglicky event listener), který nastaví při vstupu odpočet časového limitu, po jehož vypršení dojde aktualizování obou tabulek. Tento časový limit je nastaven na 500 ms a je obnoven při každé další změně tak, aby byly tabulky aktualizovány se zpožděním, teprve až při ukončení psaní.



Obrázek 4.4: Formulář pro vytvoření registrového pravidla

4.4.2 Modální okno pro vytvoření produktu

U vyhledávače produktů zobrazeném ve spodní části obrázku 4.4 lze kliknout na zelené tlačítko, čímž zobrazí modální okno. V tomto modálním okně je možné vytvořit nový produkt bez jakéhokoliv přechodu. Tímto způsobem se nezatěžuje serverová část a zároveň je také dosaženo lepšího uživatelský zážitku. Na obrázku 4.5 je vyobrazeno modální okno. Do vstupních polí je načtena extrahovaná edice z názvu registrového pravidla a verze, buď extrahovaná z názvu nebo hlavní verze ze zobrazované verze. Za modálním oknem lze navíc vidět jednotlivé položky registrového záznamu, název, verzi, vydavatele a cestu, pro případné opravy nově vytvářeného produktu nebo vyhledávání, přiřazování a vytváření organizace.

Do vstupního pole pojmenovaného „organizace“ je načten název vydavatele. Toto vstupní pole slouží pro vyhledávání organizací, přičemž je použita stejná komponenta jako pro vyhledávání produktů. Modální okno je vykresleno synchronně při načítání stránky „Vytváření klíčového pravidla“, ale vyhledávání organizace neboli vydavatele zadaného do vstupního pole proběhne až po prvním rozkliknutí tohoto modálního okna. Tím je ušetřen jeden dotaz na vyhledání organizace, pokud se přidělí již nějaký existující produkt a není nutné vytvářet nový produkt přes modální okno.

Aktualizování vyhledaných organizací probíhá přesně jako při vyhledávání produktů, tedy se zpožděním až po dokončení zadání vyhledávané fráze. V případě, že je vyhledávaná fráze prázdná nebo neodpovídá žádné organizaci, je vypsán text „Žádná organizace nenalezena“. V opačném případě je vykreslen list nalezených organizací a první organizace je vybrána, což značí modré pozadí položky.

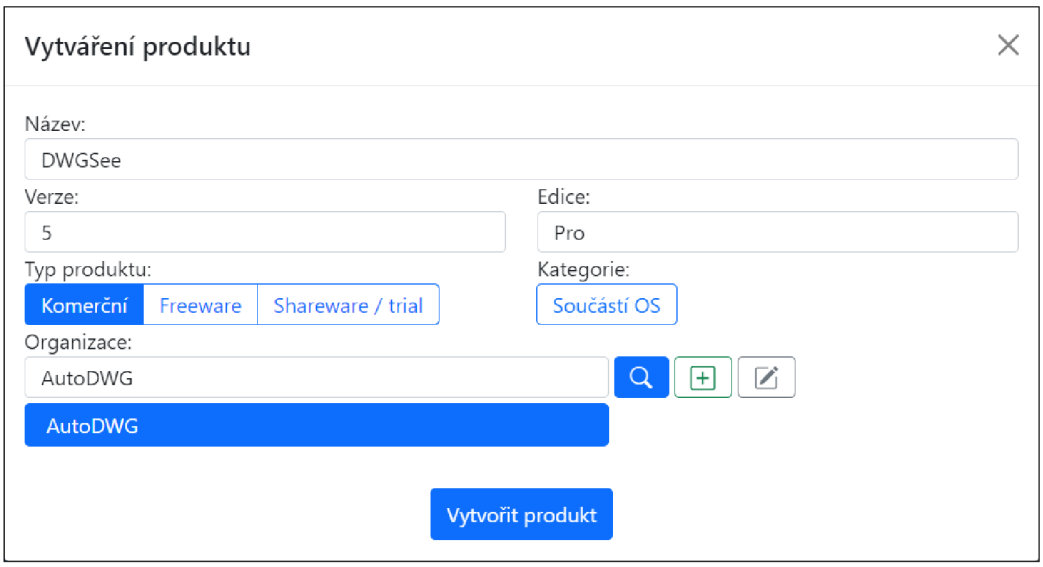
U vytváření organizace bylo vypozorováno z dat, jaká uvádí u organizací, že ve většině případů není potřeba zadávat jiné informace než název. Z tohoto důvodu bylo vytváření organizací integrováno do modálního okna *Vytváření produktů* 4.5. Organizaci lze vytvořit zadáním názvu do vstupního pole organizace a kliknutí na zelené tlačítko nacházející se vedle vyhledání. Po vytvoření organizace je organizace přidána na začátek listu a je

automaticky vybrána. Tímto krokem je opět ušetřeno přecházení mezi různými pohledy a uživateli zpříjemňuje práci v aplikaci.

Vyhledané organizace lze také upravovat, například pro přejmenování existující organizace, přidání popisu nebo vymazání. Pro přechod na úpravu organizace je nutné mít organizaci vybranou a následně kliknout na tlačítko s ikonou tužky, které se nachází vedle vyhledávacího pole.

Pro snadné pochopení, jaké akce jednotlivá tlačítka vykonávají, byly přidány všem tlačítkům atributy „title“ s náležitými popisy.

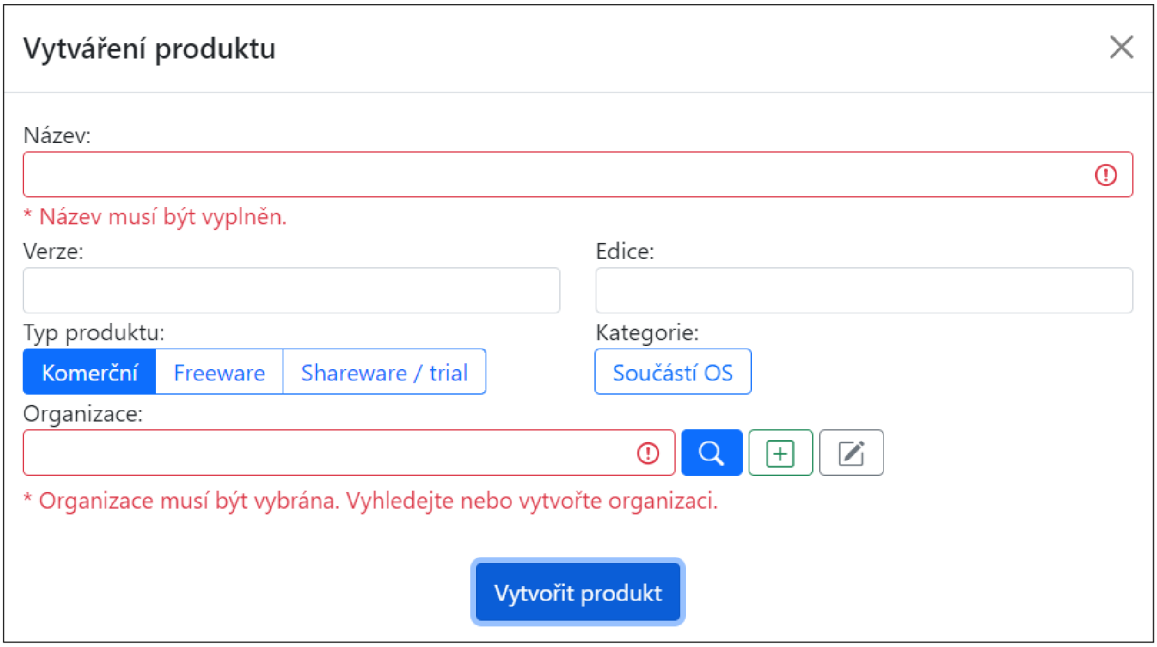
Při vytvoření produktu v modálním okně je poslán na server požadavek, v odpovědi vrací serializovaný objekt nově vytvořeného produktu ve formátu *JSON*. Zaslání celého produktu namísto pouze položky *Id* je nutné kvůli vypočítanému sloupci *FullName*. Nově vytvořený produkt je přidán na první pozici v listu vyhledaných produktů a automaticky označen jako vybraný, a nakonec se pomocí jazyka *Javascript* skryje modální okno.



Obrázek 4.5: Modální okno pro vytvoření produktu

Při zadání špatného vstupu je uživateli zobrazena chybová hláška, tato akce je zpracována na straně klienta. Jsou zde zobrazovány 4 typy chybových hlášek. První dvě z nich nastanou při pokusu o vytvoření produktu bez názvu a vybrané organizace. Další při pokusu o vytvoření organizace bez názvu a poslední při kliknutí na upravit organizaci bez vybrané organizace. Na vstupní pole je navázán posluchač událostí, který při novém vstupu skryje chybovou hlášku.

Příklad chybových hlášek pro formulář „Vytváření produktů“ jsou ukázány na obrázku 4.6. Tímto způsobem jsou řešeny a zobrazovány všechny chyby u povinných sloupců v celé aplikaci.

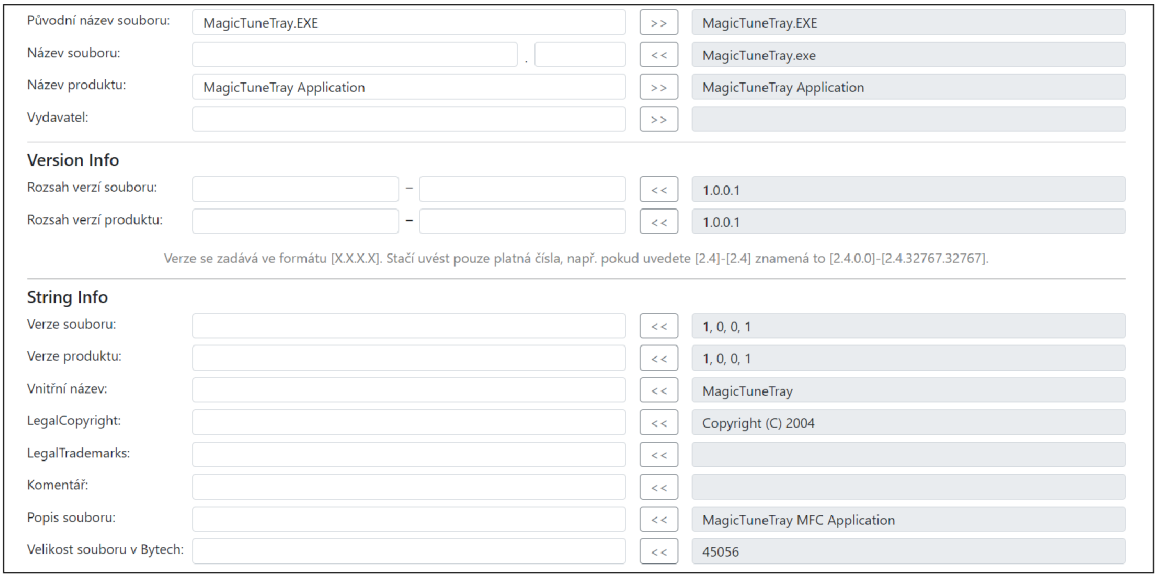


Obrázek 4.6: Zobrazení chybových hlášek na klientské straně

4.4.3 Souborová pravidla

Souborová pravidla se vytváří zřídka, pouze pokud registrový záznam není k dispozici, nebo pokud není zcela kompletní a rozpoznání takového softwarového produktu je problémové. Kvůli nižší frekvenci vytváření souborových pravidel nejsou souborové záznamy zpracovávány a klasifikovány stejně jako ty registrové.

V případě problémového produktu, pro který musí být vytvořeno pro rozpoznání souborové pravidlo, správce knihovny v tabulce se souborovými záznamy vyhledává produkt podle názvu souboru nebo produktu. Záznamů je velké množství proto je potřeba hledaný záznam mnohdy i blíže specifikovat podle části verze nebo výrobce. Souborová pravidla se liší oproti těm registrovým v informacích pomocí nichž se rozpoznávají záznamy. Informace o souborech jsou znázorněny na obrázku níže 4.7.



Obrázek 4.7: Uváděné informace u souborového pravidla

Je zde vyšší množství informací, které se uvádějí, protože souborové záznamy jsou více proměnlivé. Do horní části, viz obrázek 4.7, byly separovány nejčastěji uváděné informace při vytváření pravidel. Dále jsou rozděleny do 2 částí *VersionInfo* a *StringInfo* podle původu, ze kterého zdroje byly načteny. Pomocí tlačítek s šipkou lze nahrát původní hodnoty do vstupních polí.

Při nahrávání původní hodnoty názvu souboru se rozkládá hodnota na název a příponu, tyto dvě položky jsou poté nahrány do samostatných vstupní polí. Při nahrávání rozsahu verzí se rozdělí verze na první dvojčíslí neboli hlavní a vedlejší verzi, a poté dojde k nahrání této hodnoty do obou polí. U rozsahu verzí je kontrolován formát verze pomocí regulárního výrazu na klientské straně, pokud formát neodpovídá, tak je zobrazena chybová hláška „Verze je ve špatném formátu. Správně: [1], [1.0], [1.0.0] nebo [1.0.0.0]“.

Pro detekci některých souborů je nutné uvádět rozsahy verzí, protože například o operační systém Windows 10 se jedná při rozmezí verzí 10.0.0 - 10.0.21999 a pro verze 10.0.22000 a vyšší se jedná o Windows 11. Tento problém nastává u více programů, a proto je významné uvádět tyto rozsahy.

Verze se skládají až ze 4 čísel „major.minor.revision.buildnumber“, z nichž každé může nabývat hodnot od 0 do $2^{16} - 1$, tedy od 0 do 65535. V databázi je verze uložena ve dvou 32bitových číslech *MS* a *LS*, kde *MS* obsahuje *major* a *minor* verzi a *LS* obsahuje *revision* a *buildnumber*. Pro lepší orientaci v datech by bylo vhodnější, kdyby se místo dvou 32bitových sloupců s datovým typem *int*, použily čtyři 16bitové sloupce typu *smallint*. Pro snadné porovnávání čísla verze by naopak byl vhodnější jen jeden 64bitový sloupec typu *bigint*. Bylo by možné mít data uložená ve snadno porovnatelném tvaru (bigint) a přidat pro přehlednost 4 vypočtené sloupce (smallint). Dnes používaná varianta 2 sloupců (int) vedla k tomu, že původní aplikace chybně porovnává čísla verzí, protože porovnává jen samostatně *LS* a *MS*, takže například verze 1.1.5.0 není vyhledána při zadání rozsahu od 1.1.4 do 1.2.2, kdy stará aplikace s tímto rozsahem nenachází žádný záznam. Ve své nové aplikaci jsem se této chyby vyvaroval. Pro snadnější převod filtru uvedeným v řetězci a jeho reprezentací v *LS* a *MS* jsem si napsal funkce.

Soubor ve výjimečných případech neobsahuje záznam v tabulce *VersionInfo* nebo ve *StringInfo*. Tato struktura dat pak vede k používání *LEFT OUTER JOIN*, kde má databázový server menší možnosti optimalizace a nemůže začít vyhledáváním v těchto připojených tabulkách, přitom právě původní název ve *StringInfo* by byl dobře selektivním vyhledávacím parametrem podle indexu. Ve své aplikaci generuji dotaz do databáze s *JOIN* místo *LEFT JOIN*, pokud podmínka vylučuje možnost, že by záznam ve *VersionInfo* nebo *StringInfo* mohl chybět, což vedlo k výraznému zrychlení. Datový model mohl mít relaci nebo kontrolu, která by absenci záznamu ve *VersionInfo* nebo *StringInfo* zabránila, tedy by vždy existoval alespoň prázdný záznam, na který by se odkazovalo, nebo by všechny tyto údaje byly přímo u záznamu souboru.

Z uživatelského hlediska je v datech ještě jedna nepříjemnost, kterou jsou prázdné znaky zleva či zprava v názvech, tyto by mohly být také čištěny již při vstupu. Pokud by to neřešila aplikace vkládající data, tak by to mohl řešit tzv. *instead of trigger* na příslušné tabulce až v databázi.

4.4.4 Produkty a produktové balíky

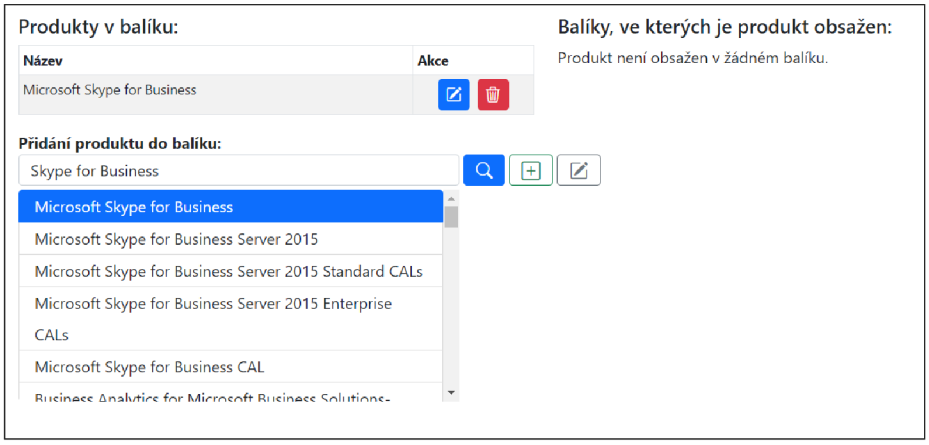
Produkt je možné upravovat, po uložení úprav se zobrazí pohled s detailem produktu, aby si uživatel mohl ověřit, že se změny správně uložily. Produkt lze také vymazat, před vymazáním je zobrazen potvrzovací dialog, potvrzení se zapíše hodnota 0 do sloupce *IsValid*, toto chování je výhodné pro zpětné opravy chyb.

Při editaci produktu lze i spravovat produkty obsažené v balíku nebo také balíky, ve kterých je produkt obsažen. Je zde nutné podotknout, že produkt může být současně i produktovým balíkem, což také vyplývá z návrhu databázového schématu 3.1.

Znovu se používá komponenta *_ProductSearchBar.cshtml* částečného zobrazení pro vyhledání a přidání produktu do balíku. Jednotlivé položky této komponenty obsahují atribut *tabindex* s různými hodnotami, aby se dalo správně navigovat pomocí klávesy *Tab* a kombinace kláves *Tab+Shift*. Po vyhledání produktů lze z vyhledávacího pole přejít pomocí klávesy *Tab* na první položku v listu nalezených produktů, poté v tomto listu procházet pomocí šipek nahoru a dolů, a tím také vybírat produkt. Pokud uživatel nenalezne chtěný produkt, je možné pomocí kláves *Tab+Shift* přejít zpět do vyhledávacího pole a upravit vyhledávací frázi. Po dopsání vyhledávací fráze se obnoví list nalezených produktů a lze v něm opět vybírat. Vyhledávání lze přímo vynutit pomocí stisknutí klávesy *Enter* nebo kliknutím na ikonu s lupou, čímž dojde k bezprostřednímu odeslání formuláře a vyhledání produktů, nemusí se tak čekat na dokončení psaní.

Při použití komponenty, je zde ale rozdílná funkcionalita zeleného tlačítka, které namísto rozbalení modálního okna pro vytváření produktu, slouží pro přidání vybraného produktu do balíku. Probíhá také kontrola, zda již produkt není součástí tohoto balíku. Ověření probíhá na straně klienta na základě porovnání *Id* produktů v balíku s právě přidávaným produktem. Pokud je produkt již součástí balíku, tak je zabráněno přidání produktu a vypsána odpovídající chybová hláška.

Produkty v balíku lze mazat nebo přejít na jejich úpravu, například pro ověření dalších informací o produktu. Akce spjaté s produktovými balíky se přímo aplikují a není potřeba nijak dále ukládat tyto změny. Na obrázku 4.8 je ukázáno uživatelské rozhraní pro správu produktový balíků.



Obrázek 4.8: Uživatelské rozhraní pro správu produktových balíků

4.4.5 Zpracování tiketů

Tiket v Service Desk je typ záznamu o žádosti nebo problému podaným od zákazníka. Každý tiket zahrnuje informace o požadavku, jako je popis problému, kategorie, priorita, stav řešení a kontakt na řešitele, pracovníka IT Service Desku, kterému byl tiket přidělen. Správa tiketů je důležitá součást procesu poskytování služeb Service Desk, zlepšuje komunikaci mezi zákazníky a IT personálem a pomáhá zajišťovat, že problémy jsou řešeny rychle a efektivně.

V případě, že zákazník, který spravuje software v organizaci, potřebuje mít přednostně zpracované nerozpoznané záznamy, tak zadá u konkrétních záznamů nový požadavek na zpracování. Do zakládací zprávy je připojen seznam nerozpoznaných registrových záznamů. Správce softwarové knihovny pak dostane tento požadavek k řešení, zkopíruje seznam nerozpoznaných záznamů z požadavku do vstupního pole ve webové aplikaci. Následně aplikace ověří stav registrových záznamů, zda jsou všechny pokryté registrovými pravidly a mají přiřazené produkty. Stavy mohou nabývat hodnot „zpracovaný“, „čeká na zpracování“ a „nenalezeno v databázi“. Registrové záznamy se stavem „čeká na zpracování“ musí pokryty pravidlem. Uživatel se tedy proklikne na registrový záznam a vytvoří nové pravidlo, kterým jej pokryje. Pokud jsou všechny záznamy správně zpracované, tak správce knihovny případ uzavře, čímž se vygeneruje automatická odpověď a zákazník je tak informován, že je požadavek dokončený.

Navrhoval jsem navíc i přepsání zpracování tiketů tak, aby nebylo nutné kopírovat obsah tiketu do webové aplikace. Nevyřešené tikety by zde byly rovnou zobrazeny do rozbalitelných kartiček a dalo se s nimi přímo pracovat. Návrh byl však zamítnut kvůli velkému zásahu do systému. Centrálním bodem, kde se řeší tikety, je aplikace Alvao WebService, veškerá data o tiketech jsou ukládána na jinou databázi a sdílení by znamenalo komplikace. Pro správce knihovny bylo důležitější zaměřit se na ostatní části aplikace.

Zpracování tiketů bylo ale zdokonaleno, načtená data z tiketu jsou uložena do globální dočasné tabulky, ze které jsou při přechodu zpět na tuto stránku načtena a automaticky aktualizována podle změn, které se udály v databázi. Pro uložení dat ze vstupního pole byla použita trvalá *tempdb* tabulka, jelikož je možné k ní přistoupit z jiného sezení (anglicky session) nebo sql připojení a je smazána pouze při restartování sql serveru.

Pro zajištění souběžného používání aplikace více uživateli, jsou k názvům dočasných tabulek přidávána unikátní čísla *Id* právě přihlášených uživatelů.

Kapitola 5

Testování a nasazení aplikace

V této kapitole se zaměříme na tři hlavní typy testů, které se používají při vývoji aplikací: jednotkové, integrační a uživatelské testy. Jednotkové testy ověřují funkčnost jednotlivých částí kódu, integrační testy ověřují funkčnost aplikace jako celku a interakce mezi jednotlivými částmi, a uživatelské testy testují, zda je aplikace snadno ovladatelná a plně funkční pro uživatele. Zaměříme se také na nástroje a metody, které lze využít k automatizovanému testování a zefektivnění celého procesu.

Testování aplikace je nezbytnou fází v procesu vývoje softwaru. Cílem testování je ověřit správnou funkčnost aplikace tak, aby byla bezproblémová a bylo ji možné bezpečně používat. Právě na tyto kvality je kladen důraz, jelikož tato interní aplikace bude používána jako nástroj pro správu a rozvíjení knihovny produktů a pravidel. Proto byly otestována primárně základní operace s daty.

5.1 Jednotkové testování

ORM Dapper má výhodu v jednoduchosti, rychlosti a v přesné kontrole toho, jaké dotazy a příkazy se posílají na databázový server. Tato těsná vazba na SQL příkazy, ve srovnání třeba s rámcem *Entity Framework*, však způsobuje obtížnější možnosti testování.

V knize [7] v 10. kapitole „Testing the database“ jsem se dočetl o možnosti využít k testování databázi v paměti, anglicky *in-memory database*, kde se autor jmenovitě zmiňuje o technologii *SQLite*. Cituji: „Another way to isolate integration tests from each other is by replacing the database with an in-memory analog, such as *SQLite*.“ Autor dále vyjmenovává výhody, ale také varuje před použitím ve složitějších integračních testech, protože ve srovnání s databázovým systémem použitým v produkčním prostředí, mohou být výsledky rozdílné.

Po zvážení kladů a záporů jsem databázový systém *SQLite* využil. Jednotkové testy ověří základní operace úložišť, kde se bude využívat SQL jazyk ve velmi jednoduché podobě nepřesahující výrazně ANSI standard SQL-86, takže by k odlišnostem v chování nemělo dojít při použití jakéhokoliv SQL databázového stroje.

Využil jsem balíček *SQLite Core s PCLRaw* pro implementaci *SQLite*. V části *SetUp* testů inicializuji připojení k *SQLite* databázi v paměti. Vytvořím základní strukturu tabulek a dat. Takto vytvořené připojení pak přes *Mock* vstupuje do objektů úložiště, nad kterými se provádí jednotkové testy. V části *TearDown* se připojení k databázi ukončí, čímž databáze zaniká.

Pro případ řešení odlišností dialektů SQL jsem si připravil *Wrapper* nazvaný jako *SqliteConnectionWrapper*, který obaluje připojení, aby dokázal příkaz modifikovat. Což se dále ukázalo jako užitečné, protože pro *SQLite* bylo potřeba přeložit název funkce *IsNull()* na *IfNull()* a *GetDate()* na *DateTime('now')*. Vytvořil jsem i *CallBack* delegáta, který by se dal využít k náhradě uložených procedur nebo složitějších konstrukcí, což jsem i vyzkoušel. Bylo by takto možné snadno přejít na *SQLite* s uložením do souboru i s produkční částí.

Byly provedeny jednotkové testy pro úložiště *RegistryRuleRepository*. Otestovány jsou především metody pro vytvoření, úpravu a vymazání pravidel, protože se jedná o nejdůležitější operace a je nutné, aby byla zaručena jejich správnost při používání aplikace.

Otestovány byly také deterministické pomocné funkce, ty jsou nejlépe testovatelnou součástí aplikace. V testu se definují vstupní parametry a otestuje se výsledná návratová hodnota. Snadno se otestují různé varianty, které snadno pokryjí veškerý kód funkce. Není potřeba vytvářet nějaká rozhraní nebo falešné objekty (anglicky fake či mock). Pro čitelnost testu jsem v testoval funkci pro převod filtru verze z řetězce na čísla (*LS* a *MS*) v jednom kroku spolu s její opačnou variantou.

Je také možné testovat modely dat. Na objektech modelu naopak není mnoho kódu k otestování. Je možné ověřit výchozí hodnoty. Dále se dá ověřit, že třída správně uchovává data. Otestován byl model *RegistryRule*.

5.2 Integrační testování

Na úspěšné jednotkové testování, které testuje jednotlivé komponenty, navazuje integrační testování, které ověřuje, zda komponenty správně spolupracují a pracují správně jako celek. Provádí se testy komunikace mezi jednotlivými komponentami na jejich rozhraní, zda dochází ke správnému předávání dat. Cílem je ověřit fungování aplikace jako celku, aby aplikace byla spolehlivá, výkonná, a hlavně bez chyb ještě před nasazením do produkce.

Z integračních testů bylo provedeno zřetězení několika operací úložiště. Bylo vytvořené nové pravidlo, následně ověřeno, že existuje, nakonec vymazáno a ověřeno, že již neexistuje.

5.3 Uživatelské testování

K otestování aplikace samotnými zaměstnanci firmy Alvao byl dodán sql skript na vytvoření nových tabulek potřebných pro klasifikaci záznamů a uložení návrhů pravidel. Poskytnut byl také skript obsahující příkazy pro importování CLR funkce do MSSQL a skript pro vytvoření indexů a *full-text* indexů v několika tabulkách.

Aplikace byla průběžně testována uživateli, což také vedlo k získání zpětné vazby a dopracování některých nedostatků. Přidány byly například položky typ a kategorie produktu u jeho vytváření, které při specifikaci požadavků byly opomenuty a nebyly tak považovány za důležité. Zpětná vazba také vedla ke změně vyhledávání produktů, organizací a aktualizování tabulek. Obnovení obsahu bylo změněno, aby proběhlo se zpožděním až při dokončení psaní. Byly také odhaleny některé chyby.

Plán vývoje byl na pravidelných schůzkách konzultován a také jsem zde předváděl implementovanou funkcionalitu.

5.4 Nasazení do prostředí Azure App Service

Azure App Service je cloudová platforma od firmy Microsoft, která umožňuje provozovat web aplikaci v cloudu. V *Azure portal*[1] se nejprve vytvoří definice *App Service*. Volí se použité předplatné (anglicky subscription), název, způsob nasazení, *runtime* (.NET 6), operační systém (Windows/Linux), region (v našem případě západní/severní Evropa), cenovou kategorii stroje, parametry sítě, monitorování a další vlastnosti. Po vytvoření je možné doplnit další vlastnosti například uživatelskou doménu, ochranu proti *DDoS*, škálování, parametry pro připojení k databázi.

Poté je možné aplikaci nasadit. To se provede z *Visual Studio* volbou *Publish*, kde se vybere připravená definice v Azure. Existuje také možnost nepřetržitého nasazení (anglicky continuous deployment), kdy se provede propojení na GIT a nasazení změny je možné automatizovat.

Společně s testovací kopií databáze byla aplikace pokusně nasazena se základním nastavením *tier f1*, které není zpoplatněno. Vyzkoušelo se tak, že je uskutečnitelné provozovat aplikaci v prostředí *Azure App Service*. U této interní aplikace se předpokládá, že bude nadále agilně vyvíjena. Tato vynaložená práce sloužila k napsání a nachystání hlavní kostry webové aplikace, proto také nebyla přidána autorizace přístupu pomocí *Azure Active Directory*, kterou si chystá přidat firma Alvao až později.

[1] Webová stránka *Azure portal* dostupná na: https://portal.azure.com/

Kapitola 6

Závěr

Cílem práce bylo vytvořit webovou aplikaci, která se bude používat jako nástroj pro správu a rozšiřování knihovny rozpoznávacích pravidel a softwarových produktů, včetně organizací vydávajících tyto produkty.

V rámci vypracování práce bylo zapotřebí se seznámit s problematikou rozpoznávání softwaru a zjistit, jak zapadá do kontextu správy softwarových aktiv. K tomu bylo zásadní nastudovat, jaké druhy informací jsou zaznamenány na zařízení při instalaci programu a jaké jsou formáty zápisu pravidel pro detekci a rozpoznání softwaru na základě těchto informací. Kromě formátu pravidel firmy Alvao byly prozkoumány alternativní řešení, které stály za návrhem nového zápisu pravidel ve formě stromové struktury. Transformace původních pravidel na nová nebyla realizována kvůli komplikované integraci s moduly, které jsou závislé na knihovně produktů.

Důraz byl kladen především na usnadnění a částečnou automatizaci nejčastěji prováděných operací při vytváření nových pravidel. Na problémy s vytvářením pravidel bylo nahlíženo z různých úhlů, byly také prozkoumány různé metody, které by vedly k usnadnění práce.

Hlavním přínosem je zpracování registrových záznamů, při kterém dojde k rozdělení názvů do logických částí a následného sestavení pravidla. Vygenerované pravidlo je navrženo uživateli a ten ho může schválit nebo upravit dle potřeby. Separování názvů do jednotlivých částí má také význam při vytváření nových produktů, protože je možné přímo nabídnout ve formuláři extrahovanou verzi a edici. Zdokonaleno bylo také vyhledávání a přiřazování existujících produktů za použití různých vyhledávacích metod, z nichž nejlepších výsledků dosahovala metoda hodnotící počet shodných slov nebo spíše prefixů slov. Práci v aplikaci významně usnadňuje také klasifikace registrových záznamů, která se prokázala jako velmi úspěšná pro určení třídy neklíčových záznamů, a s kombinací navrhování pravidel značně urychluje očišťování nežádoucích záznamů.

Přepracovaná aplikace nabízí i přehlednější a přívětivější uživatelské rozhraní, které se také podílí na zefektivnění práce. Existuje však stále prostor pro rozvíjení aplikace. Mezi nejrelevantnější návrhy rozšíření je zařazeno přidání vyhledávání pravidel s nabídkou „včetně neplatných položek“ nebo přidání operace pro náhradu pravidla, produktu či organizace za jiné. Nabízí se také propojit aplikaci s modulem *HelpDesk*, aby se daly v aplikaci přímo zpracovávat tikety s žádostmi o přidání nových produktů.

Závěrem hodnotím práci kladně, naučil jsem se s mnoha technologiemi a měl jsem možnost být v kontaktu s reálnou firmou a uživateli, což je do budoucna velmi přínosné.

Literatura

[1] Bard, G. *Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric* [online]. 2007 [cit. 2023-02-29]. Dostupné z: https://dl.acm.org/doi/10.5555/1274531.1274545.

[2] Freeman, A. *Pro ASP.NET Core 6*. 9. vyd. Apress, 2022. 1253 s. ISBN 978-1-4842-7956-4.

[3] Han Jiawei, J. P. Bayes Classification Methods. In: *Data Mining: Concepts and Techniques*. 3. vyd. Morgan Kaufmann, 2012, kap. 8, s. 350–354. ISBN 978-9380931913.

[4] Hovold, J. *Naive Bayes spam filtering using word-position-based attributes and length-sensitive classification thresholds* [online]. 2005 [cit. 2023-04-13]. Dostupné z: https://aclanthology.org/W05-1712.pdf.

[5] Hunnebeck, L. Introduction. In: *ITIL Service Design*. 2. vyd. The Stationery Office, 2014, kap. 1, s. 3–10. ISBN 978-0113313051.

[6] IAITAM. *Software catalog* [online]. 2012 [cit. 2023-02-13]. Dostupné z: https://iaitam.org/what-is-a-catalog-for-sam-identifying-the-essentials-for-software-asset-management/.

[7] Khorikov, V. Testing the database. In: *Unit Testing Principles, Practices, and Patterns*. 1. vyd. Manning Publications, 2019, kap. 10, s. 229–256. ISBN 978-1617296277.

[8] Lv, T. *Spam Filter Based on Naive Bayesian Classifier* [online]. 2020 [cit. 2023-04-12]. Dostupné z: https://iopscience.iop.org/article/10.1088/1742-6596/1575/1/012054/pdf.

[9] Peter, J. *Using Machine Learning to Detect if Two Products Are the Same*. Diplomová práce. České vysoké učení technické v Praze, Fakulta Elektrotechnická.

[10] Přichystal, J. *Product name matching* [online]. 2021 [cit. 2023-03-02]. Dostupné z: https://akela.mendelu.cz/~jprich/prichystal-product-matching.pdf.

[11] Smith, S. *Building ASP.NET Core MVC application* [online]. Microsoft Corporation, 2022 [cit. 2023-04-02]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0.

[12] Wagner, B. *Use of record types* [online]. Microsoft Corporation, 2023 [cit. 2023-04-12]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/records/.

[13] Watson, B. *General Coding and Class Design.* Ben Watson, 2014. 155-184 s. ISBN 978-0990583431.

[14] Wenzel, M. *Design the infrastructure persistence layer* [online]. Microsoft Corporation, 2023 [cit. 2023-04-24]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design.

[15] Yildiz, S. *Comparison of ORM Dapper and EF Core* [online]. Medium, 2023 [cit. 2023-03-29]. Dostupné z: https://levelup.gitconnected.com/dapper-vs-ef-core-which-orm-framework-should-you-choose-for-your-net-application-54f2723b176a.

Příloha A

Obsah přiloženého paměťového média

- *src/*
  - Adresářová struktura obsahující veškerý kód
- *src/SwLibWebAppDapper/*
  - Webová aplikace
- *src/RegistryRecordParser/*
  - Program pro zpracování registrových záznamů
- *src/ProductClassificator/*
  - Klasifikátor produktů
- *src/SwLibWebAppDapper.Test/*
  - Projekt s testy
- *db_swlib.7z*
  - Komprimovaná záloha databáze
- *doc/*
  - Text technické zprávy