Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Inteligentný organizátor súborov

Bakalárska práca

2016
Stanislav Krajčovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Inteligentný organizátor súborov

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	9.2.9 aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. František Gyárfáš, PhD.

Bratislava, 2016
Stanislav Krajčovič

Poďakovanie:

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Obsah

Úvod **1**

1 Prehľad podobných existujúcich systémov **2**
1.1 MusicBrainz Picard 2
1.2 Stuff Organizer 3
1.3 TAGSPACES 4
1.4 Zhrnutie 5

2 Analýza záverečných prác s podobnou tématikou **6**
2.1 L.I.S.T. - Long-term Internet Storage of Tasks 6

3 Použité technológie **7**
3.1 Java 8 7
3.1.1 JVM 7
3.1.2 JDK 8
3.1.3 JRE 8
3.2 JavaFX 8
3.2.1 XML a CSS 9
3.3 JSON 10
3.3.1 JSON Objekt 10
3.3.2 JSON Pole 10
3.3.3 JSON Hodnota 11
3.3.4 JSON String 11
3.3.5 JSON Číslo 11
3.4 Gson 11
3.5 Python 11

4 Návrh **12**
4.1 Databáza 12
4.1.1 Modularita 12
4.1.2 Serializácia 12

4.1.3 Jednoduchosť 13
4.1.4 Databáza v kontexte aplikácie 13
4.2 Funkcie 14
4.2.1 Položky 14
4.2.2 Tagy 15
4.2.3 Popis 16
4.2.4 Kolekcie 17
4.2.5 Filtrovanie a vyhľadávanie 19

5 Implementácia **21**
5.1 Ifofile.java 21
5.2 Ifocol.java 22
5.3 Utility.java 22
5.3.1 directoryChooser 22
5.3.2 textInput 23
5.3.3 fileChooser 23
5.3.4 deletionWarning 23
5.3.5 createBeginningAlert 24
5.4 Handler.java 25
5.4.1 fillInternalStructures 25
5.4.2 serialize 25
5.4.3 deserialize 25
5.4.4 Text Search 26
5.4.5 Logic Search 26
5.4.6 Hlavné metódy 26
5.5 FileExtensions.java 28
5.6 Views 29
5.6.1 MainMenu 29

Záver **30**

Zoznam obrázkov

1.1 MusicBrainz Picard 2
1.2 Stuff Organizer 3
1.3 TAGSPACES 4

2.1 Filtrovanie 6

3.1 Špecifikácia JVM 8
3.2 XML syntax 9
3.3 CSS syntax 10

4.1 Náhľad databázy 13
4.2 Databáza v kontexte aplikácie 13
4.3 Súbory 14
4.4 Tagy, súbory a kolekcie v kontexte 18
4.5 Znázornenie vyhľadávania 20

5.1 directoryChooser GUI 22
5.2 textInput GUI 23
5.3 fileChooser GUI 23
5.4 deletionWarning GUI 24
5.5 createBeginningAlert GUI 24

Úvod

V súčasnosti sú používatelia desktopových počítačov a notebookov všetkých operačných systémov vystavovaní doslova nezvládateľnému počtu súborov v stromových štruktúrach, ktoré začínajú byť ťažkopádne a neprehľadné.

Súborov s pribúdajúcim časom nebúde ubúdať, práve naopak. Aj pri kompulzívnej kategorizácií a manuálnom usporadúvaní všetkých nových súborov do vopred určených priečinkov v istom bode v budúcnosti nastane stav, kedy sa táto štruktúra stane nedostačujúcou - preklikávať sa z priečinku do priečinku a hľadať želaný súbor môže zaberať netriviálne množstvo času. Je nutné poznamenať, že Windows, na ktorý bol tento informačný systém vyvíjaný, nemá zlý súborový prehliadač. File Explorer na Windows 10 má veľa užitočných funkcií - vyhľadávanie podľa mena, vyhľadávanie v obsahu súboru, zoraďovanie a dokonca aj group by funkciu. Avšak stále to nie je dosť, o čom svedčí aj nehynúce využívanie Total Commander-u a iných alternatív, ktoré sú tu už od roku 1993, kedy sa priemerný používateľ stretával so značne menším počtom súborov.

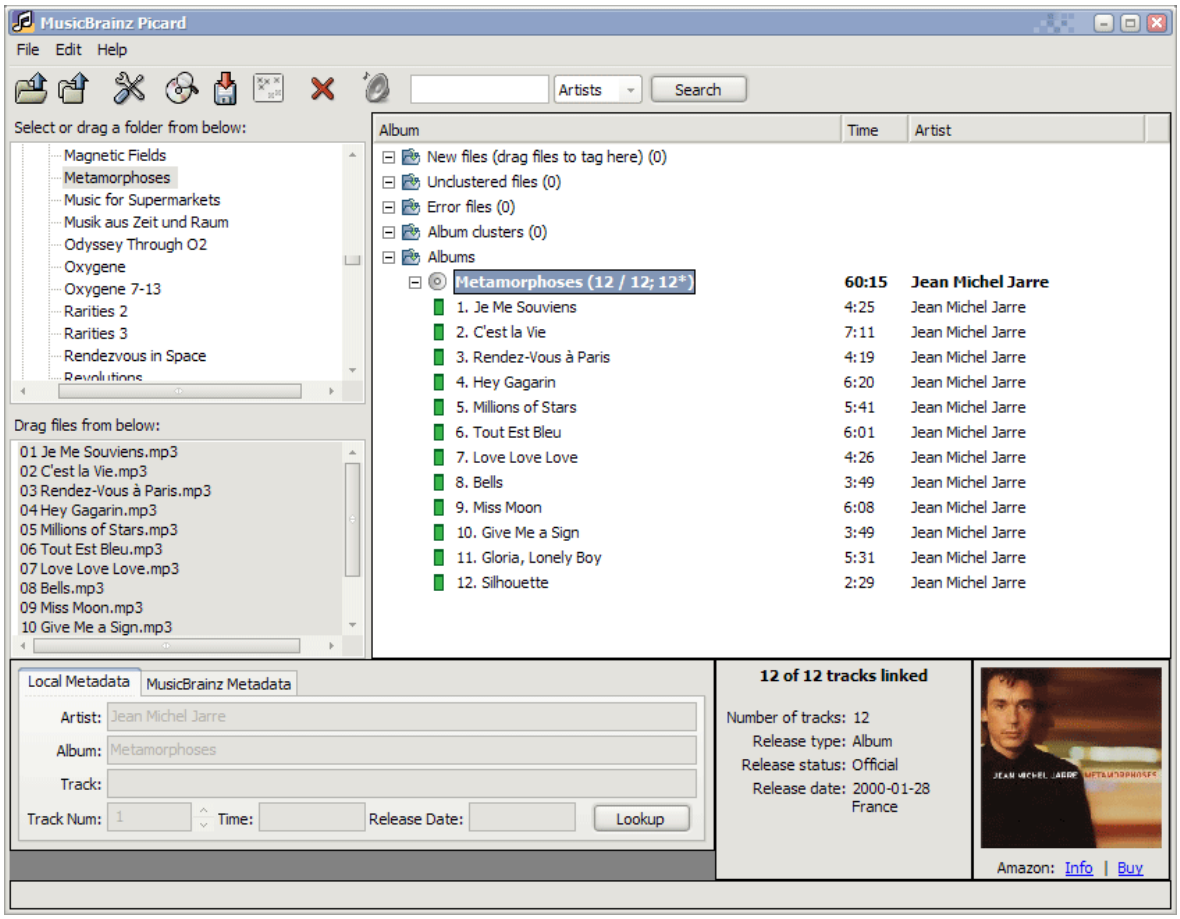
Čo keby sme na chvíľu zabudli na klasickú stromovú štruktúru a vytvorili náhradu, ktorá bude spočívať len z jedného stupňa vnorenia sa? Čo keby sme okrem klasického textového vyhľadávania poskytli používateľovi možnosť so súbormi skladovať viac informácií, podľa ktorých bude môcť využívať komplexnejšie spôsoby hľadania a mnoho ďalších funkcií?

Dostali by sme *Inteligentný Organizátor Súborov (Intelligent File Organizer), IFO*.

Kapitola 1

Prehľad podobných existujúcich systémov

1.1 MusicBrainz Picard



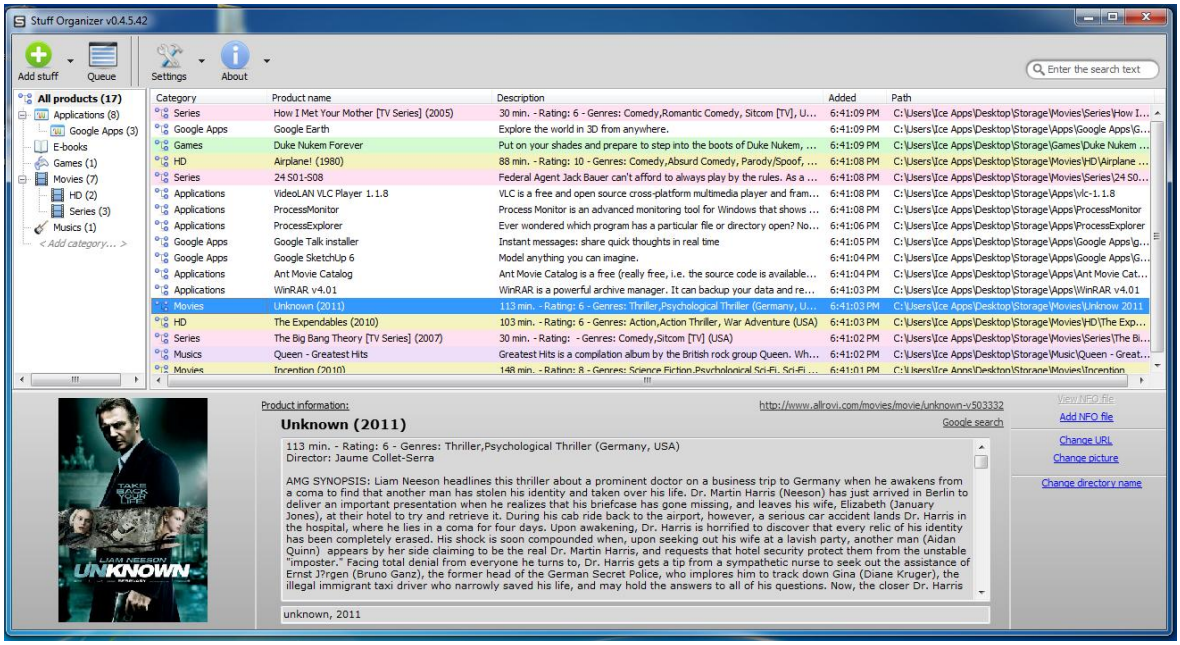
Obr. 1.1: MusicBrainz Picard

MusicBrainz Picard je voľne dostupný open-source softvér slúžiaci na identifikáciu, tagovanie a organizovanie digitálnych audio nahrávok. Picard identifikuje audio nahrávky

a CD nosiče porovnávaním ich metadát s položkami v MusicBrainz databáze. Metadáta audio nahrávok sú prostriedkom pre ukladanie informácií o súbore. Picard umožňuje aj pridávanie nových informácií, ako napríklad meno umelca, názov albumu, názov nahrávacieho štúdia, dátum vydania, prípadne zoznam umelcov a inštrumentov, na ktorých hrajú. Tieto informácie sú dostupné aj v databáze, ktorá je udržiavaná dobrovoľníkmi. [?]

Medzi nevýhody patrí fakt, že Picard je určený len pre hudobné súbory.

1.2 Stuff Organizer



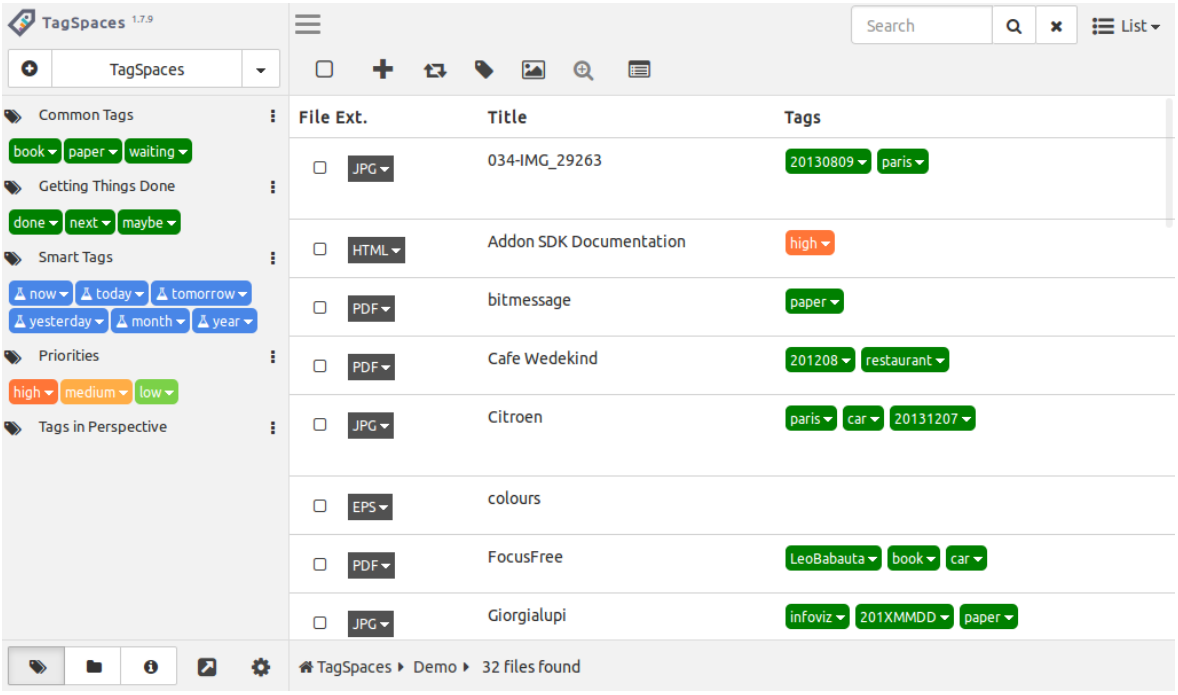
Obr. 1.2: Stuff Organizer

Stuff Organizer je voľne dostupný open-source softvér ktorý umožňuje kategorizovanie súborov a priečinkov, rozbaľovanie archívov a extrahovanie ISO súborov. Podporuje tagy a čítanie NFO súborov.

Stuff Organizer pracuje na báze vytvorenia internej databázy, v ktorej si udržiava informácie o načítaných súboroch, ktoré sa potom rozdelia do kategórií ako *Aplikácie*, *E-knihy*, *Hry*, *Movies* a pod. Používateľovi dovolí vytvoriť aj podkategórie. Načítané súbory je ďalej možné ľubovolne posúvať po disku (prípadne ich aj vytiahnúť z archívu). Ku každému súboru je možné pridať tagy a description. Do programu sa dajú pridať aj rôzne pluginy. [?]

Medzi nedostatky patrí neprítomnosť komplexnejšieho vyhľadávacieho mechanizmu a slabšia podpora tagov.

1.3 TAGSPACES



Obr. 1.3: TAGSPACES

TAGSPACES je ďalší voľne dostupný, multi platformový open-source organizátor. Medzi podporované platformy patrí Windows, Linux, OS X, Android, Firefox a Chrome. TAGSPACES obsahuje funkcie ako jednotlivé/masové tagovanie, usporadúvanie súborov, vyhľadávanie, farebné rozlišovanie tagov a pod. Tagy môžu byť organizované do skupín - ak existuje tagová skupina, dajú sa špecifikovať ktoré tagy patria do tejto skupiny a pri tagovaní súborov sa rozbalia. [?]

Medzi nedostatky patrí len základný file management, tagy sa uchovávajú v menách súborov (pri veľa tagoch pri súbore môže byť názov nečitateľný).

1.4 Zhrnutie

	výhody	**nevýhody**
MusicBrainz Picard	prepracovaný organizačný systém, veľká databáza udržiavaná dobrovoľníkmi, veľká podpora formátov a metadát, podpora pluginov, viac platoformový	zameranie len na jeden typ súborov
Stuff Organizer	presúvanie súborov po disku, vytváranie kategórií, rozzipovanie, okrem tagov pridávanie aj description	slabé vyhľadávanie, malá podpora tagov
TAGSPACES	mohutný systém tagovania, hierarchia tagov, multi platformový, responzívny dizajn,	len základné file management funkcie, uchovávanie tagov v menách súborov, zameranie len na tagy

Tabuľka 1.1: Zhrnutie

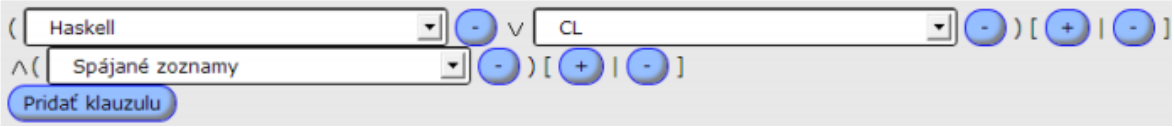
Kapitola 2

Analýza záverečných prác s podobnou tématikou

2.1 L.I.S.T. - Long-term Internet Storage of Tasks

L.I.S.T. je dlhodobé internetové úložisko úloh. Bol navrhnutý ako bakalárska práca Andreja Jursu v roku 2013. Okrem vytvárania a správy úloh umožňuje vytvárať aj zostavy úloh, odovzdávania zdrojových kódov, ich hodnotenie, vytváranie unit testov a púšťanie týchto unit testov na odovzdané odpovede so zobrazením výsledkov.

Medzi najzaujímavejšie funkcie patrí filtrovanie zadaní s využitím výrokovej logiky, konkrétne konjunkcie a disjunkcie.



Obr. 2.1: Filtrovanie

[?]

Kapitola 3

Použité technológie

Kapitola obsahuje stručné informácie o použitých technológiách.

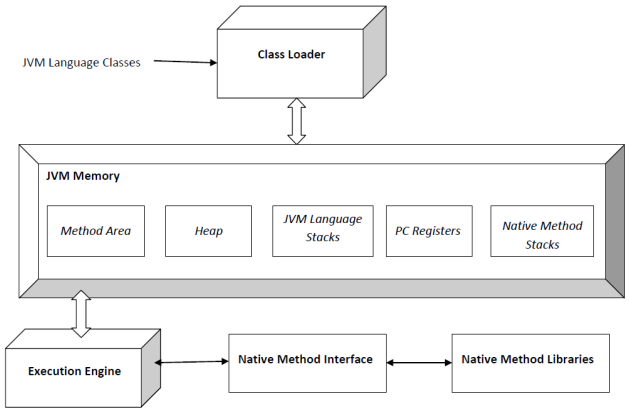
3.1 Java 8

Java je multi-paradigmový, objektovo orientovaný, generický a konkurentný programovací jazyk. [?] Java bola navrhnutá spôsobom "write once, run anywhere"[?], čo v skratke znamená, že Java programy sa kompilujú do bajtkódu, čo je séria inštrukcií pre Java Virtual Machine (JVM). [?]

Medzi najdôležitejšie novinky v Java 8 pre tento projekt patrí umožnenie písania lambda výrazov. [?]

3.1.1 JVM

Java Virtual Machine je abstraktný výpočtový stroj ktorý umožňuje počítaču spustiť program napísaný v Jave a skompilovaný do bajtkódu. Každá implementácia JVM sa musí držať špecifikácie, ktorá zaručuje interoperabilitu a schopnosť Java programov bežať všade, kde je podporovaná Java (kávomaty, bankomaty, a pod.) [?]



Obr. 3.1: Špecifikácia JVM

3.1.2 JDK

Java Development Kit obsahuje nástroje pre vývojárov, ako je javac (kompilátor), javadoc (generátor dokumentácie), java (loader/interpreter pre JVM), jar (archivátor na zabalenie tried projektu do jedného .jar súboru). [?]

3.1.3 JRE

Java Runtime Enviroment je podmnožinou JDK. Je to softvér obsahujúci prerekvizity potrebné na spustenie Java programu - spomínaný JVM a k tomu Java Class Library (JCL). JCL sú knižnice, ktoré môže Java aplikácia volať počas runtime-u. Keďže Java nie je závislá od operačného systému, aplikácie sa nemôžu spoliehať na knižnice v platforme, na ktorej bežia. [?]

3.2 JavaFX

JavaFX je softvér na vytváranie desktopových aplikácií a Rich Internet Applications (RIAs). JavaFX bola vytvorená ako náhrada za Swing, štandardnú Java GUI knižnicu pre Java SE. [?]

Do verzie 2.0 vývojári používali JavaFX Script kompilovaný do bajtkódu, ktorý bežal na každom PC obsahujúci Java SE, internetový prehliadač podporujúci Java EE a mobilné zariadenie s podporou Java ME. Od verzie 2.0 je JavaFX implementovaná ako natívna Java knižnica, napísaná v Java kóde.

JavaFX využíva FXML, skriptovací značkovací jazyk založený na XML, ktorý slúži na definovanie používateľských rozhraní. Obsahuje cez 60 formulárov, komponentov a grafov, ktoré sú prispôsobiteľné cez CSS.

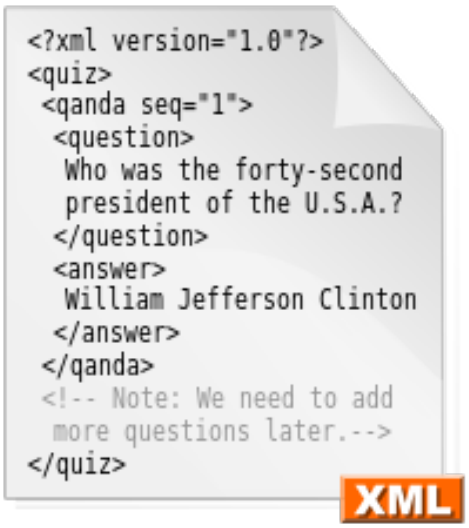
JavaFX obsahuje podporu pre desktopové aplikácie a prehliadače bežiace na operačných systémoch Windows, OS X a Linux. [?]

Na komplexnejšie prispôsobenie súborov *.fxml* sa vytvárajú tzv. controllery. Controller je Java trieda, ktorá je referencovaná v hlavičke *.fxml* súboru a dajú sa v nej definovať metódy na ovládanie tlačidiel a iných objektov v JavaFX.

3.2.1 XML a CSS

XML

Extensible Markup Language je značkovací jazyk definujúci pravidlá na kódovanie dokumentov. [?]



Obr. 3.2: XML syntax

CSS

Cascading Style Sheet je štýlovací jazyk používaný na definovanie prezentácie dokumentu písaného v nejakom značkovacom jazyku. [?] V prípade tejto práce je to FXML.

Obr. 3.3: CSS syntax

3.3 JSON

JavaScript Object Notation je alternatíva k XML. Je to formát na vymieňanie dát, čitateľný pre ľudí a ľahko parsovateľný/generovateľný pre stroje. JSON je založený na nadmnožine JavaScript-u. JSON nie je závislý od žiadneho programovacieho jazyku, aj keď používa konvencie z tých najpopulárnejších. [?]

JSON je stavaný na dvoch štruktúrach:

- asociatívne páry meno -> hodnota; ako HashMap v Java, dict v Python
- usporiadaný zoznam hodnôt; ako Array v Java, list v Python

Tieto univerzálne dátové štruktúry využíva JSON práve kvôli tomu, že sú využívané v každom modernom programovacom jazyku.

3.3.1 JSON Objekt

Objekt je neusporiadaná množina dvojíc meno -> hodnota. Definícia objektu začína *{* a končí *}*, hodnota sa k menu prideľuje cez *:* (dvojbodku) a objekty sa oddeľujú *,* (čiarkou).

3.3.2 JSON Pole

Pole je usporiadaná kolekcia hodnôt. Definícia poľa začína *[* a končí *]*, hodnoty sa oddeľujú *,* (čiarkou).

3.3.3 JSON Hodnota

Hodnota môže byť string, číslo, true, false, null, objekt alebo pole.

3.3.4 JSON String

JSON String je následnosť nula alebo viacerých Unicode charakterov obalených dvojitými úvodzovkami. Charakter je reprezentovaný ako string s jedným charakterom.

3.3.5 JSON Číslo

JSON Číslo je ako Java float.
[?]

3.4 Gson

Gson je Java knižnica od Google slúžiaca na konvertovanie Java Objektov do ich JSON reprezentácie a naspäť, textov vo formáte JSON do Java Objektov. Gson dokáže serializovať aj objekty ku ktorým používateľ nemá zdrojový kód, podporuje aj Java Generics.

Gson obsahuje jednoduché metódy ako *toJson()* a *fromJson()*, ktoré zabezpečujú samotnú serializáciu a deserializáciu. [?]

3.5 Python

Python je moderný, vysoko-úrovňový, multi-paradigmový, dynamicky programovateľný jazyk. Na beh skriptov napísaných v Python-e je potrebný nainštalovaný interpreter. Tieto skripty sa však dajú zamraziť do spustiteľných programov, ktoré nepotrebujú interpreter na svoje fungovanie. [?]

Kapitola 4

Návrh

Táto kapitola obsahuje návrh informačného systému Inteligentný organizátor súborov.

4.1 Databáza

Keďže aplikácia si bude potrebovať zachovávať informácie o načítaných súboroch a iné dôležité dáta aj po jej zatvorení, bude potrebné vytvoriť databázu, v ktorej budú tieto informácie programu kedykoľvek dostupné.

Z typu využitia databázy vyplýva, že nebude potrebné využívať súčasné databázové systémy, ale stačí si navrhnuť vlastný, jednoduchý, malý, ľahko serializovateľný, modulárny a prehľadný systém.

Databáza sa bude otvárať vždy pri spustení a vypínaní programu na čítanie a zapisovanie informácií do a z interných dátových štruktúr.

Ak databáza nebude existovať, IFO túto skutočnosť detekuje a bude sa podľa toho správať.

4.1.1 Modularita

Aplikácia bude mať práve jednu hlavnú databázu (ďalej databáza). Databáza bude dostupná na pevnom disku. Používateľ si bude môcť databázu skopírovať a používať aj v inej inštancii IFO.

4.1.2 Serializácia

Hlavný nárok je jej kompletná serializácia.

Databáza sa bude serializovať buď na vyžiadanie používateľa alebo pri zatvorení aplikácie. Vhodné prirovnanie je ukladanie ľubovolného textového dokumentu.

Deserializovanie bude prebiehať pri otvorení programu, alebo opäť na vyžiadanie. Pri veľkých databázach program používateľovi oznámi, že musí počkať a zabráni mu

zasahovať do GUI.

Aplikácia pracuje s veľkým množstvom citlivých dát a preto je dôležité, aby mala mechanizmy na zabránenie úplnej straty dát. Pri zlyhaní zapisovania databázy do súboru sa obnoví posledný fungujúci stav databázy. Tento stav sa bude udržiavať v separátnom priečinku. Pri zlyhaní načítania databázy do aplikácie o tom bude používateľ informovaný.

4.1.3 Jednoduchosť

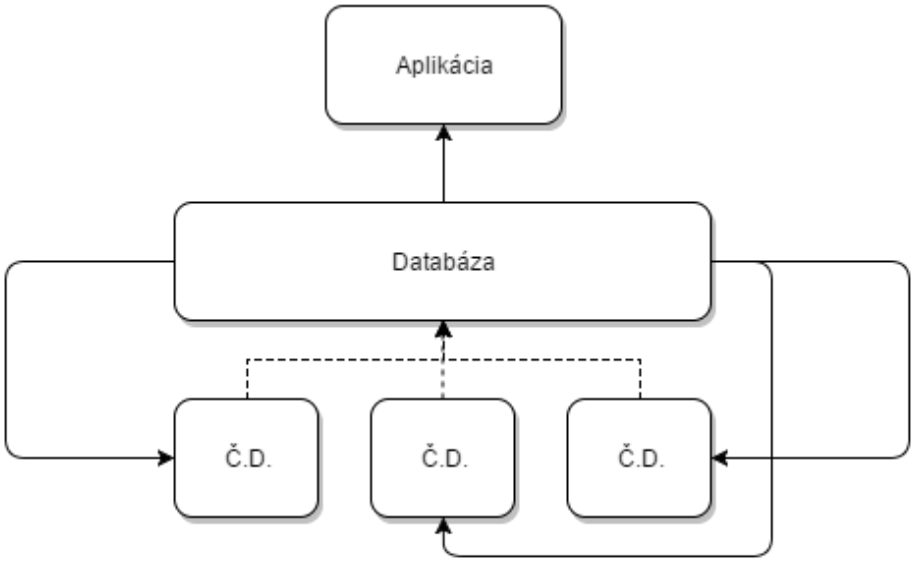
Informácie, ktoré si databáza bude musieť pamätať sú nasledovné - informácie o súboroch, informácie o kolekciách súborov a posledný použitý unikátny kľúč. V niektorých prípadoch môže obsahovať tisíce súborov a stovky kolekcií, preto je dôležité, aby parsovanie a vkladanie do pamäte aplikácie pri načítaní netrvalo dlho, čo platí aj pri zapisovaní do databázy.

Databáza bude čitateľná aj pre ľudí, avšak ako je vidno na obrázku 4.1, nie je vytváraná s týmto cieľom.


```
{"1":{"id":1,"description":"","tags":{"":["asd"]},"popularity":0,"absolutePath":"C:\\Users\
{"All":{"name":"All","filesInside":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,2
40
```


Obr. 4.1: Náhľad databázy

4.1.4 Databáza v kontexte aplikácie



Obr. 4.2: Databáza v kontexte aplikácie

4.2 Funkcie

Táto časť do podrobna popisuje požiadavky na funkcionalitu aplikácie.

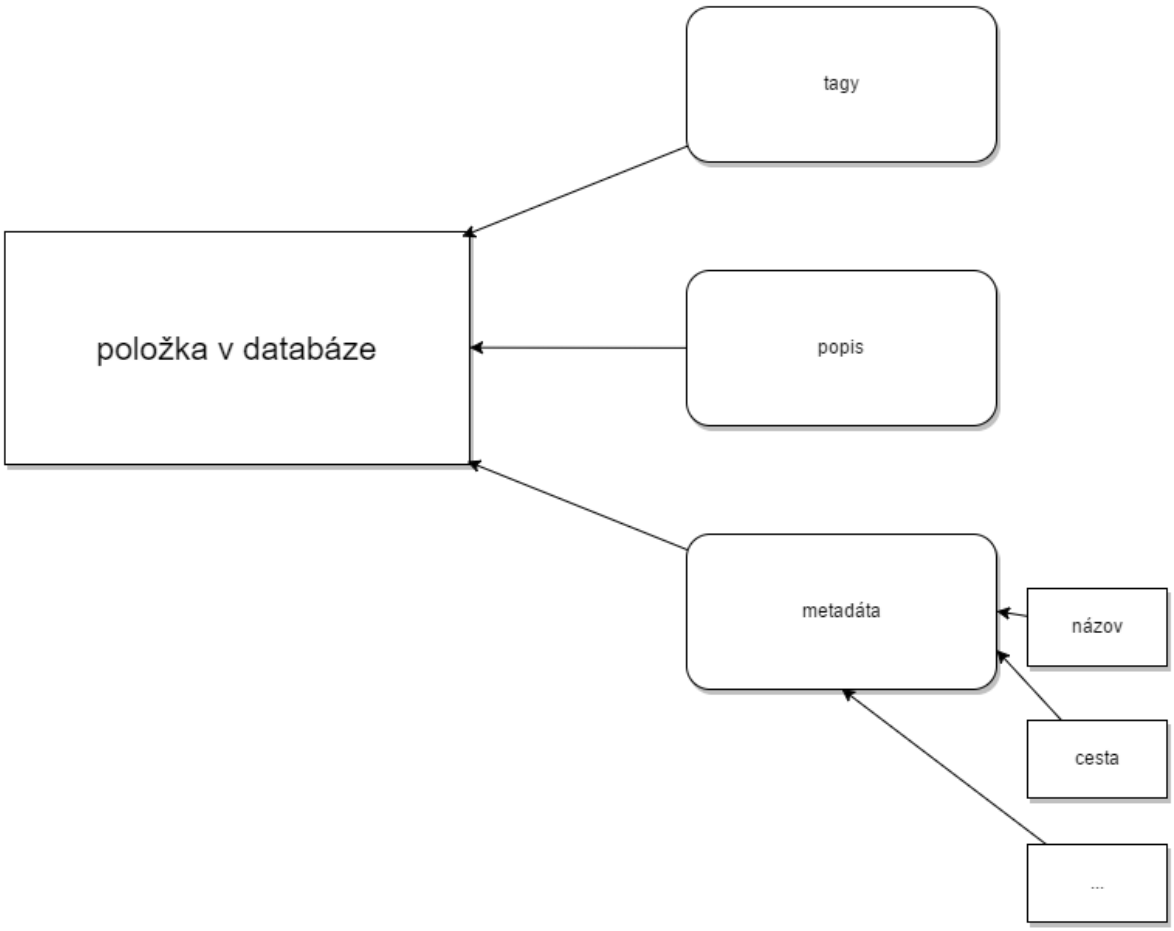
Položky databázy (ďalej len položky) budú atomickou jednotkou aplikácie.

Kolekcie budú zoznamy položiek.

4.2.1 Položky

Táto podčasť popisuje narábanie s položkami.

Položky v databáze si budú okrem informácií získaných z metadát súborov od operačného systému pamätať aj vlastné údaje - tagy, popis a iné.



Obr. 4.3: Súbory

Pridávanie súborov/položiek

Pridávanie súborov do databázy (a teda vytváranie položiek) bude zabezpečené dvoma spôsobmi.

Prvý spôsob bude pridávanie súborov manuálne, vybraním priečinku cez dialóg. Používateľ si bude môcť vybrať, či sa súbory budú nahrávať aj rekurzívne, tj. z podpriečinkov v danom priečinku. Každý súbor bude môcť byť nahraný aj viackrát. Po

úspešnom vykonaní akcie bude používateľ informovaný textom v stavovom riadku v spodnej časti aplikácie (ďalej stavový riadok).

Druhý spôsob bude importovanie databázy. Keďže aplikácia na fungovanie niektorých funkcií potrebuje vedieť lokáciu súborov na disku, po načítaní databázy (aj pri iných situáciách) sa bude overovať, či všetky súbory ëxistujú", to znamená, či sa nachádzajú na mieste, ktorý definuje databáza. Ak nie, používateľovi sa to oznámi v stavovom riadku.

Odstraňovanie položky

Aplikácia bude zabezpečovať viacero foriem odstraňovania položiek.

Jeden spôsob je úplné odstránenie, to znamená, že sa vymaže akákoľvek referencia na zvolenú položku, aj zo všetkých kolekcií.

Druhý spôsob je odstránenie z vybranej kolekcie, čo zmaže len referenciu v príslušnom kontajneri (kolekcii). Položka sa bude v databáze nachádzať aj po vykonaní tohto úkonu.

Úspešné, respektíve neúspešné vykonanie týchto akcií budú používateľovi oznámené v stavovom riadku.

Presúvanie položiek

Presúvanie položiek bude mať taktiež viacero foriem.

Prvá forma bude presúvanie položiek po kolekciách. Používateľ si vyberie položku/položky, ktoré chce presunúť a cieľovú kolekciu. V databáze sa zmenia informácie o kolekciách - z tej, ktorej boli položky presúvané sa vymažú referencie a pridajú sa do novej.

Druhá forma je presúvanie súborov po HDD, z priečinku do priečinku, dvoma spôsobmi. Operačný systém poskytuje kopírovanie a presúvanie. Pri kopírovaní sa vytvorí nový súbor na disku, totožný s originálom a v databáze sa položka skopíruje s rovnakými tagmi a popisom. Pri presúvaní sa súbor na disku zmaže zo zdrojovej cesty a vytvorí v cieľovej ceste. Nová položka reprezentujúca súbor v databáze sa skopíruje na koniec databázy so všetkými tagmi a popisom a pôvodná sa zmaže.

4.2.2 Tagy

Tagy budú kľúčové slová, pomocou ktorých sa dá okrem jednoduchého textového vyhľadávania použiť aj vyhľadávanie na základe množinových operácií - má zmysel hľadať položky, ktoré nejaký konkrétny tag neobsahujú (ak to používateľ explicitne zadá). Tagov bude môcť mať každá položka neobmedzene veľa, budú case sensitive a budú sa uchovávať v databáze pri položkách ku ktorým patria.

Pridanie tagov ku položkám

Pri tejto funkcii bude možné pridať tagy do položiek dvoma spôsobmi. Oba spôsoby zahŕňajú pridávanie do jednej, alebo aj viacerých položiek.

Prvý spôsob bude zahŕnať vpisovanie jednotlivých tagov do poľa, pričom po vyplnení každého želaného tagu ho bude treba potvrdiť. Políčko sa vymaže a bude pripravené na prijímanie ďalších tagov.

Druhá metóda bude umožňovať masové pridávanie - do druhého políčka bude možné zadať viac tagov naraz oddelených čiarkami. Tie sa potom rozparsujú a popridávajú ku položke.

Ak sa nejaký tag bude v položke už nachádzať, jeho pridanie sa odignoruje. Ak budú tagy pridávané do viacerých položiek naraz, pridajú sa len do tých v ktorých sa ešte nenachádzajú.

Aplikácia následne informuje používateľa cez stavový riadok o úspešnosti akcie.

Odstránenie tagov z položiek

Keďže bude možné zvoliť aj viacero položiek naraz, funkcia odstránenie tagov z položiek zobrazí okno so zoznamom všetkých tagov všetkých vybraných položiek. Používateľ si zvolí tagy, ktoré chce odstrániť a potvrdí vstup. Ak sa vybraný tag nenachádza vo všetkých vybraných súboroch, jeho odstránenie sa v týchto prípadoch odignoruje.

4.2.3 Popis

Popis bude text priradený k položke a bude limitovaný na určitý počet charakterov. Bude zohľadnený pri textovom vyhľadávaní a jeho hlavnou náplňou bude zorientovanie sa pre používateľa, doplní kontext tam, kde tagy nebudú stačiť.

Pridanie popisu

Funkcia slúži na pridanie popisu ku všetkým zvoleným položkám. Ak je súčasný popis neprázdny, nahradí sa.

Odstránenie popisu

Vzhľadom na možnosť vybrania viacerých položiek naraz je relevantné, aby existovala explicitná funkcionalita na odstránenie popisu.

Aj táto, aj funkcia Pridanie popisu oznámi používateľovi úspešnosť v stavovom riadku.

4.2.4 Kolekcie

Kolekcie budú zoznamy položiek. Kolekcia nebude mať limit na počet obsahujúcich objektov, bude si udržiavať len referencie na položky podľa príslušných *id* v databáze. Každá položka sa bude môcť v konkrétnej jednej kolekcii nachádzať len raz. Názvy budú case sensitive.

Kolekcia bude základný stavebný kameň celého programu - každá položka sa musí nachádzať aspoň v jednej kolekcii. Táto kolekcia (ďalej hlavná kolekcia) bude mať špeciálne nastavenia, popísané v jednotlivých podčastiach.

Vytvorenie kolekcie

Vytvorenie kolekcia sa bude dať zrealizovať opäť dvoma spôsobmi.

Prvý - vytvorenie prázdnej kolekcie. Táto metóda vytvorí prázdny kontajner/nádobu s používateľom zadaným menom, ktorá je schopná prijímať súbory.

Druhý spočíva vo vytvorení kolekcie z vybraných súborov. Používateľ si vyberie súbory z inej kolekcie a tie sa potom skopírujú do novej kolekcie, do ktorej, ak sa nevyplní meno, vloží sa prednastavené.

Kopírovanie kolekcie

Kopírovanie kolekcie vytvorí novú položku v databáze, identickú s kolekciou, ktorej kópiu používateľ vyrába. Meno kolekcie ostane rovnaké, pridá sa identifikátor ku menu kópie.

Hlavná kolekcia bude skopírovateľná.

Odstránenie kolekcie

Táto funkcia odstráni vybranú kolekciu z databázy, čím sa zabudnú referencie na jednotlivé súbory, ktoré obsahovala. Ak sa súbory budú nachádzať v iných kolekciách, ostanú v nich nezmenené a s nimi aj ich dáta (tagy, popis, a metadáta).

Hlavná kolekcia sa nebude môcť dať odstrániť.

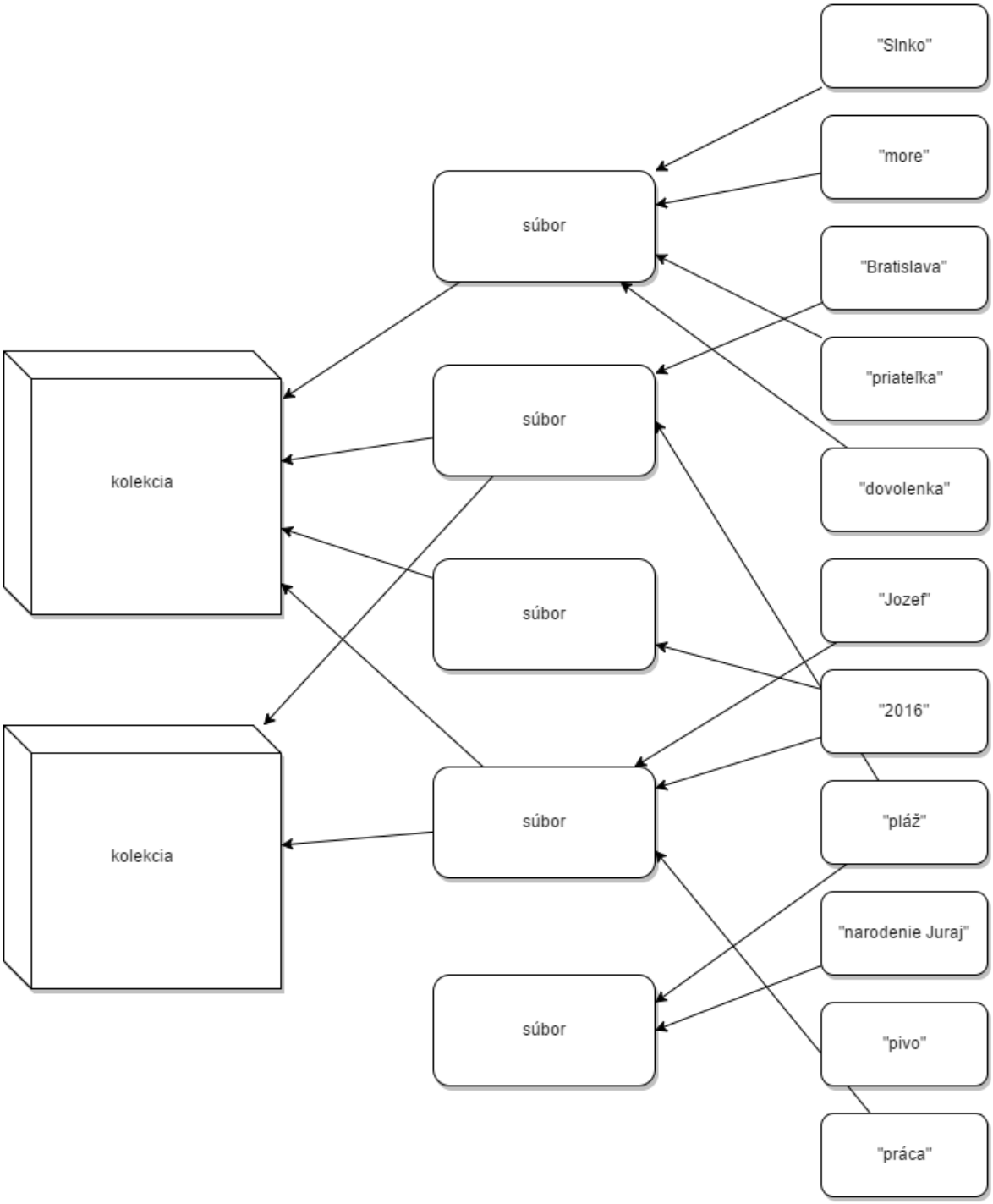
Premenovanie kolekcie

Funkcia premenuje kolekciu na používateľom zadané meno. Ak už existuje kolekcia s rovnakým menom, k novému menu sa pridá identifikátor na rozlíšenie.

Hlavná kolekcia sa nebude dať premenovať.

Kopírovanie kolekcie na disku

Táto funkcia bude slúžiť na kopírovanie súborov na disku, v kolekcii.



Obr. 4.4: Tagy, súbory a kolekcie v kontexte

4.2.5 Filtrovanie a vyhľadávanie

Ku základným funkciám bude musieť ešte zaručene patriť aj spôsob na vyhľadávanie súborov. IFO bude poznať tri cesty na vyhľadávanie:

- filtrovanie
- textové vyhľadávanie
- vyhľadávanie pomocou množinových operácií

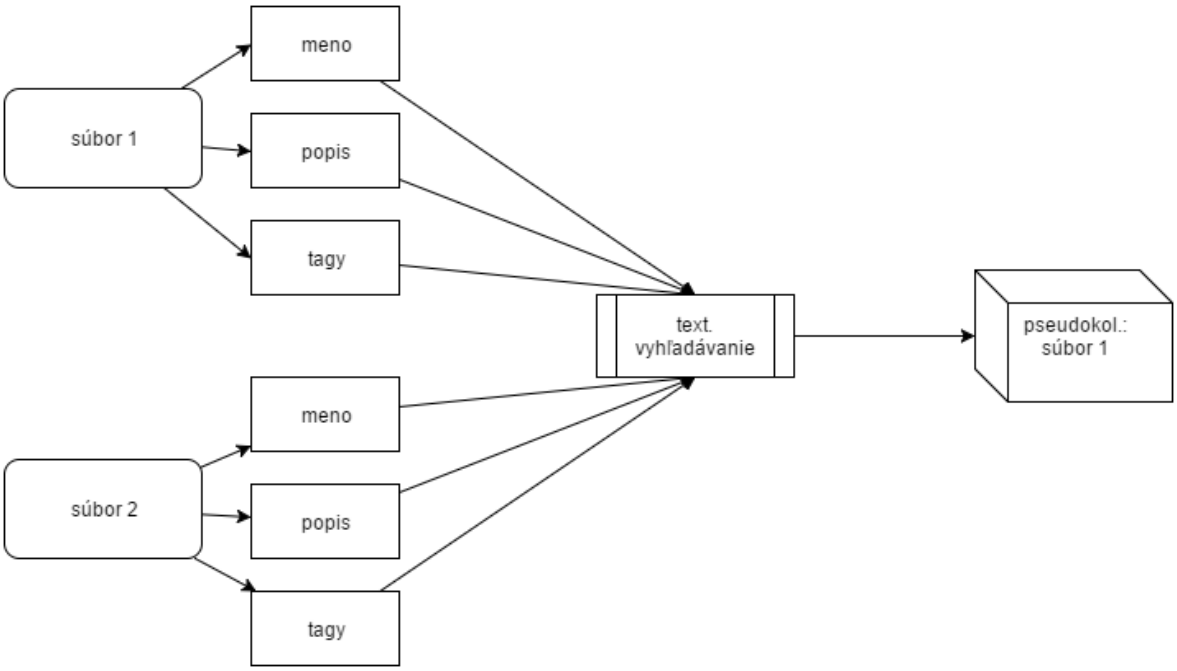
Filtrovanie

IFO bude ukazovať zoznam kolekcií a po kliknutí na kolekciu zobrazí všetky súbory, ktoré sa v nej nachádzajú. Na rýchlejší prístup k súborom bude program poskytovať funkcionalitu na zúženie výberu zobrazených položiek databázy podľa toho, či vo svojom mene obsahujú reťazec v textovom poli slúžiacom na filtrovanie.

Táto funkcionalita bude spomedzi všetkých troch poskytovaných najjednoduchšia a najprístupnejšia, pretože bude fungovať len nad menami súborov.

Textové vyhľadávanie

Textové vyhľadávanie bude zastrešovať vyhľadávanie nad menami, popismi a tagmi položiek v databáze. Vytvorí náhľad na súbory vyhovujúce vyhľadávaniu. Bude to vlastne pseudokolekcia, ktorá sa neuloží do databázy a po kliknutí na kolekciu zapamätanú v databáze sa zabudne. Nad touto pseudokolekciou bude možné vykonávať všetky funkcie ako na ostatnými, normálnymi kolekciami, popísané v kapitole Kolekcie a aj Filtrovanie.



Obr. 4.5: Znázornenie vyhľadávania

Vyhľadávanie pomocou množinových operácií

Tento spôsob vyhľadávania bude prebiehať len nad tagmi v položkách databázy. Bude to najkomplexnejší spôsob vyhľadávania. Na korektné fungovanie tejto funkcionality je potrebné vyplnenie tagov v aspoň niektorých súboroch, lebo položky bez tagov sa do vyhľadávania nezarátavajú.

Medzi operácie vykonávajúce sa patrí konjunkcia, disjunkcia a negácia. SKONZULTOVAT KOKOS

Kapitola 5

Implementácia

Táto kapitola obsahuje implementáciu informačného systému Inteligentný organizátor súborov (ďalej IFO alebo program).

5.1 Ifofile.java

Ifofile je vlastná dátová štruktúra, ktorá ukazuje na súbor na disku a je položkou v databáze.

Ifofile má nasledujúce triedne premenné:

- Integer id - identifikačné číslo v databáze; každý prvok má svoje vlastné, unikátne číslo
- String description - popis
- TreeMap tags - mapa tagov
- String absolutePath - absolútna cesta definujúca súbor na disku
- String name - meno súboru a zároveň aj položky v databáze
- boolean linked - či sa súbor nachádza na mieste definovanom absolútnou cestou
- a ďalšie...

Okrem spomínaných premenných obsahuje táto základná trieda aj metódy na manipulovanie s nimi. Medzi zaujímavejšie patrí samotný konštruktor, ktorý načíta súbor z cesty, ktorú dostane v parametri a za pomoci Java knižnice *java.io.File* z neho vyberie metadáta (meno, rodiča, či je to súbor alebo priečinok a iné).

Ďalšia netriviálna metóda je *setNewRawCustomAttributes*, ktorá do zvoleného súboru korektne nastaví nové tagy a popis.

Medzi jednoduchšie, no nemenej dôležité metódy patrí pridávanie a odstraňovanie tagov do a z TreeMap, nastavovanie popisu, prepínanie logickej hodnoty *linked* a ďalšie.

Je dôležité podotknúť, že aj keď databáza je vlastne "nádoba", v ktorej je id priradené súboru, v záujme zrýchlenia niektorých algoritmov (ktoré budú popísané neskôr) je výhodnejšie, aby si aj každý súbor pamätal vlastné *id*, pod ktorým je uložený.

5.2 Ifocol.java

Ifocol je trieda a dátová štruktúra reprezentujúca kolekcie v IFO.

Ifocol je množina, v ktorej sa nachádzajú *id* položiek z databázy. Pamätá si aj vlastné meno. Obsahuje metódy na narábanie s množinou (pridávanie, odstraňovanie, zobrazovanie).

Za spomenutie stojí metóda *clone*, ktorá sa využíva na vytváranie nových kópií kolekcie. Tieto kópie sa nevytvárajú len ako nové referencie, ale nakopírujú sa v pamäti. Toto riešenie je podstatné v pri narábaní so súbormi na disku, ktoré je rozobraté v časti (SEM REF).

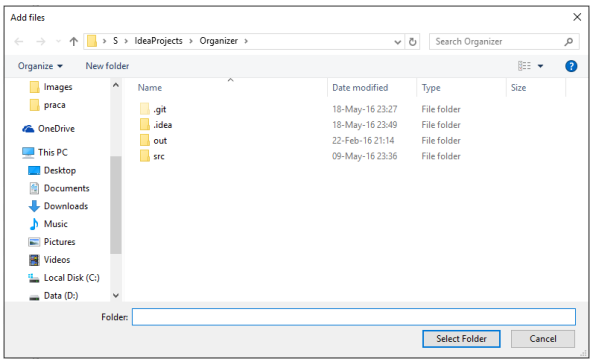
5.3 Utility.java

Utility je úžitková trieda obsahujúca statické metódy a premenné, ktoré logicky nezapadali do iných tried v IFO.

Sú to metódy obaľujúce triedy z JavaFX knižnice.

5.3.1 directoryChooser

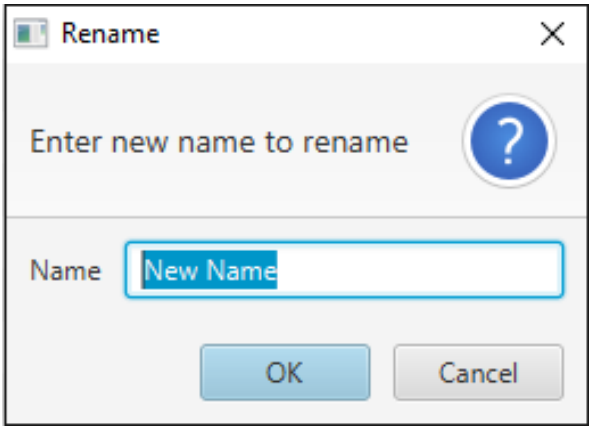
Metóda, ktorá používateľovi zobrazí dialóg, v ktorom si vyberie priečinok a *directoryChooser* ho následne spracuje a vráti.



Obr. 5.1: directoryChooser GUI

5.3.2 textInput

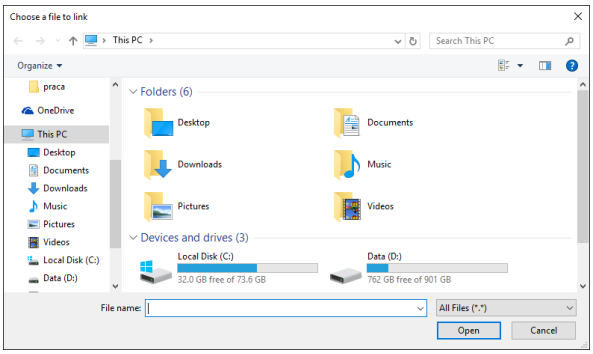
Metóda zobrazujúca dialóg s vstupným textovým políčkom, do ktorého používateľ zadá informáciu, ktorá sa vráti.



Obr. 5.2: textInput GUI

5.3.3 fileChooser

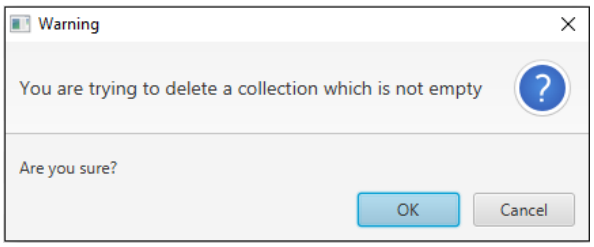
Metóda, ktorá zobrazí dialóg podobný *directoryChooser*, s tým rozdieľom, že tentokrát používateľ ukazuje na súbor, nie na priečinok.



Obr. 5.3: fileChooser GUI

5.3.4 deletionWarning

Dialóg ktorý používateľa varuje pred akciou, ktorú sa chystá vykonať. Stlačením tlačídla OK sa akcia vykoná, Cancel ju zruší.



Obr. 5.4: deletionWarning GUI

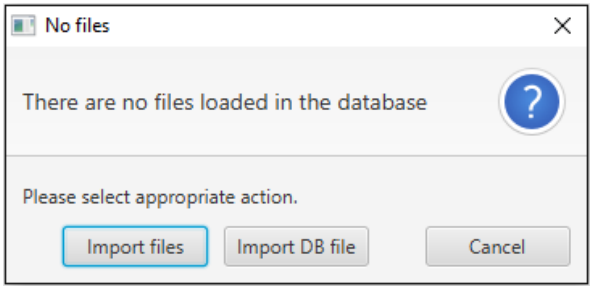
5.3.5 createBeginningAlert

Metóda zobrazujúca dialóg pri prvom spustení programu. Detekcia prvého spustenia prebieha tak, že sa program snaží nájsť a otvoriť svoj databázový súbor, ktorého lokácia je preddefinovaná a relatívna k lokácii programu. Ak sa tento súbor nepodarí otvoriť, existuje veľká pravdepodobnosť, že program je spustený po prvýkrát. V tomto prípade sa teda používateľovi poskytnú dve možnosti.

Prvá je importovanie súborov - zavolá sa *directoryChooser* a súbory zo zvoleného priečinku sa načítajú do databázy.

Druhá je importovanie databázového súboru. Toto riešenie predpokladá pokročilého používateľa. Databáza sa načíta do interných štruktúr programu a skopíruje na vlastné miesto, kde sa bude IFO pri ďalšom spustení pozerať. Pri tejto možnosti je ešte dôležité skontrolovať, či súbory definované v databáze naozaj existujú. Na to slúži trieda Handler, s ktorou táto metóda úzko spolupracuje.

Využíva konkrétne tri metódy - *fillInternalStructures*, *deserialize*, *checkFilesExistence*.



Obr. 5.5: createBeginningAlert GUI

5.4 Handler.java

Zmyslom tejto triedy je narábanie so všetkými ostatnými triedami v IFO, okrem tých, ktoré ovládajú používateľské rozhranie.

Obsahuje databázu:

- files - HashMap, ktorá ma ako kľúč id a hodnotu Ifofile
- collections - HashMap, ktorá má ako kľúč meno kolekcie a ako hodnotu samotnú kolekciu
- allTags - HashSet (množina), ktorá si pamätá všetky v súčasnosti existujúce tagy zo súborov
- logicFound - pomocná množina na uchovávanie výsledkov pomocou vyhľadávania pomocou množinových operácií

5.4.1 fillInternalStructures

Metóda, ktorá ako parametre dostáva cestu a boolean hodnotu, či má prehľadávať aj rekurzívne (podpriečinky).

addFile

Metóda volaná z *fillInternalStructures*, ktorá má za úlohu korektne vložiť súbor do databázy, to znamená so správnym id, a vložiť ju do správnych východzích kolekcií a hlavnej kolekcie. Východzie kolekcie sa určujú podľa prípon súborov, nachádzajú sa v statickom asociatívnom poli. Napríklad prípona *.avi* sa vloží do kolekcie *Video*.

5.4.2 serialize

Metóda volaná pri každom vypnutí programu a používateľom zadaným príkazom *export*, ktorá otvorí súbor predstavujúci databázu a pomocou *Gson* knižnice serializuje a následne zapíše vnútorné dátové štruktúry.

5.4.3 deserialize

Metóda volaná pri každom zapnutí programu. Metóda otvorí databázový súbor, rozparsuje informácie a naplní interné dátové štruktúry pomocou *Gson* knižnice - súbory, kolekcie, všetky tagy. Následne skontroluje existenciu súborov spomenutých v položkách databázy pomocou metódy *checkFilesExistence*, ktorá sa vykonáva v inom vlákne, ako to, na ktorom beží IFO.

5.4.4 Text Search

Realizácia vyhľadávania v menách položiek, v tagoch a popise ktoré obsahujú, je uskutočnená prejdením celej databázy súborov a hľadaním zhody s používateľom zadaným textovým výrazom. Identifikátory položiek vyhovojúcim tomuto kritériu sa uložia a vrátia.

5.4.5 Logic Search

Vyhľadávanie nad tagmi položiek je zabezpečené metódou *logicSearchCore*, ktorá využíva dve pomocné metódy - *ifoAnd* a *fileContainsTags*.

fileContainsTags berie ako argumenty Ifofile a set tagov. Ak súbor neobsahuje čo i len jeden tag, vráti logickú hodnotu *false*, inak *true*.

ifoAnd berie ako argumenty dva sety, prvý obsahuje používateľom zadané tagy, ktoré musí položka obsahovať; druhý obsahuje tagy, ktoré položka nesmie obsahovať. Na zisťovanie, či položka vyhovuje obom podmienkam sa používa predchádzajúca pomocná metóda.

Ako posledný úkon vyhľadávania sa vykoná zjednotenie medzi všetkými položkami, ktoré používateľ zadal cez dialóg.

5.4.6 Hlavné metódy

Trieda Handler.java obsahuje aj ďalšie metódy, ktoré tvoria jadro funkcionality IFO a narábajú s položkami, súbormi a kolekciami.

Sú to nasledujúce metódy:

- addTagsToFiles - pridá tagy do zvolených súborov
- addDescriptionToFiles - pridá popis do zvolených súborov
- removeDescriptionFromFiles - odstráni popis zo všetkých zvolených súborov
- copyFile - využíva Java metódu *Files.copy* na skopírovanie súborov zo zdrojovej cesty do cieľovej, vloží nový súbor do databázy a kolekcií a prekopíruje aj tagy a popis
- createAnEmptyCollection - vytvorí prázdny kontajner na súbory
- copyOnlyCollection - skopíruje celú kolekciu aj so súbormi v databáze
- deleteACollection - vymaže kolekciu
- renameACollection - premenuje kolekciu

- addFilesToCollection - pridá položky do kolekcie
- removeFilesFromCollection - odstráni položky z kolekcie
- copyFilesFromColToCol - skopíruje položky z jednej kolekcie do druhej kolekcie
- moveFilesBetweenCollections - premiestni položky z jednej kolekcie do druhej kolekcie
- deepCopyCollections - vytvorí hlbokú kópiu z kolekcií (vytvorí sa v pamäti nanovo)
- moveFile - využije Java metódu *Files.move* na premiestnenie súborov na disku

5.5 FileExtensions.java

Príjemnejšie ako otvárať a parsovať ďalší súbor (okrem databázového) bolo vytvoriť statickú mapu, v ktorej je zadefinovaná istá časť súborových prípon a kolekcií, do ktorých patria. Táto trieda sa volá hneď pri štarte IFO a zotrváva v pamäti až pokým nie je vypnutý - všetky nové položky pridané do databázy sa totiž automaticky ukladajú do kolekcií podľa tejto mapy.

Archives	Documents	Audio	Video	Data	Executables
zip	docx	mp3	avi	pdf	exe
7z	doc	wav	mp4	xls	msi
tar	txt	aac	mov	csv	bin
rar	rtf	wma	flv	ini	app
gz	odt	flac	mpg	html	dmg

Tabuľka 5.1: FileExtensions.java

5.6 Views

Používateľské rozhranie aplikácie je zabezpečené knižnicou JavaFX. Prvé, jemné využitia boli v triede Handler, pri dialógoch. Tieto dialógy sú však napevno naprogramované triedy. Okrem nich sa GUI skladá z ôsmich vlastných *.fxml* súborov a ich controllerov. Pár najzaujímavejších je stručne popísaných v tejto sekcii.

5.6.1 MainMenu

MainMenu.fxml a *MainMenuController.java* sú dva súbory vytvárajúce hlavné, najvýužívanejšie, menu programu. Controller na začiatku upraví tlačidlá na lište a vytvorí kontextové menu, ktoré sa zobrazuje pri kliknutí na kolekcie a položky. Obe menu obsahujú pevný počet položiek, ktoré sú klikatelné relatívne k vybraným riadkom, alebo počtom vybraných riadkov. Klikanie na vybrané tlačidlá na ovládacej lište sa taktiež determinuje podľa typu vybraného riadku/riadkov, respektíve ich počtu.

Náhľad na kolekcie aj položky je vyriešený cez JavaFX triedu ListView. Kolekcie sa zobrazia pri štarte programu a položky v jednotlivej kolekcii po kliknutí na ňu.

MainMenuController obsahuje aj inicializáciu všetkých ostatných controllerov a metódy využívajúce *Handler* popísané v sekcií 5.4.

Záver

V bakalárskej práci som sa zaoberal vytvorením plnohodnotného informačného systému väčšieho rozsahu, ktorý umožňuje používateľovi usporadúvať a kategorizovať súbory, ktoré si sám zvolí, za pomoci tagov, popisu, filtrovania a ďalším dvom typom vyhľadávania.

Pri vypracovaní som vytváral vlastnú hierarchiu dátových štruktúr, konkrétne vlastnú databázu a položky v nej. Položky sú zvolené súbory a kolekcie, čo sú nádoby súborov. Okrem ďalších vlastných dátových štruktúr som sa pri robení práce stretol s komplikáciami s konkurenciou pri využívaní Java knižníc na manipulovanie so súbormi a zapisovaním veľkej databázy do súboru. Počas návrhu dátových štruktúr som ich implementáciu testoval.

Pri tvorení používateľského rozhrania som dbal na jeho jednoduchosť a použiteľnosť. Ovládanie programu je zabezpečené tromi cestami - ovládacia lišta, horné menu a aj pravý klik myšou.

Do tejto práce som išiel s nadšením, ktoré mi vydržalo po celý čas písania kódu. Bola výzva vymýšľať a optimalizovať algoritmy a vlastné dátové štruktúry, ktorých využitie v praxi môžem ihneď vidieť.

Na začiatku prebiehal rozsiahly výber technológií, respektíve nástrojov na tvorbu tejto práce. Hlavné nároky bola jednoduchosť, keďže komplexita mala prísť až v neskorších, vyšších fázach práce a prepracovaná dokumentácia.

Po zhromažďovaní technológií nasledoval jednoduchý prototyp bez používateľského rozhrania a jeho následné testovanie. Tvorba GUI bola náročná už len z toho, že to bolo JavaFx, aj keď lepšia alternatíva na tvorbu grafických interfejsov v Jave neexistuje.

Nadstavba na túto prácu by mohla obsahovať:

- komplexnejšia modularita - možnosť vytvárať z kolekcií archívy a prenášať ich na externé HDD/USB a iné médiá
- možnosť používateľovi zadefinovať vlastné súborové prípony a ich zaraďovanie do kolekcií

Literatúra

[1] Css. https://www.w3.org/Style/CSS/#specs. Prístup: 22.05.2016.

[2] Gson. https://github.com/google/gson. Prístup: 22.05.2016.

[3] Java špecifikácia. https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf. Prístup: 22.05.2016.

[4] Java virtual machine. http://www.artima.com/insidejvm/ed2/jvm.html. Prístup: 22.05.2016.

[5] Javafx. http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html. Prístup: 22.05.2016.

[6] Javafx ako náhrada za swing. http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#6. Prístup: 22.05.2016.

[7] Jre a jcl. http://docs.oracle.com/javase/6/docs/technotes/tools/findingclasses.html. Prístup: 22.05.2016.

[8] Json. http://json.org/. Prístup: 22.05.2016.

[9] Lambda výrazy. https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html. Prístup: 22.05.2016.

[10] L.i.s.t. http://dai.fmph.uniba.sk/~petrovic/th13/Jursa_list.pdf. Prístup: 22.05.2016.

[11] Musicbrainz picard. http://www.pcworld.com/article/232471/musicbrainz_picard.html. Prístup: 20.01.2016.

[12] Obsah jdk. http://www.oracle.com/technetwork/java/javase/jdk-8-readme-2095712.html#contents. Prístup: 22.05.2016.

[13] Python. https://www.python.org/about/. Prístup: 22.05.2016.

[14] Stuff organizer. http://stufforganizer.sourceforge.net/. Prístup: 22.05.2016.

[15] Tagspaces. `https://www.tagspaces.org/`. Prístup: 22.05.2016.

[16] Write once, run anywhere. `http://www.computerweekly.com/feature/Write-once-run-anywhere`. Prístup: 22.05.2016.

[17] Xml. `https://www.w3.org/TR/REC-xml/`. Prístup: 22.05.2016.

[18] Jozef Piaček a Miloš Kravčík. *FILIT - Otvorená filozofická encyklopédia*. 1999. `http://dai.fmph.uniba.sk/~filit/fvo/organizacia.html`, dátum prístupu: 19.01.2016.

[19] Weishi Zhang. *Formal Description and Development of Graphical User Interfaces*. Herbert Utz Verlag, 1996. strana 21.