FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Multiplatformní nástroj pro správu souborových úložišť
Student:	Bc. Jan Zvěřina
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem této práce je navrhnout a implementovat open-source nástroj, který dokáže snadno evidovat různá souborová úložiště (pevné disky, flash disky, cloudové úložiště, paměťové karty a další) včetně jejich obsahu a dalších uživatelských parametrů pro snadné vyhledávání i ve stavu, kdy není úložiště zrovna připojeno. Při vývoji musí být uplatněn klasický cyklus vývoje software: analýza, návrh, implementace vč. testování a dokumentace:

- Proveďte analýzu a popište možnosti souborových úložišť a práce s nimi. Sestavte dle analýzy a zadání požadavky na vyvíjený nástroj.
- Zpracujte stručnou rešerši nástrojů usnadňující práci se souborovými úložišti.
- Navrhněte multiplatformní nástroj naplňující stanovené požadavky, které bude snadno rozšiřitelné.
- Implementujte nástroj v programovacím jazyce Java dle návrhu a řádně jej otestujte a zdokumentujte. Volby dalších technologií (například framework či jiné knihovny) zdůvodněte.
- Zhodnoťte přínosy nástroje a jeho možný další rozvoj.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 9. října 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Multiplatformní nástroj pro správu souborových úložišť

Bc. Jan Zvěřina

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

28. května 2020

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Markovi Suchánkovi za konzultace během vytváření diplomové práce. Dále bych chtěl poděkovat všem, kteří se mnou vyplnili úvodní dotazník a zodpověděli další dotazy. Stejně tak bych chtěl poděkovat všem, kteří podstoupili uživatelské testování výsledného programu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 28. května 2020 ……

Odkaz na tuto práci

Zvěřina, Jan. *Multiplatformní nástroj pro správu souborových úložišť.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Práce je zaměřena na návrh a implementaci programu, který by umožňoval evidenci souborů v paměťových zařízeních. Program musí implementovat také vyhledávání, a to i v případě, kdy je paměťové zařízení odpojeno od počítače.

Práce obsahuje analýzu existujících programů a průzkum s potenciálními uživateli. Na základě výstupů rešerše byly vytvořeny požadavky na program a případy užití. Návrh programu obsahuje drátový model, architekturu a relační databázový model. Z návrhu vychází volba použitých technologií pro implementaci programu. Implementace věrně dodržela návrh. Program je napsán v programovacím jazyce Java a jedná se o desktopovou aplikaci s grafickým uživatelským rozhraním.

Finální program byl podroben uživatelskému testování, ze kterého vzešel seznam potřebných změn a úprav. Program splňuje všechny cíle stanovené zadáním a všechny funkční a nefunkční požadavky.

Klíčová slova paměťové médium, index souborů, vyhledávání souborů, připojená zařízení, odpojená zařízení, Java, desktopová aplikace

Abstract

The thesis is focused on the design and implementation of a program that would allow the user to index files of memory devices. The program must also allow a file search, even if the memory device is disconnected from the computer.

The thesis contains an analysis of existing programs and a survey with potential users. Requirements for the program and use cases were created based on the results of that analysis. The design of the program includes wireframes, architecture, and a relational database model. Technologies to implement the program were chosen based on the design. The implementation faithfully adhered to the design. The program is written in the Java programming language as a desktop application with a graphical user interface.

The finalized program was subjected to user testing, which resulted in a list of needed changes and modifications. The program meets all the objectives set by the assignment and all functional and nonfunctional requirements.

Keywords memory device, index of files, file search, connected devices, disconnected devices, Java, desktop application

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Fotografové, kameramani, střihači videí, grafici, vývojáři her a lidé v dalších podobných profesích mají jednotky či dokonce desítky paměťových karet, flash disků, externích disků či jiných paměťových médií. Na nich mají zálohy svých projektů, fotografií či videí. Každý z nich se pravděpodobně dříve či později setká s jedním z následujících problémů: „Kde mám zálohované fotky z Vánoc 2015?“ „Na kterém médiu mám uložený rok starý projekt, ve kterém chce zákazník provést úpravy?“ „Zkopíroval jsem svoji novou sérii fotek na zálohovací externí disk?“ „Mám vytvořenou zálohu nové verze mé hry?“ Takové otázky mohou vést ke zběsilému zapojování paměťových zařízení do počítače, dokud dotyčný nenalezne to, co hledá.

Nemusí tomu ale tak být, pokud bude mít program, ve kterém eviduje aktuální obsah všech paměťových zařízení. V takovém případě stačí v programu vyhledat soubor či adresář. Program zobrazí, na jakém paměťovém zařízení se nachází a následně může uživatel s jistotou k počítači připojit dané zařízení. Tím si ušetří spoustu času a nervů ze zbytečného hledání.

Cíl práce

Cílem práce je vytvořit program pro evidenci souborů na paměťových zařízeních. Bude umožněno vyhledávání souborů bez ohledu na to, zda je evidované paměťové médium připojeno k počítači. Podoba programu se bude řídit požadavky, které určí rešerše existujících programů a průzkum u potenciálních uživatelů. Program by měl být snadno rozšiřitelný a multiplatformní, napsaný v programovacím jazyce Java. Výsledná podoba programu bude otestována uživateli. Z výsledků se pak určí řešení nalezených chyb a závěr práce bude obsahovat možná rozšíření programu do budoucna.

Struktura práce

Diplomová práce se skládá z pěti kapitol a dvou příloh. První kapitola *Analytická část* se zabývá shrnutím základních pojmů, které se budou v práci objevovat. Dále obsahuje průzkum s potenciálními uživateli, rešerši programů řešících podobný problém a v neposlední řadě část s požadavky na systém.

Kapitola *Návrh programu* obsahuje návrhu programu na základě funkčních a nefunkčních požadavků. Návrh není závislý na platformě a použitých technologiích.

Třetí kapitola, *Technologie*, pojednává o výběru technologií pro implementaci a čtvrtá kapitola, *Implementace*, popisuje, jakým způsobem byla provedena tvorba programu (v závislosti na použitých technologiích).

Poslední pátá kapitola, *Testování*, popisuje, jakým způsobem bylo provedeno testování, včetně podrobného uživatelského testování. Na základě zjištěných informací obsahuje shrnutí s výpisem chyb a návrhem možných řešení.

V první příloze A je uveden kompletní drátový model aplikace a v druhé příloze je zobrazena výsledná podoba programu B.

Kapitola 1

Analytická část

V analytické části se zaměřuji na získání co nejvíce informací, které budou následně využity ke správnému návrhu programu. Nejprve vysvětlím několik základních pojmů a následně se zaměřím na průzkum u možných uživatelů, abych zjistil potřeby potenciálních zákazníků. V neposlední řadě analyzuji několik programů zabývajících se správou souborů či jejich vyhledáváním. Na základě výsledků z těchto částí sestavím funkční a nefunkční požadavky na vyvíjený program.

1.1 Vysvětlení základních pojmů

Pro začátek se sluší vypsat několik základních pojmů, abych vysvětlil jejich význam. U čtenářů by pak nemělo docházet ke zbytečnému nepochopení. Jsou zde uvedeny nejčastější a klíčové pojmy. Všechny další méně používané termíny budou popsány u prvního výskytu.

1.1.1 Paměťové médium

Paměťové médium je nosič datových informací. Každá informace musí být uložena na paměťové médium, aby se s ní mohlo manipulovat. Informace jsou na takových médiích ukládána bitově pomocí jedniček a nul.

Paměťových zařízení máme spousty, kromě pevných disků (HDD - *Hard Disk Drive*) s magnetickými kotouči existují zařízení používající flash paměť (SSD - *Solid State Drive*), která je oproti klasickým pevným diskům několikanásobně rychlejší, protože nepoužívá mechanické části. Dále existují externí paměťová média jako flash disky, SD karty, diskety, CD/DVD/Blu-ray disky apod. [1] V dalším textu budu paměťové médium nazývat také jako paměťové zařízení, datové médium, médium, úložiště, paměť apod.

1.1.2 Soubor a jeho metadata

Soubor je kolekce dat uložená jako celek, identifikovatelná jménem souboru a cestou k souboru. Může to být dokument, obrázek, video, audio, aplikace, skript atd. Soubory jsou uložené na paměťových médiích. [2]

Metadata jsou informace o souboru uložené se souborem. Slouží k popsání daného souboru. Obsahují jméno souboru, velikost, datum vytvoření, datum poslední změny, datum posledního přístupu, typ souboru, klíčová slova k popsání dokumentu atd. Metadat je nepřeberné množství. Například metadata pro soubor s písničkou mohou obsahovat jméno zpěváka, jméno alba a rok vydání. Fotografie obsahuje například datum vyfocení snímku a GPS lokaci, kde byla pořízena. Metadata také obsahují práva k souboru, určující, kdo soubor může číst, kdo ho může upravovat a kdo ho může spustit. [3]

1.1.3 Souborový systém

Na webové stránce *How-To Geek* [4] se lze dočíst, že souborový systém (anglicky *filesystem*) určuje způsob organizace a ukládání dat na paměťovém zařízení. Každé paměťové zařízení má jeden nebo více oddílů. Každý takový oddíl je formátován na určitý typ souborového systému. Souborový systém tedy slouží k separaci dat na disku do celků, kterým se říká soubory.

Kromě obsahu těchto souborů souborový systém ukládá také jejich metadata, jako jsou název, datum vytvoření, práva k souboru apod. Souborový systém také poskytuje index, což je seznam všech souborů a jejich umístění na paměťovém zařízení. Nejčastějším příměrem z reálného světa je kartotéka. Operační systém počítače musí rozumět souborovému systému, aby mohl pracovat s obsahem paměťového média. Pokud mu nerozumí, tak si musí nainstalovat ovladač. V případě neexistence ovladače pro daný operační systém nelze médium používat.

Souborových systémů je mnoho a jejich zaměření je různé. Mohou být zaměřeny na rychlost čtení/zápisu/vyhledávání, bezpečnost dat, nepoškození dat při zápisu apod. Různé souborové systémy podporují větší či menší kapacity paměťových médií. Vývojáři operačních systému si zpravidla vytvářejí své vlastní souborové systémy. Popis několika nejzákladnějších souborových systému je uveden níže.

FAT32

Souborový systém FAT32 (*File Allocation Table*) byl využívaný ve starších operačních systémech Windows. Stále je ale využívaný u menších externích médií (maximální velikost souboru 4 GiB). Používá se hlavně kvůli zpětné kompatibilitě s různými staršími zařízeními jako digitální kamery, herní konzole, set-top boxy apod. [4]

NTFS
Moderní verze operačního systému Windows (od Windows XP) používají NTFS (*New Technology File System*). Teoretická maximální velikost jednoho souboru je 16 EiB (mínus 1 KiB), ale implementováno je 16 TiB (mínus 64 KiB). Používá se jak pro interní, tak pro externí média. [4]

HFS+
HFS+ (*Hierarchical File System*) je souborový systém využívaný v operačních systémech macOS od společnosti Apple. Jako HFS+ by měl být naformátovaný i externí disk, pokud je požadavek využít jej pro zálohování programem *Time Machine*. Stejný požadavek platí i pro některé další aplikace. [4]

ext4
Linuxové distribuce používají nejnovější verzi *extended filesystem* souborového systému, ext4, zatímco je ve vývoji nový Btrfs (*Better file system* - lepší souborový systém), který ext4 jednou pravděpodobně nahradí jako výchozí souborový systém pro linuxové distribuce. Ext4 je oproti svým předchůdcům rychlejší a modernější. [4]

1.1.4 Indexování

Z definice na stránkách Microsoftu pro podporu systému Windows [5] je indexování „proces prohlížení souborů, e-mailových zpráv a dalšího obsahu na vašem počítači a katalogizace jejich informací, například obsažených slov a metadat. Když prohledáváte počítač po dokončení indexování, používá počítač index termínů k rychlejšímu vyhledání výsledků. Při prvním spuštění může indexování trvat až několik hodin. Pak bude indexování spuštěné na pozadí, zatímco budete počítač používat, a budou se znovu indexovat pouze aktualizovaná data.“

1.2 Průzkum u možných uživatelů

Před vytvořením požadavků na program jsem provedl průzkum u potenciálních uživatelů. Dle popisu programu v zadání a na základě mého předpokladu, že se v těchto profesích využívá externích úložišť, jsem jako cílovou skupinu určil fotografy, editory videí, herní vývojáře a programátory.

Vytvořil jsem dotazník, ve kterém jsem nejprve popsal základní představu o podobě vyvíjeného programu a následně nechal představitele jednotlivých skupin vyplnit množinu sedmi otázek. Dotazník jsem se tedy nesnažil poslat co nejvíce respondentům, ale poslal jsem ho pěti vybraným jednotlivcům se zohledněním cílové skupiny uživatelů. Po vyplnění dotazníku jsem s každým provedl krátký rozhovor (osobně či přes internet), abych zjistil podrobnosti k jednotlivým odpovědím a případně si vyjasnil nejasnosti.

Před posláním dotazníku prvnímu respondentovi jsem dotazník nechal zkontrolovat dvěma nezainteresovanými lidmi, abych předešel případnému nepochopení otázek či jejich dvojznačnosti. Dotazník jsem na základě jejich zpětné vazby upravil. Dva respondenti dali souhlas s použitím jejich reálného jména, zbylí tři jsou uvedeni anonymně.

1.2.1 Dotazník s uživatelem č. 1

- **Jméno a příjmení:** Jan Svoboda
- **Práce/Zájmy:** Profesionální fotograf

a. Využil(a) byste aplikaci pro vyhledávání v paměťových médiích, včetně těch nepřipojených k počítači?

Ano

b. Je pro vás důležité evidovat kromě vlastností souborů (název, typ, vlastník, práva) i obsah souborů (pro vyhledávání v textu samotných dokumentů)?

Ne

c. Jaké možnosti nastavení evidence souborů byste využil(a)?

- Volbu, která paměťová zařízení se mají evidovat
- Zakázání evidence určitých složek
- Volbu, které typy souborů se mají evidovat
- Nastavení volby metadat, která se mají evidovat (název souboru, typ, vlastník, práva, ...)
- Pojmenování či označení paměťových zařízení pro lepší přehlednost

d. Jaké další funkce by měl program mít?

Neodpovězeno.

e. Vidíte smysl v evidování cloudových úložišť jako OneDrive, Dropbox, Disk Google apod.?

Ano.

f. Jaký operační systém používáte?

macOS

g. Jaké programy využíváte pro správu souborů a jejich vyhledávání?

Finder, Lightroom

Jan Svoboda je profesionální fotograf mající nespočetné množství externích disků a paměťových karet. Účel programu ho nadchl, protože se s tímto problémem shledává velmi často. Pokaždé, když něco hledá, musí připojovat velké množství disků, kterých má plný šuplík. Fulltextové vyhledávání mu nepřijde důležité, jeho obvyklý problém je vyhledání složky s fotografiemi například podle jména modelky. Představuje si, že po zadání výrazu „Katka“ mu program zobrazí všechny složky se jménem Katka, na jakých harddiscích tyto složky jsou a cestu, která k těmto složkám vede.

1.2.2 Dotazník s uživatelem č. 2

- **Jméno a příjmení:** Nikolai Lazarev
- **Práce/Zájmy:** Režisér, kameraman, střihač, streamer

a. Využil(a) byste aplikaci pro vyhledávání v paměťových médiích, včetně těch nepřipojených k počítači?

Ne

b. Je pro vás důležité evidovat kromě vlastností souborů (název, typ, vlastník, práva) i obsah souborů (pro vyhledávání v textu samotných dokumentů)?

Ne

c. Jaké možnosti nastavení evidence souborů byste využil(a)?

- Volbu, které typy souborů se mají evidovat
- Nastavení volby metadat, která se mají evidovat (název souboru, typ, vlastník, práva, ...)
- Pojmenování či označení paměťových zařízení pro lepší přehlednost

d. Jaké další funkce by měl program mít?

Vyhledávání podle data, zakázání evidence XML[1] souborů apod.

e. Vidíte smysl v evidování cloudových úložišť jako OneDrive, Dropbox, Disk Google apod.?

Ne.

f. Jaký operační systém používáte?

Windows 10

g. Jaké programy využíváte pro správu souborů a jejich vyhledávání?

Windows File Explorer, locate32

[1]Extensible Markup Language - značkovací jazyk pro ukládání dat, který je snadno čitelný jak pro člověka, tak pro stroj [6]

Nikolai Lazarev je profesionální kameraman a editor videí. Videa vytváří pro nespočet společností a sám si je režíruje. Přestože jsem jeho profesi zcela bez váhání zařadil do cílové skupiny, tak by údajně takový program nevyužil. Většinu svých projektů zpracuje a dokončí během několika dnů. Po spokojeném převzetí výsledného videa zákazníkem všechny podklady po několika měsících odstraní z počítače. Jeden projekt může zabírat desítky až stovky gigabytů a není v jeho silách si všechny materiály ukládat. Všechny projekty má navíc uložené na pevných discích v počítači a nevyužil by funkci vyhledávání v odpojených zařízeních. Nicméně odpověděl, že mezi jeho kolegy existuje několik lidí, kteří si svá videa zálohují na externí disky, a to jak dokončená videa, tak rozpracovaná. Pro ně by tento program mohl být vhodný.

Přestože nejevil o program zájem, poprosil jsem ho, zda by se mohl vyjádřit k celkové podobě programu za předpokladu, že by ho využil. Vyhledávání v textu dokumentů by v naprosté většině případů nevyužil, ale bylo by hezké takovou možnost občas mít. Chtěl by vyhledávat podle poznámek a tagů v metadatech souborů, do kterých si občas zapíše, o čem video je.

V dotazníku u třetí otázky nezaškrtl možnost „Volba, které typy souborů se mají evidovat“, ale jako další funkci od programu žádá „zakázání evidence XML souborů“, což spadá do zmíněné volby. Proto jsem v jeho dotazníku zaškrtl i tuto volbu.

1.2.3 Dotazník s uživatelem č. 3

- **Jméno a příjmení:** Dotazovaný nechtěl být jmenován
- **Práce/Zájmy:** Zakladatel a jednatel firmy, nadšenec do fotografování a akčních sportů

a. Využil(a) byste aplikaci pro vyhledávání v paměťových médiích, včetně těch nepřipojených k počítači?

Ano

b. Je pro vás důležité evidovat kromě vlastností souborů (název, typ, vlastník, práva) i obsah souborů (pro vyhledávání v textu samotných dokumentů)?

Ano

c. Jaké možnosti nastavení evidence souborů byste využil(a)?

- Volbu, která paměťová zařízení se mají evidovat
- Nastavení volby metadat, která se mají evidovat (název souboru, typ, vlastník, práva, ...)
- Pojmenování či označení paměťových zařízení pro lepší přehlednost

d. Jaké další funkce by měl program mít?

Neodpovězeno.

e. Vidíte smysl v evidování cloudových úložišť jako OneDrive, Dropbox, Disk Google apod.?

Ne.

f. Jaký operační systém používáte?

Windows 10

g. Jaké programy využíváte pro správu souborů a jejich vyhledávání?

Windows File Explorer

Respondent číslo tři nechtěl být jmenován, tak o něm zmíním jen to, že je programátor s láskou pro fotografování. Přiznal, že má svá data roztroušena skrz několik externích disků a desítku paměťových karet. Bez vyzkoušení výsledného programu si ale není jistý, zda by pro něho jeho používání mělo nějaký benefit. Požádal mě ale, abych mu výsledný program zaslal na vyzkoušení, což je dle mě dobré znamení.

1.2.4 Dotazník s uživatelem č. 4

- **Jméno a příjmení:** Dotazovaný nechtěl být jmenován
- **Práce/Zájmy:** Programátor

a. Využil(a) byste aplikaci pro vyhledávání v paměťových médiích, včetně těch nepřipojených k počítači?

Ne

b. Je pro vás důležité evidovat kromě vlastností souborů (název, typ, vlastník, práva) i obsah souborů (pro vyhledávání v textu samotných dokumentů)?

Ano

c. Jaké možnosti nastavení evidence souborů byste využil(a)?

- Volbu, která paměťová zařízení se mají evidovat
- Pojmenování či označení paměťových zařízení pro lepší přehlednost

d. Jaké další funkce by měl program mít?

Neodpovězeno.

e. Vidíte smysl v evidování cloudových úložišť jako OneDrive, Dropbox, Disk Google apod.?

Ne.

f. Jaký operační systém používáte?

macOS

g. Jaké programy využíváte pro správu souborů a jejich vyhledávání?

Finder či další výchozí správce souborů v jiných operačních systémech

Ani respondent číslo čtyři nechtěl být jmenován, nicméně se jedná o programátora, který v minulosti vytvářel zábavná videa. Otázky v dotazníku mu přišli pochopitelné. Má většinu svých kódů uložených ve verzovacích kontrolních systémech[2] jako Git[3] nebo SVN[4]. Osobně nepotřebuje ukládat svá data na externích discích, takže by program nevyužil.

1.2.5 Dotazník s uživatelem č. 5

- **Jméno a příjmení:** Dotazovaný nechtěl být jmenován
- **Práce/Zájmy:** Herní vývojář

a. Využil(a) byste aplikaci pro vyhledávání v paměťových médiích, včetně těch nepřipojených k počítači?

Ano

b. Je pro vás důležité evidovat kromě vlastností souborů (název, typ, vlastník, práva) i obsah souborů (pro vyhledávání v textu samotných dokumentů)?

Ne

c. Jaké možnosti nastavení evidence souborů byste využil(a)?

- Volbu, která paměťová zařízení se mají evidovat
- Zakázání evidence určitých složek
- Volbu, které typy souborů se mají evidovat
- Nastavení volby metadat, která se mají evidovat (název souboru, typ, vlastník, práva, ...)
- Pojmenování či označení paměťových zařízení pro lepší přehlednost

[2] Systémy pro správu zdrojových kódů programů a jejich změn [7]
[3] *open-source* distribuovaný verzovací systém [8]
[4] Apache Subversion, *open-source* centralizovaný verzovací systém [9])

d. Jaké další funkce by měl program mít?

Cloudové sdílení indexu mezi více vývojáři. Vyhledávání souborů podle metadat.

e. Vidíte smysl v evidování cloudových úložišť jako OneDrive, Dropbox, Disk Google apod.?

Ano.

f. Jaký operační systém používáte?

Windows 10

g. Jaké programy využíváte pro správu souborů a jejich vyhledávání?

Windows File Explorer, Total Commander, Everything

Respondent číslo pět nechtěl být jmenován. Živí se mimo jiné jako vývojář Indie[5] videoher. Program a jeho smysl se mu velmi líbil, chtěl by ho použít v rámci vývoje her pro kontrolu stavu záloh na externích discích. Jako zajímavou funkci do budoucna by uvítal cloudové sdílení indexu, čímž myslí index nahraný na úložišti přístupném přes internet, sdíleném mezi více uživateli, vždy v aktuální podobě [11]. Využil by možnost vyhledávání souborů podle metadat. V rámci zachování kompaktní velikosti indexu nepotřebuje fulltextového vyhledávání.

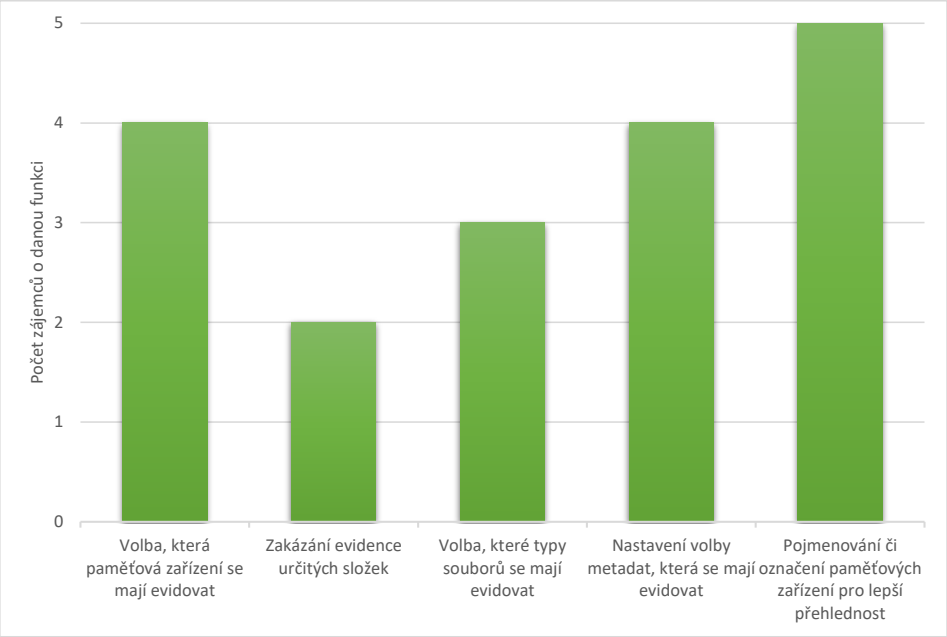
1.2.6 Shrnutí výsledků průzkumu mezi uživateli

Průzkum mezi pěti vybranými uživateli mě přinutil přehodnotit velikost mé cílové skupiny. Mnou určená cílová skupina zahrnovala fotografy, editory videí, herní vývojáře a programátory. Tři lidé z dotazovaných odpověděli, že by program využili (dva fotografové a jeden herní vývojář). Oba fotografové potvrdili, že se s problémem vyhledávání v externích paměťových zařízeních setkávají. Herní vývojář by program také využil, protože si na externích discích zálohuje verze své hry včetně všech herních assetů[6].

Programátor a editor videí odpověděli, že by program nejspíš nevyužili. Programátora mohu z cílové skupiny odstranit. Je pravda, že většina programátorů má své projekty uložené ve verzovacích systémech jako Git či SVN a zálohy na externích discích nepotřebují. Editory videí bych zařadil do sekundární cílové skupiny, protože dotazovaný respondent by sice program nevyužil, ale zná kolegy, kteří by ho využít mohli. Moje primární cílová skupina tedy jsou fotografové a herní vývojáři, sekundární editoři videí.

[5] Označení pro nezávislé počítačové hry; nejsou financované herním vydavatelem a většinou na nich pracují jednotky lidí [10]

[6] Herní assety zahrnují vše, co se ve hře může objevit (zvuky, textury, 3D modely, ikony apod.) [12]



Obrázek 1.1: Graf s počtem respondentů, kteří by využili dané základní funkce

Třetí otázka v dotazníku se týkala základních funkcí, které by respondenti od programu požadovali. Výsledky lze vidět v grafu 1.1. Pouze dva dotazovaní navrhli další funkce, které by program mohl mít. Hernímu vývojáři by se líbilo cloudové sdílení indexu mezi více vývojáři a vyhledávání souborů podle metadat. Vyhledávání podle metadat by využil i editor videí. Vyhledávání v obsahu souborů či evidence cloudových úložišť jako Dropbox, OneDrive apod. nejsou moc žádané funkcionality. Tyto funkce žádali pouze dva respondenti. Funkce tak můžeme rozdělit do tří kategorií podle priority, s přihlédnutím ke složitosti implementace:

1. Nejvyšší priorita
    - Pojmenování či označení paměťových zařízení pro lepší přehlednost
    - Volba, která paměťová zařízení se mají evidovat
    - Nastavení volby metadat, která se mají evidovat
2. Střední priorita
    - Volba, které typy souborů se mají evidovat
    - Zakázání evidence určitých složek
    - Vyhledávání souborů podle metadat

3. Nízká priorita
    - Cloudové sdílení indexu
    - Evidence obsahu souborů
    - Vyhledávání v obsahu souborů
    - Evidence cloudových úložišť

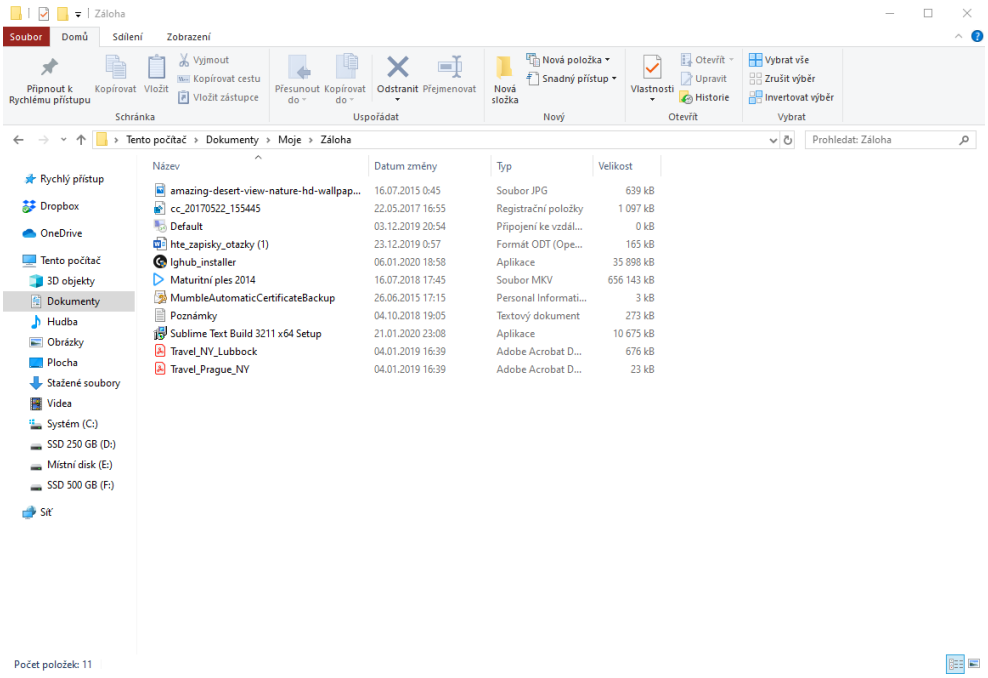
V rámci diplomové práce budu brát v potaz pouze funkce z kategorií s nejvyšší a střední prioritou. Funkcionality s nízkou prioritou mohou sloužit k případnému budoucímu rozšíření aplikace. Z dotazníku vyplynulo, že vyvinout multiplatformní aplikaci je důležité, protože respondenti používají jak Windows 10, tak macOS, případně různé distribuce Linuxu jako sekundární operační systém. Většina respondentů používá výchozí správce souborů jako jsou Windows File Explorer ve Windows 10 či Finder v macOS. Herní vývojář dále používá Total Commander či Everything pro vyhledávání souborů. Nejpoužívanější správce a vyhledávače souborů si podrobněji popíšeme.

1.3 Rešerše existujících programů

Existují spousty programů zaměřujících se na správu souborových úložišť. Tyto programy se ale většinou zaměřují na správu a nastavení těchto úložišť, čímž myslím rozdělení média na oddíly, změnu souborového systému, formátování apod. Tento druh funkcí můj program podporovat nebude a tak je zbytečné tyto programy analyzovat. Navrhovaný program se bude zaměřovat na evidenci obsahu médií a vyhledávání. Proto analyzuji programy, kterým se říká správci souborů (*file managers*).

Správce souborů je program, který slouží ke správě, procházení a zobrazení obsahu médií. Umožňuje procházet adresáře, otvírat, kopírovat, přesouvat, vyhledávat soubory, spouštět aplikace a spoustu dalších funkcí [13]. Dále chci analyzovat programy určené výhradně k vyhledávání v médiích a jejich indexování.

Z průzkumu vyplynulo několik správců a vyhledávačů souborů, které respondenti používají. Proto několik z nich (plus některé další) popíši v této sekci. Rešerši existujících programů je dobré udělat, abych zjistil, zda se vůbec vyplatí vyvinout další indexovací a vyhledávací nástroj. Další výhodou provedení rešerše je to, že získám představu o zvyklostech v návrhu uživatelského rozhraní a o funkcích, které by program měl mít. Takto získané informace mohu použít pro lepší návrh programu.



Obrázek 1.2: Rozhraní Průzkumníka souborů ve Windows 10

1.3.1 Windows File Explorer

Windows Explorer, česky Průzkumník Windows, od Windows 10 nazývaný File Explorer (Průzkumník souborů), je nativní správce souborů v operačních systémech Windows od Microsoftu. Jeho první verze se vyskytla ve Windows 95 [14]. Rešerši provedu ve verzi Průzkumníka souborů pro Windows 10, který má velice přehledné grafické rozhraní.

Obrazovka programu je rozdělena do tří částí. Většinu prostoru obrazovky zabírá centrální část sloužící k zobrazení souborů v aktuální složce. Soubory lze zobrazit v mřížce pomocí či jako seznam. Typ zobrazení lze zvolit ve funkční části aplikace či kliknutím pravým tlačítkem myši do okna. Zobrazení ve formě seznamu je nejzajímavější. Buď lze soubory zobrazit bez jakýchkoliv dodatečných informací, kromě jména a ikony, či s podrobnostmi. Při zobrazení podrobností si lze vybrat, které všechny informace o souborech se mají zobrazit: název, typ, datum změny, velikost, autoři apod. Jakýkoliv další typ informace tvoří nový sloupec. Nad výpisem souborů je lišta s cestou do aktuálního umístění od kořenového adresáře. Lze na ni kliknout a vynořit se do jakékoliv úrovně v cestě. Vpravo od ní je pole pro vyhledávání. Po zadání hledaného výrazu se zobrazí výsledky vyhledávání. Lze nastavit, které složky se mají prohledat. Rychlost vyhledávání je pomalá, ale může se zvýšit, pokud se povolí indexování ve Windows. Více informací o indexování v systému Windows je uvedeno níže, v podkapitole 1.3.1.1.

Levou část okna zabírá rozcestník, ze kterého se lze dostat jedním kliknutím k nejčastěji navštěvovaným složkám (v sekci Rychlý přístup) či do cloudových úložišť jako Dropbox nebo OneDrive (záleží, co používáte). Dále je zde sekce Tento počítač obsahující všechny připojené jednotky a rozcestník do různých kategorií složek jako Dokumenty či Stažené soubory.

Horní část obrazovky obsahuje funkce, které je možné se soubory či zobrazením souborů provádět. Funkce jsou rozděleny do karet. Na první kartě Domů se vyskytují základní operace se soubory jako kopírování, přesouvání či přejmenování souborů. Karta Sdílení obsahuje sadu funkcí pro sdílení souborů pomocí mailu, aplikace Skype atd. Dále umožňuje komprimovat skupinu souborů do archivu ZIP a tisk dokumentů či fotek. Karta Zobrazení obsahuje funkce pro řazení a různé způsoby zobrazení souborů v dané složce. Řadit lze dle názvu souboru, typu, velikosti, data vytvoření, značky atd. V této kartě lze navolit zobrazení skrytých souborů, zobrazení přípon souborů či zobrazení zaškrtávacích políček vedle souborů pro snazší označování více souborů najednou. V sekci Podokna na této kartě lze nastavit zobrazení či skrytí rozcestníku vlevo či zobrazení podokna vpravo, které ukazuje informace o vybraném souboru či jeho náhled.

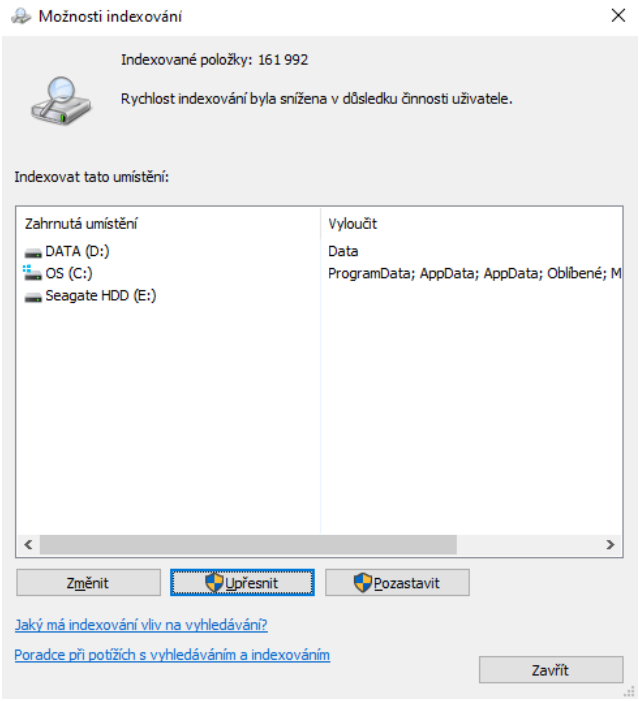
Některé typy souborů mohou vyvolat další kartu s nástroji pro daný typ souboru. Například vybrání spustitelné aplikace vyvolá nabídku Nástroje aplikace, přes kterou lze spustit aplikaci jako správce nebo lze odstranit problémy se spuštěním. Při vybrání obrázku se zobrazí karta Nástroje obrázků, ve které lze obrázek otočit či nastavit jako pozadí. Všechny tyto možnosti většinou pokrývají část funkcí, které lze vyvolat kliknutím pravým tlačítkem myši na soubor.

Při zahájení vyhledávání se zobrazí karta Hledání, ve které lze nastavit, zda se má prohledat pouze aktuální složka, všechny podsložky či celý počítač od kořenového umístění Tento počítač. Dále lze hledání upřesnit pomocí filtrů či zobrazit historii předchozích hledání. Jak se uvádí na webu *Lifewire* [15], lze aktuální vyhledávání uložit jako šablonu do speciálního XML souboru s příponou search-ms. Spuštěním souboru lze vyhledávání zopakovat. [15]

Okno s průzkumníkem je plně responzivní a nevadí mu jakékoliv roztažení či zúžení. Při moc velkém zúžení se schovají panely s rozcestníkem a funkcemi, ale soubory lze stále procházet. Dřívější verze průzkumníka měly jistá omezení, ale verze ve Windows 10 působí skvěle. Jako jediný nedostatek lze uvést, že není možné zobrazení dvou adresářů zároveň, ale to lze vyřešit dvěma okny vedle sebe, či použitím programu Total Commander, který byl speciálně navržen pro tento případ.

1.3.1.1 Indexování souborů v MS Windows

Pro rychlejší získávání výsledků při hledání souborů skrz *Průzkumník souborů* se v operačním systému Windows využívá indexování souborů. Na webových stránkách Microsoftu pro podporu systému Windows [5] se uvádí, že

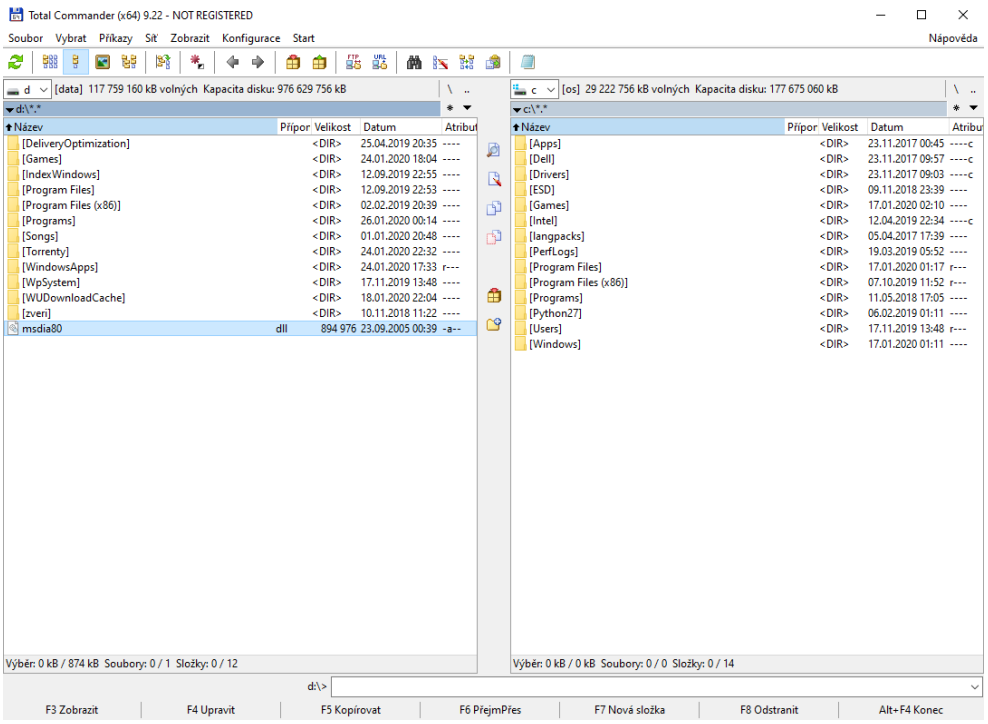


Obrázek 1.3: Možnosti indexování ve Windows 10

se zapnutým indexováním a plně sestaveným indexem lze výsledky na běžné dotazy typu „Dovolená Chorvatsko“ či „Všechny soubory vytvořené 15. 2. 2019“ získat během zlomku sekundy. Bez indexování může vyhledávání trvat i několik minut. Je pravidlem, že velikost indexu je menší než 10 % velikosti indexovaných souborů. Toto pravidlo má výjimku. Pokud indexujete spoustu velmi malých souborů (menších než 4 kB), tak se velikost indexu dramaticky zvýší. Mnoho integrovaných aplikací v systému Windows index používá: *Průzkumník Windows, Fotky, Groove, Edge* nebo *Outlook*, který ho používá při vyhledávání v e-mailech. Index ale mohou využívat i aplikace třetích stran. Výkon takových aplikací ovlivňuje, zda je indexování zapnuté.

Nastavení se provádí v programu *Možnosti indexování*. Na úvodní obrazovce 1.3 lze vidět seznam všech jednotek v počítači, které jsou indexovány. Indexovat lze jak interní pevné disky, tak externí paměťová média. Po odpojení indexovaného externího média se při vyhledávání nezobrazují výsledky pro odpojené médium, nicméně systém Windows si pamatuje obsah tohoto zařízení, včetně nastavení indexování. Po zpětném připojení se zobrazují výsledky i z tohoto média, zatímco si systém Windows aktualizuje index podle změn.

Grafické rozhraní obsahuje pod výpisem indexovaných médií tři tlačítka: *Změnit*, *Upřesnit* a *Pozastavit*. Tlačítkem Pozastavit se zastaví indexování, dokud se znovu nespustí, či do restartu počítače. Po stisknutí tlačítka Změnit se



Obrázek 1.4: Rozhraní programu Total Commander, verze 9.22

zobrazí okno s nabídkou všech připojených paměťových zařízení. Zaškrtnutím neindexovaných médií lze spustit indexování. Z indexování lze vyloučit libovolné složky na médiu. Tlačítko Upřesnit zobrazí okno, ve kterém lze nastavit, do jaké složky a na jakém médiu se má index vytvořit. Dále lze zvolit, zda se mají indexovat šifrované soubory a zda považovat podobná slova s diakritikou za jiná slova. V případě potíží lze vyvolat nové sestavení indexu. V kartě Typy souborů lze nastavit, které typy souborů se mají indexovat a zda indexovat pouze vlastnosti souborů bez obsahu nebo zda indexovat vlastnosti včetně obsahu souborů.

1.3.2 Total Commander

Total Commander je další ze série správců souborů pro operační systém Windows. Vznikl v roce 1993 pro Windows 3.1x. V té době měl název Windows Commander. V roce 2002 se musel přejmenovat na Total Commander, protože s ochrannou známkou Windows v názvu by program mohl vyvolávat dojem, že se jedná o produkt Microsoftu. Na konci ledna 2020 vychází verze 9.50. [16]

Hlavní rozdíl mezi Průzkumníkem souborů a Total Commanderem je očividný - stromovou strukturu složek lze procházet ve dvou oknech najednou, což je vidět na obrázku 1.4. To sebou přináší mnoho výhod, jako jednodušší

kopírování a přesouvání souborů. V jednom okně může být otevřen například vzdálený server a v druhém mohou být lokální soubory, které chceme na server zkopírovat. Total Commander podporuje FTP[7] připojení.

Kromě dvou oken pro procházení souborů současně GUI[8] ve spodní části obsahuje sadu nejčastěji používaných funkcí, které jsou namapovány na funkční klávesy (F3 - F8), a Input box[9], přes který lze v dané složce např. spustit aplikaci nebo vyvolat příkazovou řádku příkazem *cmd*.

Horní část obrazovky zobrazuje všechny podporované funkce. Podobně jako v Průzkumníku souborů lze nastavit různé zobrazení obsahu složek včetně vyvolání rozcestníku na levé straně obrazovky. Dále kromě očekávaných funkcí pro filtrování apod. rozhraní nabízí komprimaci souborů do archivu, extrahování archivu, připojení k FTP serveru, vyhledávání, hromadné přejmenování souborů, synchronizaci obsahu složek, zakódování/dekódování souborů, rozdělení souborů na více částí či jejich sloučení, porovnání obsahu souborů, ověření validity dat pomocí kontrolních součtů atd. Program obsahuje několik verzí vzhledu včetně několika typů vzhledu ikonek a rozložení panelů. Přednastavená rozložení lze pak dále upravovat podle svých preferencí. Program je responzivní, ale při zúžení si stále zachovává rozložení s dvěma okny namísto jednoho, což je dle mě chyba, protože obsah složek nelze přečíst.

Vzhledem k povaze navrhované aplikace je dobré projít si podrobněji možnosti vyhledávání v Total Commanderu. Kliknutím na ikonu dalekohledu (klávesová zkratka Alt+F7) se zobrazí okno Hledání souborů, viz 1.5. Hledat soubory lze pomocí názvu či části názvu souboru i pomocí regulárních výrazů. Lze vyhledávat v obsahu souborů včetně archivů. Nastavit lze umístění, která se mají prohledat, a další filtry jako stáří, velikost či atributy souborů. V záložce Doplňky lze nastavit více podmínek pro vyhledávané soubory. Například, aby soubor končil příponou „.txt“ a nebyl starší než dva měsíce. Lze zakliknout, aby nalezený soubor vyhovoval všem těmto podmínkám, či aby splňoval alespoň jednu. Stejně jako v Průzkumníku souborů lze provedená vyhledávání uložit jako šablony a použít je později znovu. Vyhledávání nepoužívá zabudované indexování systému Windows, takže prohledávání není moc rychlé. Nicméně lze v záložce *Obecné* zaškrtnout možnost *Everything*, která k vyhledávání použije stejnojmenný program specializovaný na vyhledávání souborů a jejich indexaci. Tím se vyhledávání zrychlí. Pro tuto funkci musí být program Everything samozřejmě nainstalován.

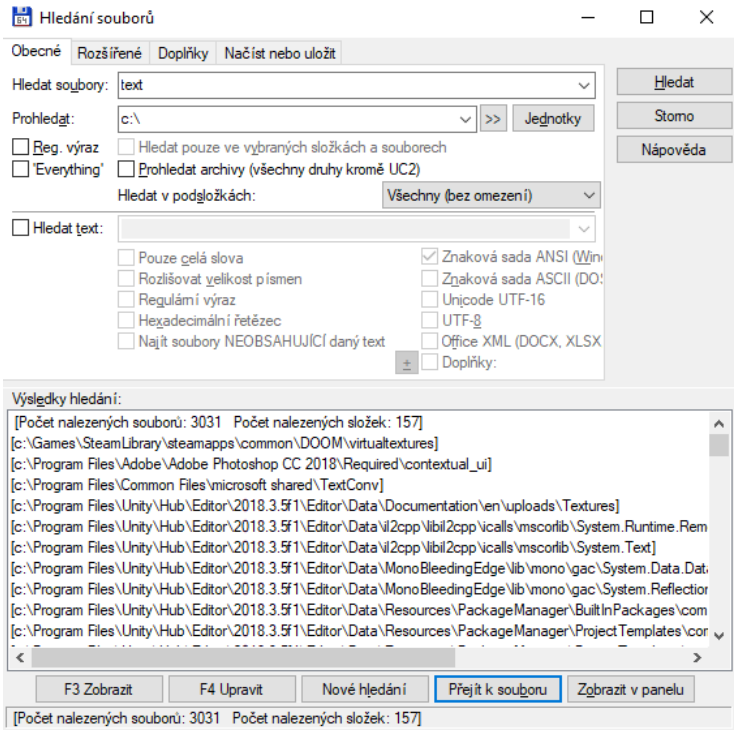
1.3.3 Everything

Everything je program pro okamžité vyhledání souborů a složek podle jména v systému Windows. Při spuštění si program vytvoří index a zobrazí všechny soubory a složky v počítači (proto se nazývá Everything - vše). Program je

[7] File Transfer Protocol - protokol pro přesun dat mezi dvěma počítači [17]

[8] Graphical User Interface - grafické uživatelské rozhraní

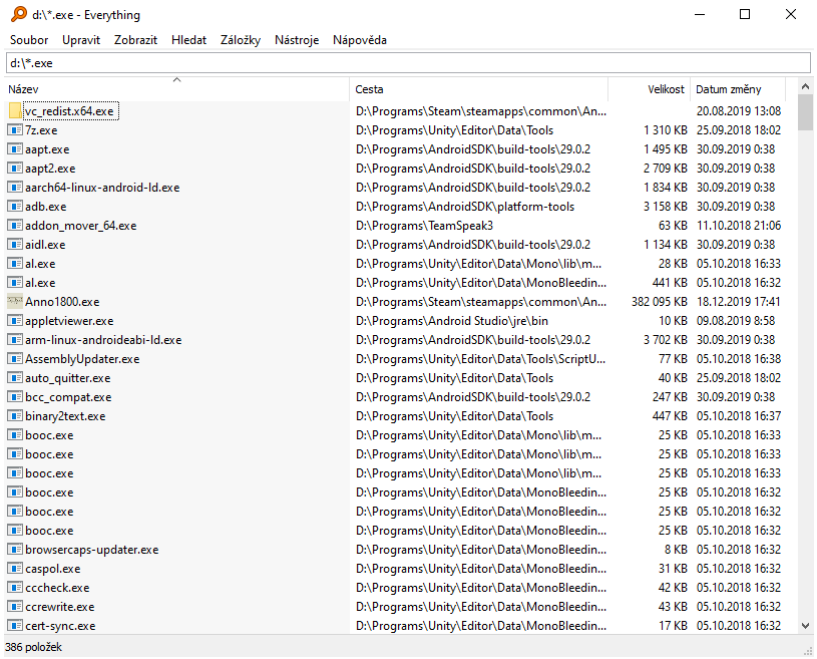
[9] Input box - Vstupní dialogový box



Obrázek 1.5: Vyhledávací okno programu Total Commander, verze 9.22

šířený pod freeware licencí. Nevýhodou je, že podporuje pouze indexování souborového systému NTFS. Podporuje i vyhledávání podle jiných kritérií než je jméno souboru (například podle obsahu souboru), ale takové vyhledávání je pomalé, protože nic jiného než jména souborů a složek nejsou indexována. Výhodou je malé využití paměti (cca 75 MB na milion indexovaných souborů) a rychlé vytvoření indexu při prvním spuštění (minuta pro milion souborů, sekunda pro čistou instalaci Windows). To platilo do verze *1.3*. Od verze *1.4* program ve výchozím nastavení indexuje i datum změny a velikost souborů, čímž se velikost indexu zvýšila, ale je nyní možné instantně vyhledávat i podle těchto metadat. Pokud je program zapnutý, tak si průběžně aktualizuje svůj index, případně si ho zaktualizuje ihned po opětovném spuštění. [18]

Rozhraní programu je velice jednoduché, viz 1.6. Horní lištu obrazovky zaplňuje menu s obvyklými možnostmi pro zobrazení dat, filtrování a vyhledávání. V záložce *Nástroje* lze otevřít nové okno *Možnosti*, kde lze mimo jiné nastavit, jaké disky a jaké atributy souborů (mimo obsah) indexovat. V záložce *Nápověda* lze nalézt syntaxi pro vyhledávání. Everything podporuje regulární výrazy a logické operátory. Pod menu se táhne dialogový box pro napsání hledaného výrazu. Většinu obrazovky zaplňuje výpis všech souborů v počítači — za předpokladu, že je vyhledávací box prázdný — případně výpis aktuálně hledaných souborů. Výpis souborů je standardně rozdělen do několika sloupců



Obrázek 1.6: Výsledek vyhledávání v programu Everything

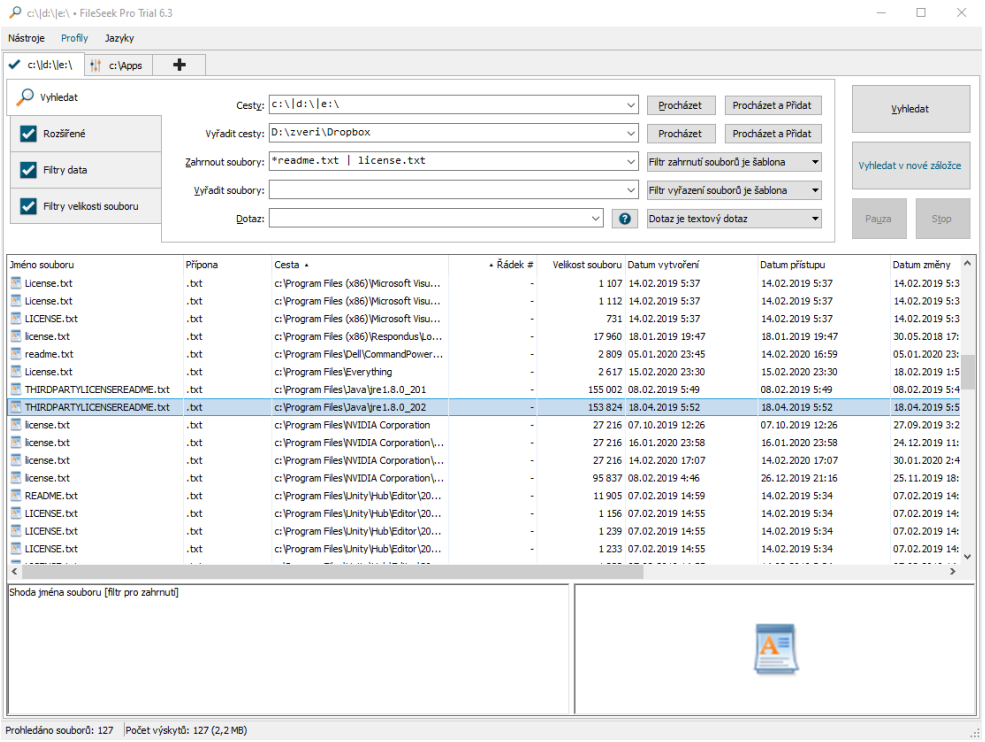
s názvem souboru, cestou k souboru, velikostí a datem změny. Podoba výpisu se dá změnit v záložce *Zobrazit* či v nastavení programu. V tomto ohledu program nenabízí nic výjimečného či neobvyklého.

1.3.4 FileSeek

FileSeek je vyhledávací program šířený pod freeware licencí, s možností zakoupit si pokročilou verzi, ke které máte přístup prvních 30 dnů používání. Jeho devizou je, že používá vícevláknové vyhledávání bez použití indexování, takže nezatěžuje systém spuštěným indexováním na pozadí. Přesto je často rychlejší než výchozí vyhledávání ve Windows, a to i ve chvíli, kdy je zapnuto indexování. K vyhledávání lze použít jak regulární výrazy, tak dotazovací jazyk tohoto programu [19]. Placená pokročilá verze poskytuje funkce vyhledávání v několika kartách najednou, ukládání dotazů do profilů pro budoucí použití, synchronizaci těchto profilů mezi několika počítači a export výsledků do HTML a CSV formátů [20].

Rozhraní programu lze vidět na obrázku 1.7. V horní části obrazovky je velice stručná nabídka. V záložce Nástroje jsou na výběr možnosti pro nastavení aplikace a různé informace o aplikaci. V záložce Profily lze spravovat profily (uložení, nahrání, synchronizace s dalšími počítači). V záložce Jazyky lze vybrat lokalizaci aplikace včetně češtiny.

Zajímavější jsou karty pro vyhledávání. Každá vyhledávací karta obsahuje

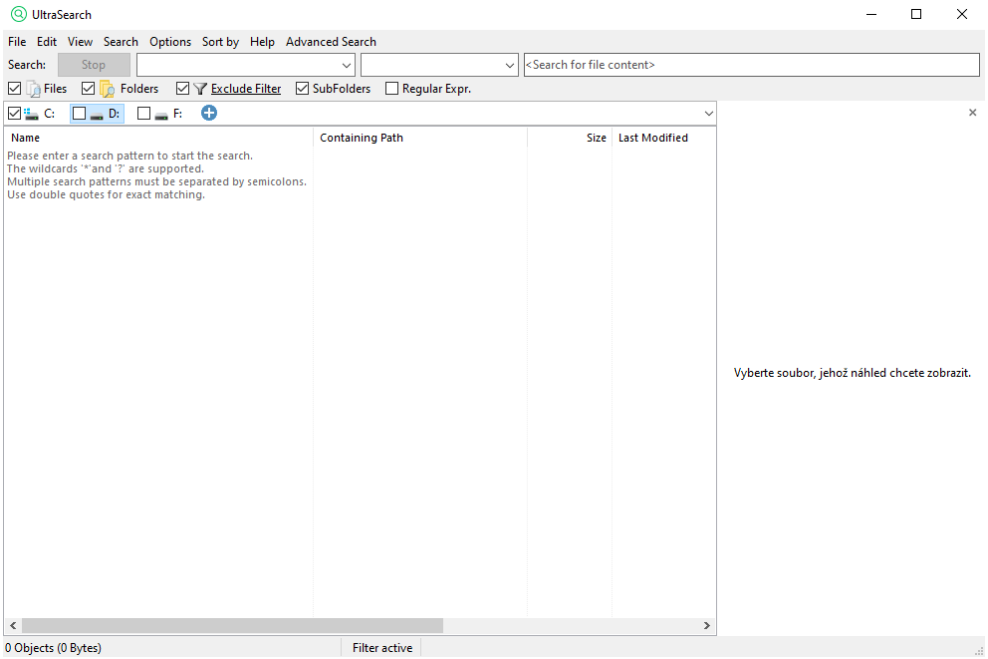


Obrázek 1.7: Výsledek vyhledávání v programu FileSeek

menu s několika obrazovkami, ve kterých lze dotaz intuitivně upřesnit bez znalosti regulárních výrazů. Možnosti jsou široké. Dotazovat se lze na metadata i obsah souborů. Z vyhledávání lze vyloučit určité složky či soubory. Při vyhledávání v obsahu souborů lze zobrazit buď jen název souboru, který hledaný výraz obsahuje, nebo číslo řádku, na kterém se hledaný výraz nachází. Hlavní část obrazovky zabírá výpis výsledků. Výpis je standardní, po kliknutí pravým tlačítkem myši do výsledků lze zobrazit kontextovou nabídku, přes kterou lze například exportovat či archivovat všechny výsledky. Pod výpisem výsledků jsou dvě okna sloužící k zobrazení náhledu souborů, pokud se vyhledávalo v obsahu souborů.

1.3.5 UltraSearch

UltraSearch je další freeware program pro vyhledávání souborů v operačních systémech Windows. Jeho hlavní devizou je, že k vyhledávání nepoužívá index, ale *Master File Table* v NTFS souborových systémech, čímž je schopen poskytnout výsledky během několika jednotek sekund. Je to zároveň i jeho nevýhodou, protože nepodporuje jiné souborové systémy. Vyhledávat lze jak v metadatech souborů, tak v jejich obsahu. Vyhledávat lze i pomocí regulárních výrazů. Výsledky lze exportovat do několika formátů (Excel, text,



Obrázek 1.8: UltraSeek okno v pokročilém (*advanced*) módu

RTF, HTML a CSV), předchozí vyhledávání se mohou uložit pro další použití. Kliknutím pravým tlačítkem myši na výsledek vyhledávání se otevře kontextová nabídka Windows. UltraSearch je distribuovaný i jako přenosná verze bez nutnosti instalace [21]. Zamrzí absence češtiny.

Rozhraní aplikace je velice podobné programu Everything, s tím rozdílem, že vpravo je okno pro zobrazení obsahu souboru. Jeho zobrazení se dá zrušit. Po spuštění se aplikace zeptá, zda se má spustit ve standardním či pokročilém módu, který do rozhraní přidává několik funkcí navíc: filtrování nechtěných souborů či složek, prohledávání podložek, povolení regulárních výrazů a dva vstupní dialogové boxy navíc pro zvolení typu vyhledávaných souborů a pro prohledávání obsahu souborů. Všechny tyto možnosti standardní mód skrývá. Líbí se mi výpis základních pokynů před započetím vyhledávání, aby měl uživatel představu, jak zadat dotaz. V případě potřeby je v horním menu dostupná nápověda. Nicméně vzezření aplikace naznačuje, že je zaměřená na techničtěji zkušenější uživatele.

1.4 Požadavky na program

V této části jsou popsány všechny funkční a nefunkční požadavky na program. Všechny požadavky vycházejí z průzkumu u potenciálních uživatelů a z rešerše existujících programů.

1.4.1 Funkční požadavky

F1 - Indexování souborů
Každé paměťové médium aktuálně připojené k počítači bude možno indexovat. Zapnutí indexování se provede ručně stisknutím tlačítka „Vytvořit index“. Po stisknutí tlačítka se zobrazí detailnější okno s možnostmi indexace - které složky/typy souborů se mají/nemají indexovat a která metadata (velikost, datum vytvoření, datum změny, datum posledního přístupu) indexovat. Po vyplnění možností indexace se spustí indexování paměťového zařízení. V indexu bude možno vyhledávat i pokud je zařízení odpojeno. Po opětovném připojení bude možno manuálně spustit aktualizaci indexu pomocí tlačítka „Aktualizovat index“.

F2 - Vyhledání souborů či složek
Uživatel bude moci vyhledat soubory či složky v indexovaných paměťových zařízeních. Vyhledávání bude možno omezit filtry, např. na určitá paměťová zařízení či složky. Na jména souborů se bude možno dotazovat příponou, částí, či celým jménem souboru. K vyhledávání bude možno použít i další metadata souborů jako datum vytvoření, datum změny, datum posledního přístupu a velikost. Po nastavení všech filtrů uživatel klikne na tlačítko „Hledat“ a systém podle požadavků vyhledá množinu odpovídajících souborů a složek, které zobrazí na výstup.

F3 - Seřazení výpisu výsledků vyhledávání
Systém po vyhledání souborů uživatelem zobrazí výpis nalezených odpovídajících souborů a složek. Tento výpis bude moc uživatel seřadit podle veškerých indexovaných metadat, vzestupně či sestupně.

F4 - Pojmenování paměťových médií
Uživatel bude moci každé paměťové médium libovolně pojmenovat pro jeho rozeznání v odpojeném stavu.

1.4.2 Nefunkční požadavky

N1 - Multiplatformní program
Ze zadání plyne, že systém musí být multiplatformní. Program musí být funkční na operačních systémech Windows, macOS a linuxových distribucích.

	Případy užití							
Požadavky	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
F1	+	+	+	+				
F2					+			+
F3						+		+
F4	+			+				
F5							+	+

Tabulka 1.1: Kontrola pokrytí funkčních požadavků pomocí případů užití

N2 - Použití programovacího jazyka Java
Ze zadání plyne, že systém musí být implementován v jazyce Java.

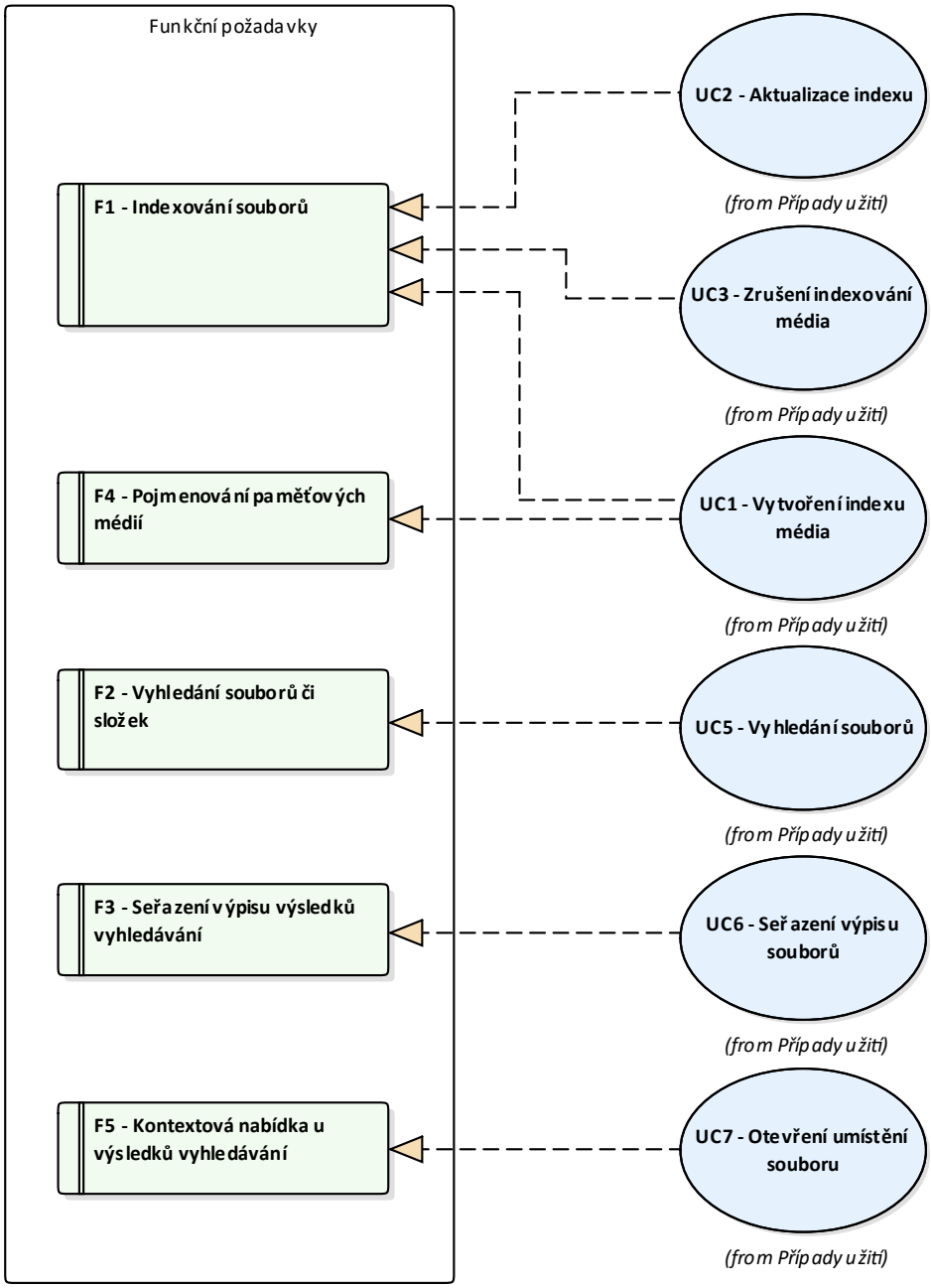
N3 - Grafické uživatelské rozhraní
Systém bude využívat grafické uživatelské rozhraní pro vyšší využitelnost mezi běžnými uživateli.

N4 - Multijazyčnost
Systém bude poskytovat anglickou a českou lokalizaci. Systém bude poskytovat možnost přidat i další jazyky pomocí slovníku termínů.

1.4.3 Případy užití

Podle specifikace funkčních požadavků jsem sestavil několik případů užití upřesňujících jednotlivé funkční požadavky. Případy užití pokrývají všechny funkční požadavky a typicky je rozpadají na několik případů. K jednotlivým případům užití jsem následně sepsal scénáře. Ty slouží jako zadání pro programátory, k přesnějšímu odhadu pracnosti a také k tvorbě akceptačních testů. Jednotlivé případy užití lze vidět v diagramu 1.9 a jejich navázání na funkční požadavky je zobrazeno v diagramu 1.10. Jediným aktérem v případech užití je uživatel, který bude provádět veškerou interakci s programem. Kontrola pokrytí všech funkčních požadavků pomocí případů užití je zobrazena v tabulce 1.1.

Obrázek 1.9: Diagram případů užití



Obrázek 1.10: Mapování případů užití na funkční požadavky

- UC1 - **Vytvoření indexu média**
  - Hlavní scénář: **Korektní vytvoření indexu paměťového média**
  - Předpoklad: **Paměťové médium musí být připojeno k počítači**

1. Případ užití začíná ve chvíli, kdy se uživatel rozhodne indexovat vybrané médium.
2. *Include (Výběr paměťového média)*
3. Uživatel klikne na tlačítko „Vytvořit index“.
4. Systém zobrazí nové okno, které po uživateli požaduje vyplnit název média (povinné), vybrat ze seznamu metadat ta, která chce indexovat, a zvolit složky, které indexovat nechce.
5. Uživatel vyplní název média, vybere metadata k indexaci a případně vybere složky, které na médiu indexovat nechce. Dále klikne na tlačítko „Vytvořit index“.
6. Systém zkontroluje zadané hodnoty a pokud je vše v pořádku, spustí vytváření indexu média. Průběh indexování procentuálně zobrazí v novém okně s plnící se lištou. Po dokončení toto okno samo zmizí, včetně okna s nastavením indexace. V hlavním okně aplikace zobrazí „datum poslední aktualizace indexu“ vedle názvu média. Tlačítko „Vytvořit index“ se změní na „Aktualizovat index“. Dále se zobrazí tlačítko „Možnosti“ pro změnu nastavení indexování daného média a „Zrušit indexování“ pro zrušení indexování média.

- UC1 - **Vytvoření indexu média**
  - Alternativní scénář: **Nevyplněný či špatně vyplněný název média**
  - Předpoklad: **Paměťové médium musí být připojeno k počítači**

1. Případ užití začíná v kroku 6 hlavního scénáře, jestliže systém detekuje nevyplněné pole „Název média“ či pokud toto pole obsahuje nepovolené znaky.
2. Systém zobrazí varovnou hlášku upozorňující na prázdné či špatně vyplněné pole „Název média“.
3. Uživatel vyplní či upraví název média a případně upraví výběr metadat k indexaci a složek, které na médiu indexovat nechce. Dále klikne na tlačítko „Vytvořit index“.

4. Systém zkontroluje zadané hodnoty. Pokud je „Název média“ prázdný či špatně vyplněný, pokračuje scénář krokem číslo 2. Pokud je vše v pořádku, tak scénář pokračuje krokem číslo 6 hlavního scénáře.

- UC2 - **Aktualizace indexu**
  - Hlavní scénář: **Aktualizace indexu**
  - Předpoklad: Vybrané médium má vytvořený index

  1. Případ užití začíná ve chvíli, kdy se uživatel rozhodne aktualizovat již indexované paměťové médium.
  2. *Include (Výběr paměťového média)*
  3. Uživatel klikne na tlačítko „Aktualizovat index“.
  4. Systém provede aktualizaci indexu. Při aktualizaci indexu zobrazí nové okno, které bude procentuálně zobrazovat průběh aktualizace. Po skončení aktualizace okno zmizí a v hlavním okně se upraví „datum poslední aktualizace indexu“.

- UC3 - **Zrušení indexování média**
  - Hlavní scénář: **Zrušení indexování média**
  - Předpoklad: **Vybrané médium má vytvořený index**

  1. Případ užití začíná ve chvíli, kdy se uživatel rozhodne zrušit indexovaní paměťového média a kdy uživatel požaduje vymazání veškerých informací o obsahu média z databáze.
  2. *Include (Výběr paměťového média)*
  3. Uživatel klikne na tlačítko „Zrušit indexování“.
  4. Systém zobrazí nové okno s výzvou, zda si je uživatel opravdu jistý, že chce zrušit indexování daného média.
  5. Uživatel potvrdí svou volbu kliknutím na tlačítko „Ano“.
  6. Systém smaže veškerá data o paměťovém médiu z databáze. Po dokončení operace zobrazí hlášku o úspěšném zrušení indexování.

- UC4 - **Výběr paměťového média**
  - Hlavní scénář: **Výběr paměťového média**
  - Předpoklad: **Médium musí být připojeno k počítači nebo musí mít** vytvořený index

  1. Případ užití začíná ve chvíli, kdy se uživatel rozhodne vybrat jedno z připojených paměťových médií či médií s vytvořeným indexem.

2. Uživatel klikne na kartu „Správa indexování“.
3. Systém v hlavním okně zobrazí všechna připojená paměťová média a další nepřipojená média, která mají vytvořený index.
4. Uživatel vybere jedno ze zobrazených paměťových médií.
5. Systém zobrazí nabídku pro vybrané paměťové médium.

- UC5 - **Vyhledání souborů**
  - Hlavní scénář: **Vyhledání množiny souborů podle filtrů**
  - Předpoklad: **Paměťové médium musí mít vytvořený index**

1. Případ užití začíná ve chvíli, kdy se uživatel rozhodne vyhledat soubory na indexovaných paměťových médiích.
2. Uživatel klikne na kartu „Vyhledávání souborů“.
3. Systém v hlavním okně zobrazí vyhledávací obrazovku s menu obsahujícím filtry pro soubory (paměťová média, názvy , data, velikosti). Ve výchozím nastavení zobrazí výběr z paměťových médií, ve kterých se bude vyhledávat.
4. Uživatel vybere paměťová média, ve kterých chce vyhledávat, nebo zvolí možnost „Všechna média“.
5. Systém upraví vyhledávací dotaz.
6. Uživatel v menu filtrů klikne na položku „Název souboru“.
7. Systém zobrazí obrazovku pro filtrování podle názvů souborů.
8. Uživatel si vybere, zda chce vyhledávat pomocí filtrů typu obsahuje, neobsahuje, začíná na, končí na.
9. Systém zkontroluje, zda uživatelský vstup neobsahuje nepovolené znaky. Pokud ne, tak upraví vyhledávací dotaz.
10. Uživatel v menu filtrů klikne na položku „Datum“.
11. Systém zobrazí obrazovku pro filtrování vyhledávání podle data vytvoření, změny a posledního spuštění.
12. Uživatel zaškrtne, podle kterých dat chce vyhledávat a vyplní potřebná data. U každého data si může vybrat, zda se bude vyhledávat podle filtru „od“, „do“ či „mezi“.
13. Systém zkontroluje validitu dat a upraví vyhledávací dotaz.
14. Uživatel v menu filtrů klikne na položku „Velikosti“.
15. Systém zobrazí obrazovku pro filtrování pomocí velikosti souborů.
16. Uživatel vybere, zda chce vyhledávat pomocí podmínky „menší než“, „menší rovno než“, „větší než“, „větší rovno než“ nebo „mezi“. Po vybrání podmínky vyplní číselnou hodnotu.

17. Systém zkontroluje zadanou hodnotu. Pokud je v pořádku, tak upraví vyhledávací dotaz.
18. *Include (Zobrazení výsledků hledání)*

- UC6 - **Seřazení výpisu souborů**
  - Hlavní scénář: **Seřazení výpisu souborů**
  - Předpoklad: **Výsledky vyhledávání nejsou prázdné**

1. Případ užití začíná tím, kdy systém zobrazil neprázdnou množinu výsledků vyhledávání.
2. *Include (Zobrazení výsledků hledání)*
3. Uživatel klikne na název jednoho ze sloupců „Název“, „Cesta“, „Velikost“, „Datum vytvoření“, „Datum změny“, „Datum posledního přístupu“ ve výsledcích vyhledávání.
4. Systém vyhodnotí, na jaký název sloupce uživatel klikl a vzestupně/sestupně seřadí výstup podle daného sloupce. Seřazený výsledek zobrazí.

- UC7 - **Otevření umístění souboru**
  - Hlavní scénář: **Otevření umístění souboru**
  - Předpoklad: **Výsledky vyhledávání nejsou prázdné a paměťové médium obsahující daný soubor je připojeno k počítači**

1. Případ užití začíná tím, kdy systém zobrazil neprázdnou množinu výsledků vyhledávání.
2. *Include (Zobrazení výsledků hledání)*
3. Uživatel klikne pravým tlačítkem myši na soubor ve výsledcích vyhledávání.
4. Systém otevře kontextovou nabídku obsahující možnost „Otevřít umístění souboru“.
5. Uživatel klikne na možnost „Otevřít umístění souboru“.
6. Systém vyvolá výchozího správce souborů daného operačního systému s umístěním daného souboru.

- UC7 - **Otevření umístění souboru**
  - Alternativní scénář: **Paměťové médium obsahující daný soubor není připojeno k počítači**
  - Předpoklad: **Výsledky vyhledávání nejsou prázdné**

1. Případ užití začíná tím, kdy systém zobrazil neprázdnou množinu výsledků vyhledávání.
2. *Include (Zobrazení výsledků hledání)*
3. Uživatel klikne pravým tlačítkem myši na soubor ve výsledcích vyhledávání.
4. Systém zobrazí nové okno s informační hláškou „Paměťové médium obsahující tento soubor není připojeno k počítači“.
5. Uživatel klikne na „Ok“.
6. Systém schová okno s informační hláškou a nechá uživatele pokračovat v hlavním okně.

- UC8 - **Zobrazení výsledků hledání**
  - Hlavní scénář: **Zobrazení výsledků hledání**

1. Případ užití začíná tím, kdy uživatel nastavil určité vyhledávací filtry.
2. Uživatel klikne na tlačítko „Vyhledat“.
3. Systém na základě sestaveného vyhledávacího dotazu získá výsledky a zobrazí je uživateli na výstup. V případě, kdy nic nenašel, zobrazí hlášku „Nenalezeny žádné výsledky“.

1.5 Doménový model

Doménový model znázorňuje entity v programu a vztahy mezi nimi. Návrh doménového modelu se vytváří spolu s případy užití, aby nedošlo k nesouladu mezi běžnými úkony uživatele a návrhem entit, které tyto úkony podporují. V doménovém modelu se uvádějí jen názvy entit, volitelně jejich atributy (bez datového typu) a vztahy. Případné metody jednotlivých entit nejsou uvedeny. Díky tomu je tento model nezávislý na platformě. [22]

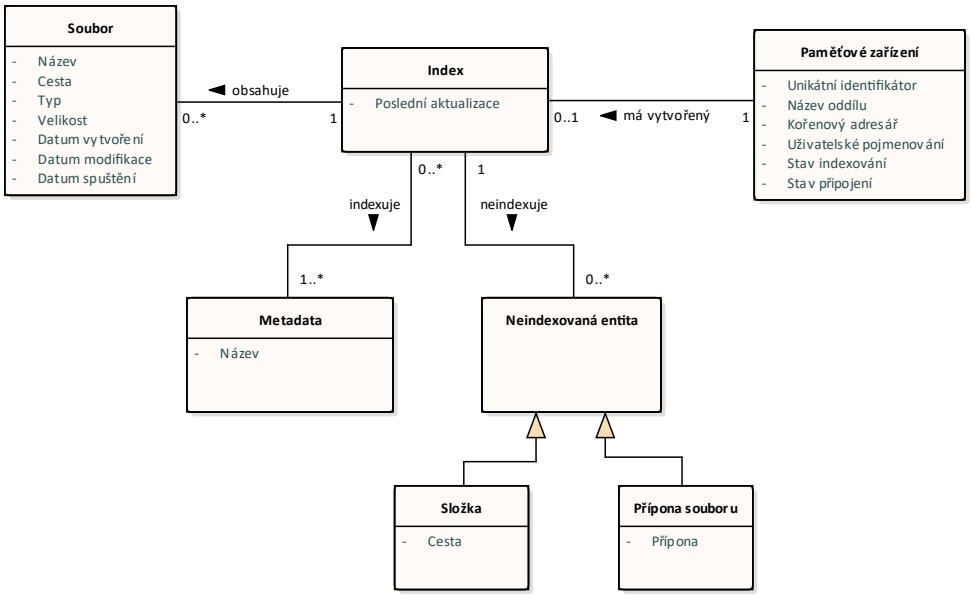
Základní entitou je *paměťové zařízení*. To v sobě jako atribut ukládá „stav indexování“ určující, zda pro toto zařízení existuje index, či nikoliv. Dalším atributem „stav připojení“ je signalizováno, zda je toto médium připojeno k počítači. Dále obsahuje unikátní identifikátor, díky kterému program zaručeně pozná, že se jedná o dané zařízení. Nemůže to být název média, protože uživatel může mít více médií se stejným názvem, například dvě média s názvem „flashka“. K tomuto účelu bude nejspíš použit unikátní identifikátor, který operační systém přiřazuje každému oddílu. Atribut „uživatelské pojmenování“ vytvoří uživatel ve chvíli, kdy bude chtít médium indexovat. Nemusí být unikátní a slouží k snazší identifikaci odpojeného zařízení. Štítek se stejným názvem může uživatel nalepit například na flash disk, aby ho poznal.

Atributy „název oddílu“ a „kořenový adresář“ vyplní program z dat operačního systému.

Ve chvíli, kdy uživatel spustí indexování pro paměťové médium, vytvoří se entita *index*. Každý vytvořený index je přiřazen právě jednomu paměťovému zařízení, paměťové zařízení může existovat bez indexu. Index v sobě uchovává pouze jediný atribut, a to čas poslední aktualizace. Tento čas bude sloužit k určení, zda se jednotlivé složky a soubory změnily a zda se mají při aktualizaci obnovit jejich metadata.

Na index je dále navázána entita *metadata* určující, která metadata se u paměťového zařízení indexují. Index může evidovat více metadat a stejná metadata mohou být indexována ve více indexech. Pokud uživatel nechce indexovat určitou složku nebo typ souborů, tak může k indexu navázat několik entit typu *neindexovaná entita*. Touto entitou může být buď *složka* nebo *přípona souboru*.

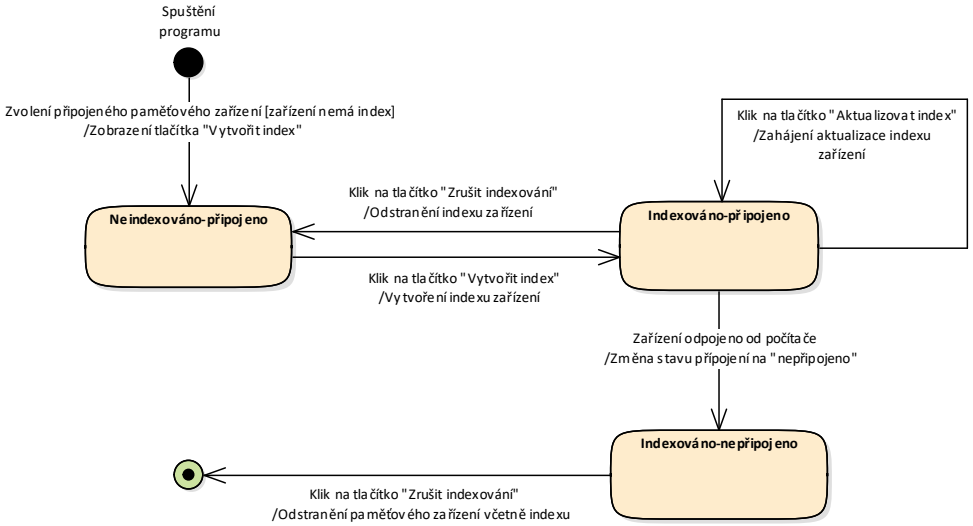
Nejdůležitější entitou navázanou na index je *soubor*. Entita soubor představuje reprezentaci existujícího souboru v paměťovém zařízení. Povinné atributy jsou název, cesta a typ, zatímco nepovinné jsou velikost, datum vytvoření, datum modifikace a datum spuštění. Indexování nepovinných metadat je určeno vztahem mezi entitami index a metadata. Každý soubor je navázán na právě jeden index.



Obrázek 1.11: Analytický doménový model

1.5.1 Stavy paměťového zařízení

Paměťové zařízení může být v jednom ze tří stavů: *neindexováno-připojeno*, *indexováno-připojeno* a *indexováno-nepřipojeno*. Pro každé paměťové médium připojené k počítači se vytvoří entita paměťové zařízení. Pokud toto zařízení nemá vytvořený index, nastaví se mu stav neindexováno-připojeno. U tohoto zařízení má uživatel možnost vytvořit index. Po vytvoření indexu se zařízení dostane do stavu indexováno-připojeno. V takovém případě má uživatel možnost aktualizovat nebo odstranit index. Pokud se indexované zařízení odpojí od počítače, přejde zařízení do stavu indexováno-nepřipojeno. Uživatel má pak možnost jen odstranit index. Pokud index odstraní, zařízení přestává v programu existovat. Proto stav neindexováno-nepřipojeno nemůže existovat. Stavový diagram paměťového média je zobrazen na obrázku 1.12.



Obrázek 1.12: Stavový diagram paměťového zařízení

Kapitola 2

Návrh programu

V této kapitole popisuji veškeré náležitosti ohledně návrhu programu jako architektura programu, drátový model popisující raný návrh vzhledu programu a relační databázový model popisující způsob ukládání dat.

2.1 Model architektury

Pro vyvíjenou aplikaci jsem zvolil třívrstvou architekturu skládající se z prezentační, logické a datové vrstvy.

Prezentační vrstva
: Prezentační vrstva slouží k zobrazení rozhraní a dat uživateli. Způsob implementace prezentační vrstvy bude záležet na výběru grafické knihovny v programovacím jazyku Java. Většinou se tato vrstva skládá ze dvou komponent - *view* a *controller*. View (pohled) slouží k definici vzhledu rozhraní a popisu zobrazení dat uživateli. Controller (ovladač) slouží k reakci na uživatelský vstup (kliknutí na tlačítko, vyplnění textového pole, zaškrtnutí políčka apod.) a jeho následné odeslání na logickou vrstvu pro zpracování. Po zpracování požadavku logickou vrstvou controller pošle výsledek zpět na view a obnoví jej, aby uživateli zobrazovalo aktuální informace nebo aby zobrazilo novou obrazovku.

Logická vrstva
: Logická vrstva se skládá z tříd zvaných *managers* (správců). Tyto třídy se starají o výpočty a zpracování příkazů z controllerů. Pro výpočty používá logická vrstva entity z doménového modelu, které jsou získávány z datové vrstvy. Většinou má každá entita z doménového modelu svůj vlastního správce, ale v mém případě počet správců snížím, protože pro menší entity by bylo zbytečné je vytvářet. Jedná se o entity závislé na indexu: *metadata* a *neindexovaná entita*.

Datová vrstva

Datová vrstva slouží k obsluze databáze. O ukládání, načítání a mazání datových entit se starají *DAO (data access objects)* objekty. Jedná se o tzv. objekty pro přístup k datům, které implementují potřebné CRUD (create, read, update, delete) operace a své rozhraní vystavují pro logickou vrstvu. Zpravidla je každá entita obsluhována vlastním DAO objektem.

Každý správce v logické vrstvě a každý DAO objekt v datové vrstvě bude mít vystaveno své rozhraní, přes které budou objekty mezi sebou komunikovat. Prezentační vrstva může volat rozhraní správců v logické vrstvě, zatímco správcové mohou volat rozhraní DAO objektů z datové vrstvy. Dále mohou správcové volat rozhraní jiných správců, podobně jako DAO objekty mohou volat rozhraní jiných DAO objektů. Díky použití rozhraní lze snadno implementaci jednotlivých DAO objektů/správců nahradit jinou implementací.

2.2 Drátový model - *wireframes*

Na základě případů užití jsem vytvořil drátový model jednotlivých obrazovek aplikace, jinak zvané *wireframes*. Většinou se dělají na začátku projektu pro ujasnění představy o podobě aplikace a pro vysvětlení procesů se zákazníkem. Dají se díky jim objevit chybějící funkce a odhalit nedorozumění, což ve výsledku ušetří jak čas, tak peníze. Slouží jako výchozí bod pro vzhled finální aplikace. Tvorba wireframes je levná a rychlá. [23]

V mém případě jsem drátový model vytvořil, abych zjistil, zda jsem nezapomněl na některou z klíčových funkcí programu a abych si lépe představil vykonání jednotlivých případů užití. Z tohoto návrhu bude navíc vycházet podoba výsledné aplikace.

Obrázek 2.1 ukazuje kartu *Správa indexování*. Na levé straně jsou zobrazeny všechna aktuálně připojená paměťová média k počítači stejně jako ta nepřipojená, která mají vytvořený index. Je vybrán oddíl „C“. Tento oddíl je neindexovaný a detail tohoto paměťového zařízení tak nabízí pouze jedno tlačítko k zahájení tvorby indexu.

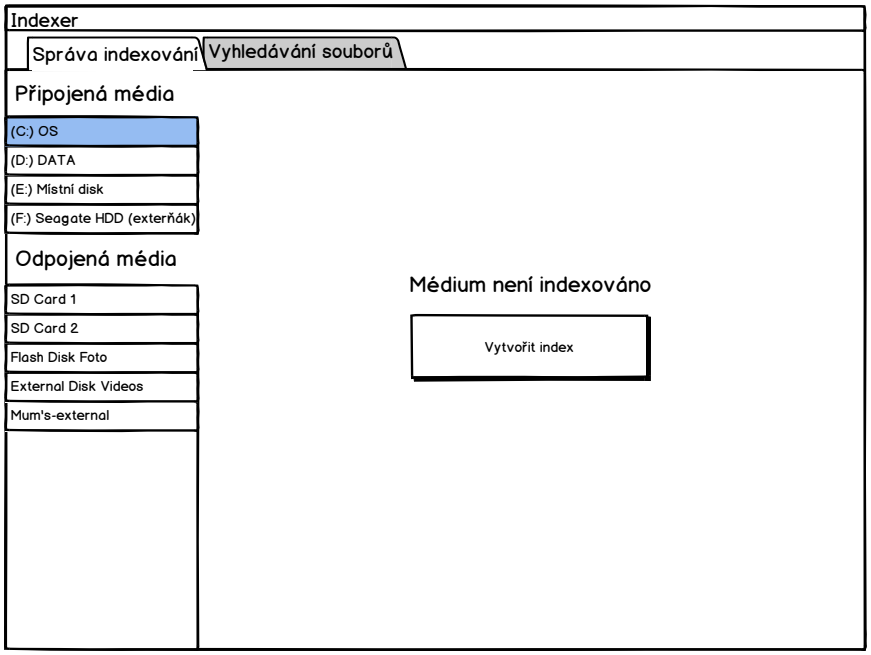
Pokud uživatel klikne na tlačítko *Vytvořit index*, zobrazí se nové okno (obrázek 2.2), ve kterém uživatel zvolí potřebná nastavení pro daný index. Povinné je pole s názvem média, které slouží jako případná náhrada v cestě k souboru. Taková cesta je využita v době, kdy je médium odpojeno, ale soubor z něj je zobrazen ve výsledcích vyhledávání. Dál si uživatel vybere, která nepovinná metadata se mají u souborů indexovat a které adresáře a typy souborů se mají vyloučit z indexování. Po kliknutí na tlačítko *Vytvořit index* se zahájí indexování a zobrazí se nové okno s ukazatelem postupu, které automaticky zmizí po dokončení indexování.

Další obrazovka 2.3 zobrazuje detail paměťového média „C“ s vytvořeným indexem. Detail nyní ukazuje čas poslední aktualizace indexu a nabízí dvě

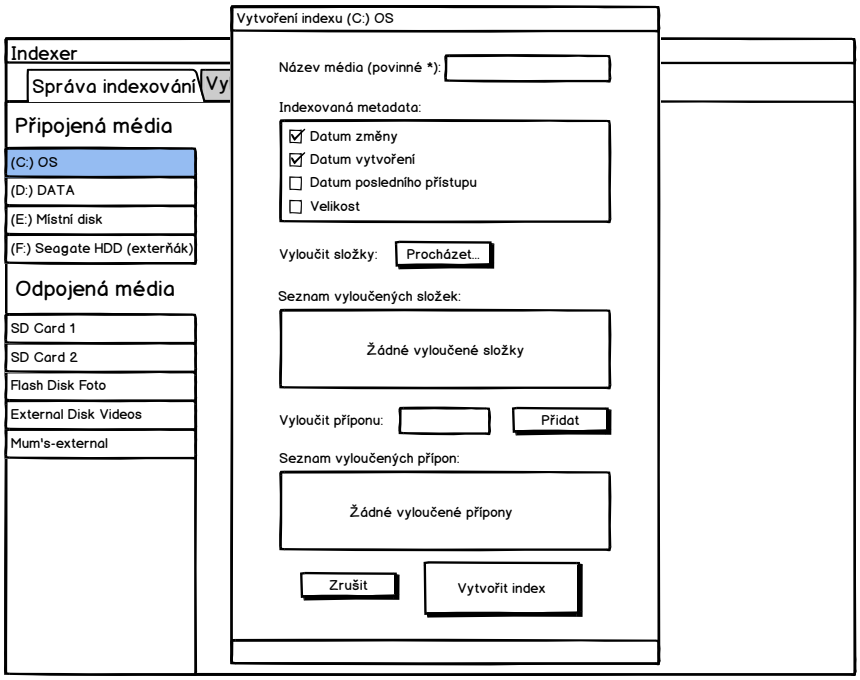
možnosti: aktualizovat index a odstranit index. V seznamu připojených zařízení je nyní vedle názvu oddílu uvedený také uživatelský název, který uživatel zadal při vytváření indexu.

Poslední ukázka z drátového modelu 2.4 ukazuje kartu *Vyhledávání souborů* s vybraným filtrem *Paměťové médium*. V tomto filtru uživatel může nastavit, ve kterých paměťových zařízeních se má vyhledávat. Může vybrat buď všechna indexovaná paměťová zařízení, nebo zaškrtat jejich část.

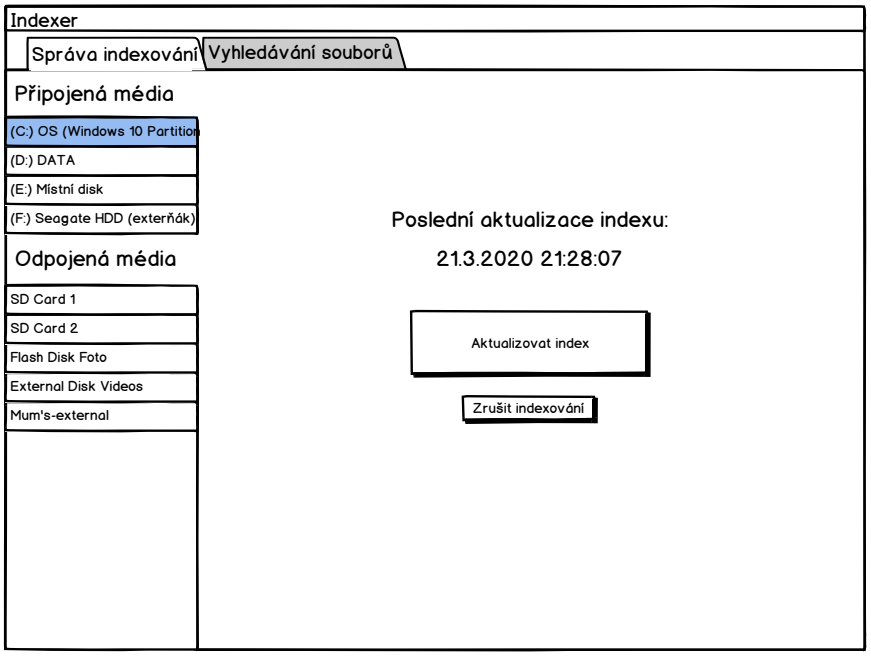
Uvedena je jen část vytvořeného drátového modelu. Kompletní drátový model je uveden na konci práce v příloze A.



Obrázek 2.1: Hlavní okno s neindexovaným médiem



Obrázek 2.2: Okno s nastavením pro vytvářený index



Obrázek 2.3: Hlavní okno s indexovaným médiem

Obrázek 2.4: Vyhledávací okno - filtr paměťových médií

2.3 Způsob uložení dat

Datové modelování je proces vytváření datového modelu pro data uložená v databázích. Takový model je reprezentace datových objektů, vztahů mezi nimi a definicí pravidel. Datové modely se liší svojí podrobností. Konceptuální model zobrazuje pouze názvy objektů, atributy a vztahy mezi objekty, podobně jako doménový model 1.11. Logický datový model přidává typy k atributům, ale zůstává platformně nezávislý. Fyzický datový model je dělaný na míru určité platformě, zobrazuje vztahy včetně primárních a cizích klíčů, využívá platformně závislé datové typy, definuje omezení a další. [22]

Ačkoliv stále nemám vybránu konkrétní implementaci relační databáze, kterou pro svůj projekt použiji, rozhodl jsem se pro databázový model použít fyzický model. Rozhodl jsem se tak kvůli potřebě zobrazit primární a cizí klíče a kvůli určení přesné velikosti určitých řetězců. Datové typy jsem použil z implementace systému řízení bází dat *MySQL*[10]. V konečné implementaci se tak budou datové typy pravděpodobně lišit, jedná se pouze o nástin pro lepší představu.

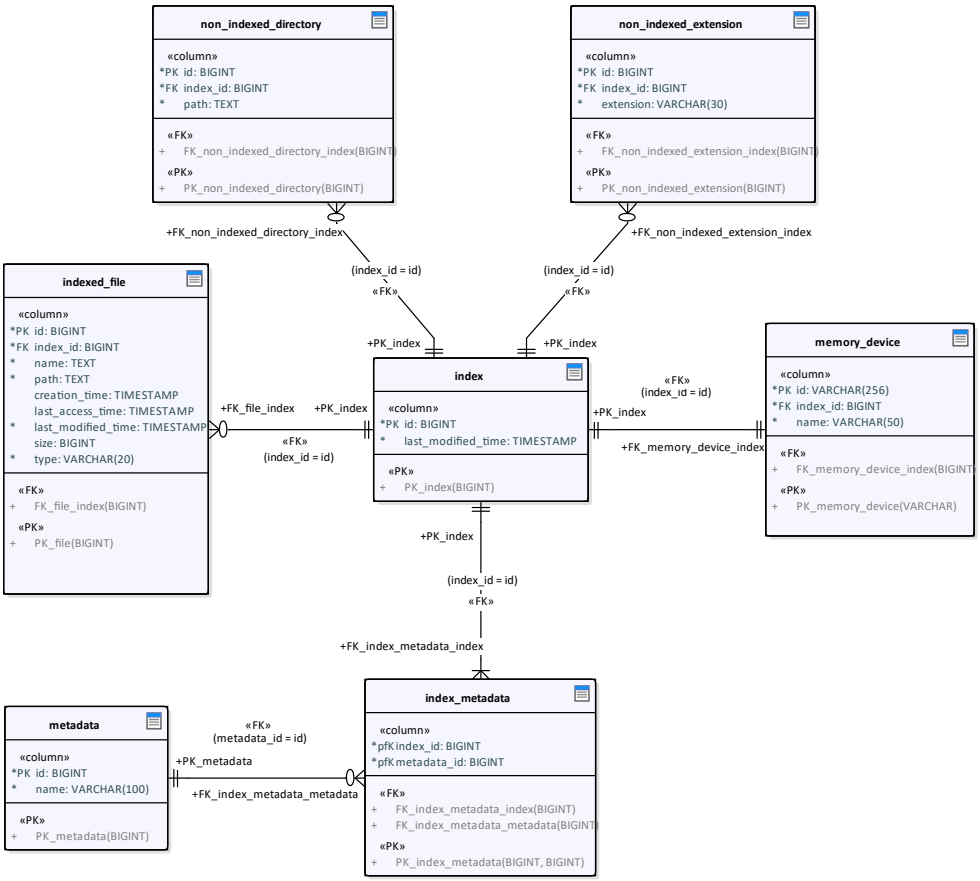
Tabulka *memory_device* pro paměťové zařízení obsahuje pouze atributy „id“ (unikátní identifikátor v operačním systému), „index_id“ (identifikátor spárovaného indexu) a název nastavený uživatelem. Ostatní atributy z doménového modelu není nutno ukládat, název v operačním systému ani kořenový adresář nejsou neměnné atributy a budou získány z operačního systému v době, kdy je paměťové zařízení připojeno k počítači. Paměťové zařízení je s tabulkou *index* ve vztahu 1:1. Každé paměťové zařízení v databázi má vytvořený index a žádný index nemůže existovat bez paměťového zařízení. Pokud paměťové zařízení nemá vytvořený index, tak neexistuje v databázi.

Tabulka *metadata* obsahuje všechny metadata, která mohou být volitelně indexována. S indexem jsou metadata ve vztahu M:N (index může obsahovat více metadat a stejná metadata mohou být indexována více indexy). Je proto nutná dekompozice tohoto vztahu pomocí tabulky *index_metadata*, která páruje index s volitelnými metadaty pomocí identifikátorů.

Vzhledem k tomu, že jsou pouze dva druhy neindexovaných entit (složky a souborové přípony), tak jsem se rozhodl nevytvářet v databázi dědičnost a vytvořil jsem dvě tabulky *non_indexed_directory* (neindexovaná složka) a *non_indexed_extension* (neindexovaná přípona). Obě tabulky jsou s indexem ve vztahu 1:N (index může obsahovat jednu či více neindexovaných přípon/složek, neindexovaná přípona/složka patří právě jednomu indexu).

[10] `https://www.mysql.com/`

Tabulka *indexed_file* pro indexované soubory a *index* jsou ve vztahu 1:N. Každý indexovaný soubor patří právě jednomu indexu, k indexu může patřit jeden či více indexovaných souborů. Tabulku pro soubory jsem přejmenoval z *file* na *indexed_file*, aby nedocházelo k nedorozuměním. Při implementaci budu používat stejné názvy entit a název *File*[11] by se pletl s reprezentací souborů v Javě.



Obrázek 2.5: Datový model programu

[11] `https://docs.oracle.com/javase/7/docs/api/java/io/File.html`

Kapitola 3

Technologie

V této kapitole jsou uvedeny důležité technologie a knihovny, které použiji pro implementaci navrženého programu. Popisuji zde, jaké důvody mám pro použití konkrétních technologií a porovnávám je s jinými možnostmi, které jsem pro implementaci mohl vybrat.

3.1 Java

Nefunkčním požadavkem N2 (více v části 1.4.2) je použití programovacího jazyku Java. Otázka ale zní, jakou verzi bych měl použít? Aktuálně jsou dvě verze s dlouhodobou podporou: Java 8 a o čtyři roky mladší Java 11. Dle dotazníku z ledna 2020 webu JRebel [24] používá pro své programy Javu 8 58 % respondentů, zatímco Javu 11 23 % respondentů. Přesto se více přiklánním k použití novější verze Javy. Z hlediska budoucnosti mi přijde zbytečné omezovat se starší verzí Javy pro vývoj nového programu. Dále se na webu JRebel [24] uvádí, že přechod na novější verzi Javy je u větších projektů občas problémový, což může být také jedním z důvodů, proč je Java 8 stále tolik používaná.

Přinese mi ale použití Javy 11 nějakou výhodu? Dle blogu od Daniela Warrena [25] je jednou z výhod modularita, kterou přinesla Java 9. Nyní lze kód rozdělit do samostatných modulů. Další novinkou jsou *multi-jar releases*. Tato funkcionalita poskytuje možnost používat funkce nabízené Javou 11, zatímco zachová podporu a funkčnost pro klienty využívající Javu 8. Od Javy 9 se navíc bezplatná a open-source implementace Javy *OpenJDK*[12] snaží být co nejbližší placené implementaci *OracleJDK*[13] od firmy Oracle. Díky tomu jsou tyto dvě implementace zaměnitelnější než v případě *OpenJDK 8* a *Oracle JDK 8*. Z těchto důvodů jsem se rozhodl použít Javu 11.

[12] `https://openjdk.java.net/`
[13] `https://www.oracle.com/java/`

3.2 Uživatelské rozhraní

U desktopové aplikace jsou dva typy uživatelského rozhraní, které lze použít: grafické uživatelské rozhraní (GUI - *Graphic User Interface*) a rozhraní příkazového řádku (CLI - *Command Line Interface*). V případě GUI se aplikace ovládá pomocí tlačítek a různých grafických komponent, kdežto aplikace používající CLI je ovládána psaním příkazů do příkazového řádku. Dle nefunkčního požadavku N3 1.4.2 bude navrhovaná aplikace používat GUI, tudíž se možnostmi CLI dále nebudu zabývat. Dalším nefunkčním požadavkem vycházejícím přímo ze zadání je, že má být program implementován v Javě 1.4.2. Musím tedy zvolit jednu z grafických knihoven, kterou nabízí Java. Nejznámější jsou *AWT*, *Swing* a *JavaFX* [26].

AWT

Abstract Window Toolkit je původní knihovna pro vytváření grafického rozhraní v Javě. Jedná se o platformně závislou knihovnu, protože vytváření grafického objektu (například tlačítka) zavolá výchozí metodu operačního systému, která se v každém operačním systému chová jinak. Stejný program bude v operačních systémech Windows vypadat jako Windows aplikace, zatímco v Linuxových distribucích jako Linuxová aplikace. Dnes se nepoužívá kvůli své zastaralosti, platformní závislosti a „těžkému“ propojení s operačním systémem (vytváření grafických objektů je delegováno na operační systém). [27]

Swing

Swing je grafická knihovna postavená na rozhraní AWT. Stejně jako AWT je součástí *Java Foundation Classes*, což je knihovna grafických knihoven zabudovaná v Javě. Na rozdíl od AWT je to platformně nezávislá knihovna a nedeleguje vytváření objektů na operační systém. Kromě základních grafických komponent podporuje i ty pokročilejší jako karty, „scrollovací“ panely, stromy, tabulky a seznamy. Na všech systémech bude mít program jednotný vzhled. [28]

JavaFX

JavaFX je grafická knihovna pro vývoj RIA[14] (*Rich Internet Applications*) aplikací a desktopových aplikací. Aplikace s JavaFX rozhraním jsou funkční jako webové, mobilní i desktopové aplikace. Byla vyvinuta jako náhrada za Swing. Obsahuje více funkcí než Swing. Je také platformně nezávislá a nedeleguje vytváření objektů na operační systém. Podporuje hardwarovou akceleraci. Vzhled oken a jejich komponenty se mohou definovat pomocí FXML dokumentů, které jsou ve formátu XML. Takové dokumenty lze generovat pomocí programu *Scene Builder*. V něm lze

[14] „bohatá webová aplikace“: aplikace běžící ve webovém prohlížeči s vlastnostmi desktopové aplikace

navrhnout vzhled okna přetahováním komponent z menu na plátno. Do Javy ve verzi 8 byla tato knihovna součástí *Java Development Kit* od firmy *Oracle*. Nyní je vyvíjena komunitou pod názvem OpenJFX[15] a musí se do Java projektů přidávat zvlášť. [29]

Pro implementaci jsem zvolil grafickou knihovnu *JavaFX*, protože je nejmodernější a líbí se mi možnost definovat vzhled v programu *Scene Builder*, který mi vygeneruje FXML soubory. Díky tomu bude můj kód čistší, protože nebude obsahovat definice jednotlivých oken a jejich komponent.

3.3 Apache Derby

Pro vyvíjený program použiji tzv. *embedded* databázi, což je systém řízení bázi dat (SŘBD) zabudovaný do programu jako jeho součást [30]. Rozhodl jsem se použít SŘBD *Apache Derby*, které je vyvinuto neziskovou organizací *Apache Software Foundation*.

Rozhodl jsem se pro něj, protože se jedná o *open-source* relační databázi s malými nároky na hardware a prostor (základní funkce a JDBC ovladač zabírají okolo 3.5 MB prostoru). Plně podporuje Java, JDBC a SQL standardy. [31]

3.4 Java Persistence API

Pro práci s databází jsem se rozhodl použít *Java Persistence API* (JPA). Jedná se o standard popisující rozhraní a chování knihoven pro objektově-relační mapování (ORM). ORM tedy ve zkratce znamená, že jakoukoliv třídu (anotovanou jako *@Entity*) v programu lze za splnění určitých podmínek anotovat tak, aby se pro ní v databázi automaticky vytvořila tabulka, včetně vztahů s jinými entitami. Jak jsem již psal, JPA je standard pro rozhraní, kterého se jednotlivé implementace drží a jsou díky tomu zaměnitelné (pokud nejsou v kódu použity rozšířené možnosti jednotlivých implementací). Nejznámější jsou dvě *open-source* implementace: *Hibernate*[16] a *EclipseLink*[17]. [32]

Hibernate

Údajně jedna z nejvyspělejších implementaci JPA, s obrovskou komunitou pracující na projektu. Implementuje veškeré standardní funkce JPA a navíc přidává vlastní funkcionality: *Hibernate tools*[18], validaci podmínek pomocí anotací (délka řetězců, minimální hodnota čísla apod.)

[15] `https://openjfx.io/`
[16] `https://hibernate.org/`
[17] `https://www.eclipse.org/eclipselink/`
[18] `http://hibernate.org/tools/`

a fulltextové vyhledávání. Oproti EclipseLink podporuje Hibernate anotaci *@BatchSize* pro nastavení velikosti dávky získaných entit z tabulky. [32]

EclipseLink
: JPA implementace vytvořena *Eclipse Foundation*. EclipseLink podporuje řadu dalších perzistentních standardů jako *Java Architecture for XML Binding* (JAXB). Jednoduše řečeno se jedná o ukládání objektů do XML reprezentace, namísto řádku v tabulce relační databáze. Na rozdíl od Hibernate podporuje anotaci *@ReadOnly* pro specifikaci entit uložených v módu pouze pro čtení, nebo anotaci *@Struct*, která namapuje třídu do databáze jako typ *struct*. [32]

Protože si myslím, že pro implementaci mého programu budu potřebovat jen základní rozhraní JPA, tak mi nezáleží na výhodách a rozšířených funkcionalitách jednotlivých implementací. Zajímá mě hlavně rychlost. Dle porovnání rychlosti Hibernate s *embedded* databází Apache Derby oproti EclipseLink s Derby na webu *jpab.org* [33] vyplývá, že tato dvě řešení jsou podobně rychlá v případě ukládání dat do databáze, ale EclipseLink je až třikrát rychlejší než Hibernate v případě získávání velkého počtu entit z databáze. Celkově je řešení EclipseLink s Derby databází vyhodnoceno jako až třikrát rychlejší než Hibernate s Derby.

Proto jsem se rozhodl použít JPA implementaci EclipseLink. Až trojnásobná rychlost při získávání dat z databáze může znamenat významný rozdíl při vyhledávání ve statisících souborů.

3.5 Gradle

Gradle je nástroj pro automatizaci, resp. jazyk pro automatizaci. Díky němu mohu automatizovat sestavení projektu, vytvoření spustitelného *JAR* balíčku, vygenerování dokumentace, připravení releasu, provolání testovacích scénářů atd. Gradle je zdarma pod licencí *Apache License, Version 2.0*. [34]

Pomocí jej přidávám do projektu další knihovny, včetně JavaFX a Apache Derby databáze. Dříve jsem pro automatizaci používal nástroj *Maven*[19], ale Gradle má dle mého názoru kratší, srozumitelnější a přehlednější syntaxi nevyužívající XML formát.

3.6 Lombok

Lombok je *open-source* knihovna v Javě, která slouží pro anotaci tříd. Díky této třídě nemusím psát gettery, settery, toString, equals, implicitní konstruktory a další. Stačí pouze anotovat celou třídu anotací „Data“ a celá třída bude

[19] `https://maven.apache.org/`

obsahovat všechny výše zmíněné metody s výchozí implementací. To se hodí hlavně u tříd představujících datové objekty, které nemají žádnou nebo skoro žádnou logiku a slouží jen pro ukládání hodnot. Pokud nepotřebuji pro všechny atributy gettery a settery, tak stačí ke každé proměnné zvlášť připsat anotaci „Getter“ nebo „Setter“ pro jejich vytvoření. Použití anotací výrazně přispívá k čistotě kódu, proto knihovnu Lombok používám. [35]

3.7 Oshi

Oshi je knihovna pro získávání informací o operačním systému a hardwaru počítače pro programovací jazyk Java. Je zdarma a nepotřebuje instalaci dodatečných knihoven. Zaměřuje se na multiplatformní implementaci získávání informací o verzi operačního systému, procesech, paměťové a procesorové vytíženosti, paměťových zařízeních a jejich oddílech a dalších připojených zařízeních. [36]

Tuto knihovnu jsem použil z důvodu multiplatformního získávání informací o systému. Díky tomu nemusím pro každý operační systém implementovat logiku zajišťující získání informací o paměťových zařízeních zvlášť.

3.8 AppDirs

AppDirs je malá Java knihovna poskytující cesty k platformně závislým speciálním složkám/adresářům. Jako příklad lze uvést adresáře pro ukládání aplikačních dat, které programy používají ke své správné funkci. Obvyklá cesta k takovému adresáři se na každém systému liší [37]:

- **Mac OS X**: /Users/<Account>/Library/Application Support/<AppName>
- **Windows 7**: C:\Users\<Account>\AppData\<AppAuthor>\<AppName>
- **Unix/Linux**: /home/<account>/.local/share/<AppName>

Knihovnu využiji pro získání správné cesty do adresáře, kde budu uchovávat data (databázi). Tím elegantně vyřeším problém s multiplatformním řešením.

3.9 Logování

Pro logování důležitých kroků v běhu programu a hledání chyb při debugování jsem použil *Simple Logging Facade for Java*: *SLF4J*. Jedná se o jednoduchý návrhový vzor fasáda, která slouží jako abstrakce pro využití konkrétního logovacího frameworku (například *java.util.logging*, *logback* nebo *log4j*). Použití SLF4J umožňuje jednoduchou záměnu jednoho logovacího frameworku za druhý bez změny v kódu [38].

Z logovacích frameworků jsem zvolil *Log4j 2* od *Apache Software Foundation*, což je vylepšená a rychlejší verze jeho předchůdce. Je také výkonnější jak *Logback*[20]. Veškerá vylepšení oproti předchůdci, nové funkce a výkonnostní testy lze nalézt na oficiálním webu o Log4j 2 [39].

[20] `http://logback.qos.ch/`

Kapitola 4

Implementace

Kapitola *Implementace* rozebírá implementaci nejdůležitějších částí programu *Indexer*. Popsal jsem způsob tvorby grafického uživatelského rozhraní, jednotlivé komponenty prezentační vrstvy, tvorbu indexu pro paměťová zařízení, vyhledávání, lokalizaci aplikace, způsob ukládání dat do databáze. V neposlední řadě také zhodnocuji výsledný program.

4.1 Prezentační vrstva

V kapitole *Technologie* v části *Uživatelské rozhraní* 3.2 jsem vybral pro grafické rozhraní knihovnu *JavaFX*. Návrhový vzor, který prezentační vrstva reprezentuje, je prakticky MVC (*Model-View-Controller*). *View* definuje vzhled a zobrazení dat uživateli. *Controller* aktualizuje view a reaguje na uživatelské interakce. Získává data z modelu či (v mém případě) z logické vrstvy programu. Tu používá jak pro získání dat, tak pro volání operací (vyhledat soubory, vytvořit index atd.).

Pro reprezentaci view používám FXML soubory, které generuji pomocí programu *Scene Builder*. Co znamená FXML soubor a co je to program Scene Builder jsem již zmínil v části 3.2. Pro každou obrazovku a všechny dialogy jsem vytvořil samostatné FXML soubory. Každý FXML soubor má svůj vlastní *controller* s logikou. Některé FXML soubory v sobě zahrnují vložené FXML soubory. Používání FXML souborů má tu výhodu, že nemusím v kódu vytvářet jednotlivé komponenty, ale mohu je popsat pomocí XML syntaxe mimo kód. Tím zachovám čitelnost kódu a můžu se soustředit na definování logiky, a ne na definování vzhledu. Následně mi stačí použít třídu *FXMLLoader* pro načtení struktury obrazovky a všech definovaných komponent. Načtená data předám scéně a tu vložím do aktuální *stage* (jeviště), kterou pak zobrazím uživateli.

Hlavní okno programu popsané v FXML souboru *MainScene.fxml* je vyvoláno při startu programu. Obsahuje panel s kartami a patičku s ikonami

pro změnu jazyka. Ikony vlajek vytvořili *Roundicons*[21] na webu *flaticon.com*. Tento FXML soubor má v sobě vnořené dva další FXML soubory, jeden pro každou kartu: *IndexManagement.fxml* pro správu indexování a *FileSearch.fxml* pro vyhledávání souborů. Vnořený panel pro správu indexování obsahuje vlevo seznam s připojenými/odpojenými zařízeními a pravou část pro detail paměťového zařízení. Ikonu pro obnovení seznamu připojených a odpojených zařízení vytvořili *Vectors Market*[22] na webu *flaticon.com*.

Pravá část je po vybrání paměťového zařízení zaplněna buď panelem pro indexované zařízení, či panelem pro neindexované zařízení. Ty jsou popsány také FXML soubory, ale jsou nastavovány dynamicky v kódu podle atributu „stav indexování“ paměťového zařízení.

Obrazovka pro vyhledávání souborů je tvořena kontejnerem zvaným *BorderPane*. Ten je rozdělen do několika panelů strukturou podobně jako webová stránka: obsahuje hlavičku, levou stranu, střed, pravou stranu a patičku. Rozmístění komponent bylo problematické, kvůli zachování responzivity (aby se obsah okna roztahoval společně s velikostí okna). Vyřešil jsem to tak, že jsem do hlavičky BorderPane přidal další BorderPane. V tomto vnořeném panelu jsem na levou stranu umístil menu s tlačítky pro filtry a pravou stranu jsem vyplnil detailem filtru, který zvolí uživatel. V kořenovém BorderPane je pak střed vyplněn komponentou *TableView* (tabulka) s výsledky vyhledávání.

Používání programu Scene Builder mi velice ulehčilo práci s vytvářením responzivního grafického rozhraní. Nalezl jsem ale jeden problém, pokud není FXML soubor a jeho controller ve stejném adresáři. V takovém případě nelze předem připravit názvy tlačítek a dalších komponent v controlleru a následně je přiřadit pomocí Scene Builderu komponentám v rozhraní. Nezobrazí se totiž jejich seznam definovaný v controlleru. Tento proces se musí otočit a jména komponent se musí definovat v Scene Builderu a následně je přiřadit do controlleru.

4.2 DAO vrstva

Datová vrstva je spravována pomocí JPA implementace EclipseLink. Aby třída mohla být uložena v databázi pomocí JPA (mluvím o entitách z databázového modelu 2.5), musí splňovat několik podmínek [40]:

- musí být anotována pomocí *@Entity*
- musí obsahovat jeden veřejný či chráněný konstruktor bez parametrů
- třída ani její metody nesmí být final
- atributy třídy nejsou veřejné a ke každému z nich existuje setter a getter

[21] `https://www.flaticon.com/authors/roundicons`
[22] `https://www.flaticon.com/authors/vectors-market`

Objekty třídy, která tyto podmínky splňuje, jsou poté ukládány do stejnojmenné tabulky jako její řádky. Pokud třída obsahuje proměnné obsahující objekty jiné třídy, musím definovat vztah s touto třídou. Ty jsou stejné jako relace v relačních databázích: *@OneToOne* (1:1), *@OneToMany* (1:N), *@ManyToOne* (N:1) a *@ManyToMany* (M:N) [40].

O správu entit a jejich životní cyklus (tedy o načítání, ukládání, mazání a obnovování) se stará třída implementující rozhraní *EntityManager*. Toto rozhraní předepisuje následující klíčové metody [40]:

- find - načte entitu se zadaným klíčem z úložiště do kontextu
- refresh - obnoví entitu v kontextu dle úložiště
- persist - přidá entitu do kontextu
- merge - upraví entitu v úložišti dle kontextu
- remove - odebere entitu z úložiště
- detach - odebere entitu z kontextu

EntityManager se většinou získává tzv. *dependency injection* ze souboru *persistence.xml*, kde je popsáno nastavení JPA, logování a připojení k databázi. V mém případě jsem tento postup nemohl použít, protože já při startu programu nevím, na jakém operačním systému program běží a kde mám vytvořit adresář s databází. Umístění adresáře s programovými daty musím zjistit dynamicky za běhu. K tomu používám utilitu *AppDirs* 3.8. Po získání cesty k adresáři musím tuto informaci sdělit třídě *EntityManagerFactory*, která se stará o vytváření instance EntityManager. Při vytváření instance EntityManager použije jak neměnná nastavení ze souboru *persistence.xml*, tak dynamicky zjištěnou cestu k adresáři s programovými daty. Tato logika je umístěná mimo třídní hierarchii programu ve třídě *UtilTools*.

V ukázce kódu 4.1 je zobrazena kompletní třída *Index*, na které ukazuji, jak se anotují třídy pomocí JPA a knihovny Lombok, zatímco v ukázce kódu 4.2 ukazuji pouze kus třídy *IndexedFile* s vytvořením vztahu N:1 se třídou Index pomocí JPA anotace. Ve třídě Index neukládám množinu všech indexovaných souborů kvůli lepšímu výkonu. Bylo by zbytečné načítat všechny indexované soubory při každé inicializaci paměťového zařízení. Pokud indexované soubory potřebuji při vyhledávání nebo při aktualizaci/mazání indexu, tak se vždy doptám datové vrstvy podle zvoleného paměťového zařízení.

Ukázka kódu 4.1: Třída Index anotovaná pomocí JPA a Lombok pluginu

```
@Entity
public class Index {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(nullable = false)
        @Getter @Setter private LocalDateTime lastModifiedTime;

        @OneToOne(mappedBy = "index", fetch = FetchType.EAGER)
        @Getter private MemoryDevice memoryDevice;

        @ManyToMany
        @Getter private Set <Metadata> indexedMetadata = new
            HashSet<>();

        @OneToMany(mappedBy = "index", cascade = CascadeType.ALL,
            orphanRemoval = true)
        @Getter private Set<NonIndexedDirectory>
            nonIndexedDirectories = new HashSet<>();

        @OneToMany(mappedBy = "index", cascade = CascadeType.ALL,
            orphanRemoval = true)
        @Getter private Set<NonIndexedExtension>
            nonIndexedExtensions = new HashSet<>();

        @Override
        public String toString() {
                return "Index{" +
                        "id=" + id +
                        ", lastModifiedTime=" + lastModifiedTime +
                        ", memoryDevice=" + memoryDevice +
                        '}';
        }

        public Index() {}
}
```

Ukázka kódu 4.2: Ukázka z třídy IndexedFile s definicí vztahu s třídou Index

```
@Data @Entity
public class IndexedFile {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @ManyToOne(fetch = FetchType.EAGER)
        private Index index;
}
```

Kvůli použití JPA jsem implementoval pouze tři DAO objekty: pro indexované soubory, paměťové zařízení a metadata. Nepotřebuji DAO objekty pro všechny entity. V případě, kdy ukládám paměťové zařízení do databáze, tak spolu s ním ukládám i entitu index, která je s paměťovým zařízením ve vztahu 1:1. Entita index musí existovat (nesmí být *null*). Stejně tak všechny entity napojené na index (kromě indexovaných souborů) vytvářím spolu s indexem. Metadata v databázi vytvářím zvlášť, protože musí existovat už při spuštění programu (případně se vytvořit při prvním spuštění) a jsou s indexem ve vztahu M:N. Indexované soubory vytvářím v databázi zvlášť při vytváření/aktualizaci indexu. Vztah s indexem se vytvoří díky vztahu N:1.

Každý DAO objekt má své rozhraní a implementaci poskytující základní CRUD (*create*, *read*, *update*, *delete*) operace. *IndexedFileDAO* navíc obsahuje metodu pro vyhledávání souborů, která na základě přijatých parametrů získá z databáze hledané soubory. U metod pro získání objektů z databáze používám tzv. *CriteriaBuilder*. Ten slouží pro složení komplexního vyhledávacího dotazu bez použití SQL. Díky tomu zůstává kód čistý a bude fungovat i při použití jiných databázových systémů. Proto (i když *EclipseLink* takovou možnost nabízí) v žádných databázových dotazech SQL nepoužívám.

4.3 Indexování

U indexování jsem měl dlouho problém s tím vymyslet co nejefektivnější řešení, které by bylo multiplatformní. Pokud bych se zaměřil na rychlost, tak bych musel pro každý operační a souborový systém zvlášť vymyslet optimální řešení. Já chtěl ale zachovat obecnost, čitelnost a jednoduchost kódu. Proto jsem zvolil metodu třídy *Files*[23] knihovny *java.nio.file*: *walkFileTree*. Metoda slouží k procházení stromové struktury všech souborů od zvoleného kořenového adresáře. Má dva parametry: objekt třídy *Path* a implementaci rozhraní *FileVisitor*.

Třída Path v sobě obsahuje zvolený kořenový adresář. V mém případě je to vždy kořenový adresář (*mount*) paměťového zařízení, kterému vytvářím index. Druhým parametrem je třída implementující rozhraní FileVisitor. Implementovat musí čtyři metody. Metodu *preVisitDirectory* pro operace vykonané před navštívením adresáře a *postVisitDirectory* pro akce vykonané po opuštění adresáře. Je použit algoritmus prohledávání do hloubky, takže zavolání metody postVisitDirectory probíhá při vynoření o úroveň výš ve stromové struktuře. Dalšími metodami jsou *visitFile* pro operace se soubory ve složce a *visitFileFailed* pro operace ve chvíli, kdy čtení souboru skončilo chybou (nejčastěji kvůli chybějícím přístupovým právům).

Pro ovládání procházení stromové struktury adresářů se používá výčet hodnot *FileVisitResult*, který je návratovou hodnotou všech čtyř výše zmíněných

[23] `https://docs.oracle.com/javase/7/docs/api/java/nio/file/Files.html`

metod. Eviduje čtyři různé stavy: *continue* pro pokračování v procházení, *terminate* pro okamžité ukončení procházení adresářové struktury, *skip_subtree* pro ignorování dětí daného adresáře a *skip_siblings* pro ignorování všech dalších souborů ve stejném adresáři.

Pro implementaci rozhraní jsem použil implementační třídu *SimpleFileVisitor*[24], u které jsem přepsal metody *visitFile*, *visitFileFailed* a *preVisitDirectory*. Pokud návštěva souboru skončí chybou, tak ignoruji všechny jeho děti. Při úspěšném navštívení souboru (pokud se nejedná o vyloučený typ souborů) vytvořím indexovaný soubor. Během navštívení adresáře při zanořování aktualizuji zobrazený průběh indexování. Před započetím indexování totiž nejprve projdu celou stromovou strukturu pro spočítání celkového počtu adresářů ke zpracování. Následně při indexování u každého navštíveného adresáře zvýším počítadlo a na základě toho se uživateli zobrazuje ukazatel průběhu. Pokud se nejedná o vyloučený adresář, tak adresář také indexuji. Implementace metody walkFileTree je uvedena v ukázce kódu 4.3.

Aby zobrazování průběhu fungovalo a program při indexování nezamrzl, tak musí indexování běžet v nezávislé úloze. Proto je celý tento proces zapouzdřen v třídě *CreateIndexTask*, která dědí od třídy *Task*. Obdobně je vytvořena třída *UpdateIndexTask* pro aktualizaci indexu.

Aktualizace indexu funguje trochu jinak. Používá se v ní zejména metoda *postVisitDirectory*. V té se kontroluje, zda byla daná složka změněna od doby poslední aktualizace (či poprvé vytvoření) indexu. Pokud ne, tak se pokračuje dál. Pokud ano, tak se zavolá metoda *updateDirectory*, která získá všechny indexované soubory v daném adresáři z databáze a všechny soubory, které adresář aktuálně obsahuje. Následně se naleznou shody u existujících a indexovaných souborů (v případě změny se aktualizují uložená metadata), přidají se nové soubory a odstraní se z databáze ty indexované soubory, ke kterým už neexistuje do páru existující reálný soubor.

[24] `https://docs.oracle.com/javase/7/docs/api/java/nio/file/SimpleFileVisitor.html`

Ukázka kódu 4.3: Metoda walkFileTree - vytváření indexu

```
Files.walkFileTree(start, new SimpleFileVisitor<Path>() {
    @Override
    public FileVisitResult visitFile(Path file,
      BasicFileAttributes attrs) throws IOException {
        if (!isFileExtensionIndexed(file, disabledExtensionsSet)) {
            return FileVisitResult.CONTINUE;
        }

        return createIndexedFile(file, attrs, memoryDevice,
            indexedMetadataSet, indexedFileBuffer);
    }

    @Override
    public FileVisitResult visitFileFailed(Path file,
      IOException e) throws IOException {
        return FileVisitResult.SKIP_SUBTREE;
    }

    @Override
    public FileVisitResult preVisitDirectory(Path dir,
      BasicFileAttributes attrs) throws IOException {
        if (isCancelled()) {
            return FileVisitResult.TERMINATE;
        }

        updateProgress(++iterations[0], totalIterations);

        if (!isDirectoryIndexed(dir, memoryDevice,
            disabledDirectoriesSet)) {
            return FileVisitResult.SKIP_SUBTREE;
        }

        if (memoryDevice.getMount().equals(dir.toString())) {
            return FileVisitResult.CONTINUE;
        }

        return createIndexedFile(dir, attrs, memoryDevice,
            indexedMetadataSet, indexedFileBuffer);
    }
});
```

4.4 Vyhledávání

Při implementaci vyhledávání jsem musel vyřešit problém, jak všechny filtry vyplněné uživatelem předat z prezentační vrstvy do logické. Vyřešil jsem to vytvořením několika tříd, které slouží jako kontejnery pro uživatelem zadané hodnoty: *SearchDateValue*, *SearchFileNameValue* a *SearchSizeValue*. Všechny tyto třídy ukládají dvojici: hodnota z *choiceBox* komponenty (výběrového menu) a hodnota zadaná uživatelem. Například u filtru pro jméno souboru uložím do *SeachFileNameValue* z výběrového menu jednu z hodnot „obsahuje“, „neobsahuje“, „začíná“ nebo „končí“ a pak samotný řetězec zadaný uživatelem. U ostatních filtrů je funkce obdobná, s výjimkou filtru pro paměťová zařízení. Do logické vrstvy se posílá *boolean* hodnota, zda chce uživatel prohledávat všechna paměťová média. Také se předává seznam zaškrtnutých zařízení k prohledání. Ten je ale použit jen v případě, kdy je *boolean* hodnota nastavena na nepravdu (*false*).

Musím ošetřovat případy, ve kterých uživatel zadá chybný vstup. Například vybere hodnotu z choiceBoxu, ale nevyplní řetězec. V takovém případě se kontejner SeachDateValue nevytvoří a do logické vrstvy se pošle *null* hodnota. Uživatel se může rozhodnout, zda chce prohledávat všechna indexovaná média, nebo vybraná média. Pokud si chce sám zvolit, zaškrtne možnost „Vybraná paměťová média“. Pokud ale následně nezaškrtne ani jedno z indexovaných zařízení a klikne na tlačítko „Vyhledat“, tak program zahlásí chybu.

FileSearchManager v logické vrstvě v závislosti na vstupu rozhodne, která paměťová zařízení se mají prohledávat. Vytvoří jejich seznam a spolu s výše zmíněnými kontejnery je pošle na datovou vrstvu, metodě *searchFiles* DAO objektu *IndexedFileDAO*. Metoda na základě nastavených filtrů vytvoří seznam podmínek reprezentovaných třídou *Predicate*. Nejprve vytvoří podmínku pro prohledání každého získaného paměťového zařízení zvlášť a následně tyto podmínky spojí disjunkcí do jedné podmínky. Vytvořenou podmínku spojí s podmínkami pro ostatní filtry (velikost, datum vytvoření/změny/posledního přístupu, jméno souboru) pomocí konjunkce. Následně připraví a spustí dotaz do databáze pomocí *CriteriaQuery* a získané výsledky vrátí logické vrstvě. Tam se indexované soubory uloží do *ObservableList<IndexedFile>*, což je seznam, u kterého konzument může sledovat změny. Konzument je v tomto případě tabulka výsledků v prezentační vrstvě, která zpracuje nalezené indexované soubory a zobrazí jejich atributy jako výsledky hledání. Výhodou je, že JavaFX komponenta pro tabulky *tableView* implicitně podporuje řazení sloupců a nemusím ho proto implementovat.

4.5 Lokalizace

Pro implementaci lokalizace jsem se inspiroval tutoriálovým řešením z webu *JavaFXPedia* [41]. Výhodou tohoto řešení je, že můžu přepínat lokalizaci za běhu programu bez nutnosti obnovení scény. Návrh využívá zdrojů (*resources*) v podobě souborů obsahujících lokalizované řetězce pro všechny texty v programu. V mém případě implementuji českou a anglickou lokalizaci, tudíž mám dva soubory ve zdrojích: *messages_cs.properties* a *messages_en.properties*. Každý překlad se skládá z klíče a hodnoty, například „search.filters=Filtry“. První je klíč, poté hodnota.

O změnu lokalizace se stará třída *I18N* (zkratka pro slovo *internationalization*, mezi I a N je 18 písmen), která v sobě ukládá aktuálně nastavenou lokalizaci a umožňuje získávání řetězců ze zdrojů podle klíče. Také implementuje metodu pro vytvoření vazby (*binding*) mezi řetězcem v rozhraní programu a lokalizovaným řetězcem ze zdrojů. Díky tomu při změně lokalizace dojde k okamžité změně řetězců v uživatelském rozhraní bez nutnosti obnovovat scénu. Bylo tedy potřeba při inicializaci všech komponent v rozhraní vytvořit vazbu mezi textem u komponenty a lokalizovanými řetězci pomocí klíče.

4.6 Výsledná aplikace

Výsledný program jsem nazval *Indexer*. Obrázky z finální verze programu si lze prohlédnout v příloze B. Pokud se porovnají s drátovým modelem v příloze A, tak si lze všimnout, že jsem navržený vzhled až na pár výjimek dodržel. Program implementuje všechny funkce vyplývající ze zadání. Stejně tak plní všechny funkční a nefunkční požadavky. Program jsem otestoval na operačních systémech *Windows 10* a *Ubuntu 16.04* a můžu zaručit stoprocentní funkčnost. Neměl jsem možnost program vyzkoušet v operačním systému *macOS* od společnosti *Apple*, takže nemůžu zaručit bezproblémovou funkčnost.

Testování ukáže, jak dobře je navrženo rozhraní programu a jak je srozumitelné pro uživatele. Stejně tak tím otestuji, zda jsou funkce programu pro uživatele dostačující.

Kapitola 5

Testování

Poslední kapitola popisuje testování. Program jsem testoval v průběhu implementace na funkčnost vyvinutých komponent. Po finalizaci programu jsem otestoval všechny funkce mezními hodnotami a chybnými vstupy. Dále jsem vytvořil testovací scénář pro uživatelské testování, které je popsáno v nadcházející části.

5.1 Uživatelské testování

Uživatelské testování jsem původně zamýšlel provést na zástupcích cílové skupiny. Kvůli nešťastným okolnostem (koronavirus) bych ale musel takové testování provést prostřednictvím internetového videohovoru a sdílením plochy. To se mi nezdálo nejšťastnější pro sledování reakcí uživatele na jednotlivé úkoly a tak jsem se rozhodl pro testování čtyř členů mé rodiny. Dva zástupci mé rodiny mají velice omezené počítačové dovednosti, ale alespoň jsem podle nich mohl zjistit, zda je můj program přístupný i pro naprosté laiky.

5.1.1 Testovací scénář

Testovací scénář se skládá z počátečního stavu, ve kterém započalo testování uživatele, a samotného testovacího scénáře. Scénář se skládá z bodů, které musel uživatel splnit. Já u každého bodu vyhodnotil, jak si uživatel vedl a uvedl jsem případný návrh vylepšení. Pokud se uživatel někde zasekl a nevěděl, jak pokračovat dál, musel jsem mu pomoc. Všechny tyto interakce s uživatelem jsou také uvedeny ve vyhodnocení. Před samotným testem jsem každému uživateli zvlášť vysvětlil funkci programu a pojem indexování.

Počáteční stav

Uživatel začíná se spuštěným programem **Indexer**, ve kterém je vytvořen index pro oddíl **(C:) OS**.

Testovací scénář

1. Uživatel připojí připravený externí disk k počítači a obnoví seznam připojených zařízení.
2. Uživatel vybere připojené zařízení **(E:) Seagate HDD** a spustí vytváření indexu.
3. Uživatel vytvářený index pojmenuje jako **Externí HDD**. V tomto indexu se budou indexovat metadata **velikost**, **datum vytvoření** a **datum změny**. Z indexu bude vyloučen adresář **Osobní** a **Foto**. Index nebude evidovat textové soubory (přípona **.txt**) a archivy ZIP (přípona **.zip**).
4. Po vyplnění předchozích údajů uživatel spustí vytváření indexu a počká na jeho dokončení.
5. Uživatel vyhledá všechny PDF soubory (přípona **.pdf**) **změněné před rokem 2018** s velikostí **větší jak 100 kB**. Vyhledávání provede pouze na paměťovém médiu **(E:) Seagate HDD** [**Externí HDD**].
6. Uživatel seřadí výsledky vzestupně podle velikosti.
7. Uživatel vybere soubor **Bila_nemoc.pdf** a otevře jeho umístění ve výchozím správci souborů.
8. Uživatel smaže tento soubor a následně aktualizuje index pro paměťové médium **(E:) Seagate HDD**.
9. Uživatel změní lokalizaci programu na angličtinu. Pokud chce, může jazyk přepnout zpět na češtinu.
10. Uživatel odpojí externí disk **(E:) Seagate HDD** od počítače a aktualizuje seznam paměťových zařízení.
11. Uživatel vybere **Externí HDD** v odpojených zařízeních a odstraní index.

5.1.2 Test s uživatelem č. 1

- **Věk:** 27
- **Pohlaví:** žena
- **Dovednosti s počítačem:** mírně pokročilé

a. Uživatel připojí připravený externí disk k počítači a obnoví seznam připojených zařízení.

Obnovení seznamu provedla, ale nebyla si jistá, zda tlačítko pro obnovení dělá to, co chce. Zkoušela na něj najet a očekávala zobrazení vysvětlivky. Bylo by dobré buď přidat vysvětlivku při najetí nebo k tlačítku přidat text „Obnovit seznam“.

b. Uživatel vybere připojené zařízení (E:) Seagate HDD a spustí vytváření indexu.

V pořádku.

c. Uživatel vytvářený index pojmenuje jako Externí HDD. V tomto indexu se budou indexovat metadata velikost, datum vytvoření a datum změny. Z indexu bude vyloučen adresář Osobní a Foto. Index nebude evidovat textové soubory (přípona .txt) a archivy ZIP (přípona .zip).

a) Pojmenování paměťového média proběhlo v pořádku.

b) Indexovaná metadata nastavena v pořádku.

c) Váhala, jak přidat vyloučené adresáře. Vylučovací metodou přešla k tlačítku „Procházet“. Přes něj v pořádku přidala adresáře *Osobní* a *Foto*. Bylo pro ní ale matoucí tlačítko „Odebrat“, nepochopila jeho funkci v rámci odstranění položek ze seznamu a myslela si, že pomocí jej odebere adresáře z indexování. Musel jsem jeho smysl vysvětlit.

d) Přípony vyloučila v pořádku. V tuto chvíli už věděla, k čemu tlačítko „Odebrat“ slouží.

d. Po vyplnění předchozích údajů uživatel spustí vytváření indexu a počká na jeho dokončení.

Vytváření indexu spustila v pořádku. Po skončení vytváření indexu si nebyla jistá, zda je vytvořený. Ukazatel průběhu po dosáhnutí sta procent zmizí, možná by bylo lepší zobrazit hlášku s informací o úspěšném dokončení indexování.

e. Uživatel vyhledá všechny PDF soubory (přípona .pdf) změněné před rokem 2018 s velikostí větší jak 100 kB. Vyhledávání provede pouze na paměťovém médiu (E:) Seagate HDD [Externí HDD].

a) Správně nalezla kartu „Vyhledávání souborů“. Správně u jména souboru nastavila, aby vyhledávání nalezlo všechny soubory končící na příponu „.pdf“.

b) Chvíli trvalo, než si všimla filtru pro nastavení data.

c) Správně použila ikonku kalendáře a nastavila datum 1.1.2018. V kalendáři ale přecházela k roku 2018 po měsících místo použití nastavení roku.

d) Velikost nastavena správně.

e) Bez problému nastavila vyhledávání pouze v paměťovém médiu „Externí HDD“.

f) Tlačítko „Vyhledat“ viděla okamžitě, v průběhu nastavování jednotlivých filtrů zkoušela jeho funkci.

f. Uživatel seřadí výsledky vzestupně podle velikosti.

V pořádku.

g. Uživatel vybere soubor Bila_nemoc.pdf a otevře jeho umístění ve výchozím správci souborů.

Zkusila na soubor dvojí kliknutí levým tlačítkem myši. Vrátila se zpět na kartu „Správa indexování“. Použití pravého tlačítka myši pro vyvolání kontextového menu ji vůbec nenapadlo.

h. Uživatel smaže tento soubor a následně aktualizuje index pro paměťové médium (E:) Seagate HDD.

Správně přešla na kartu „Správa indexování“. Nejprve klikala na tlačítko pro aktualizaci seznamu zařízení. Bylo potřeba ji navést, co se po ní chce. Po navedení provedla požadovanou akci v pořádku. Možná špatně položená otázka.

i. Uživatel změní lokalizaci programu na angličtinu. Pokud chce, může jazyk přepnout zpět na češtinu.

V pořádku. Přešla zpět na češtinu.

j. Uživatel odpojí externí disk (E:) Seagate HDD od počítače a aktualizuje seznam paměťových zařízení.

V pořádku odpojila externí disk, ale nevěděla, jak aktualizovat seznam. Náhodou to udělala, ale myslela si, že to není správný postup. Možná špatně položená otázka, protože na počátku testování krok obnovení seznamu zařízení proběhl lépe.

k. Uživatel vybere Externí HDD v odpojených zařízeních a odstraní index.

V pořádku. Odstranění z databáze trvalo pár vteřin, což ji zneklidnilo. Možná by bylo lepší zobrazit ukazatel postupu.

5.1.3 Test s uživatelem č. 2

- **Věk:** 63
- **Pohlaví:** muž
- **Dovednosti s počítačem:** základní

a. Uživatel připojí připravený externí disk k počítači a obnoví seznam připojených zařízení.

Nejprve vybral paměťové zařízení „C“ a pokusil se aktualizovat index, ale po upozornění zrušil aktualizaci a v pořádku obnovil seznam.

b. Uživatel vybere připojené zařízení (E:) Seagate HDD a spustí vytváření indexu.

V pořádku.

c. Uživatel vytvářený index pojmenuje jako Externí HDD. V tomto indexu se budou indexovat metadata velikost, datum vytvoření a datum změny. Z indexu bude vyloučen adresář Osobní a Foto. Index nebude evidovat textové soubory (přípona .txt) a archivy ZIP (přípona .zip).

a) Pojmenování paměťového zařízení a vybrání množiny metadat proběhlo v pořádku.

b) U vyloučení adresářů klikl správně na tlačítko „Procházet“, otevřel složku *Osobní*, ale nevěděl, jak ji přidat do seznamu vyloučených adresářů. Musel jsem poradit.

c) Omylem napsal příponu textových souborů jako „txt.“ s tečkou na konci, na což program zareagoval chybovou hláškou. Díky ní identifikoval chybu a opravil problém. Poté správně vyřadil příponu „.zip“.

d. Po vyplnění předchozích údajů uživatel spustí vytváření indexu a počká na jeho dokončení.

V pořádku.

e. Uživatel vyhledá všechny PDF soubory (přípona .pdf) změněné před rokem 2018 s velikostí větší jak 100 kB. Vyhledávání provede pouze na paměťovém médiu (E:) Seagate HDD [Externí HDD].

a) Správně přešel do karty „Vyhledávání souborů“. Klikl na tlačítko „Vyhledat“ bez použití filtrů. Musel jsem mu ukázat menu s filtry, které neviděl.

b) Následně přešel do filtru „Název souboru“. Tam správně napsal příponu „.pdf“, ale filtr nastavil na hodnotu „neobsahuje“. Musel jsem poradit.

c) Přešel do filtru „Datum“ a nastavil pro datum změny správně možnost „před“. Uživatele zmátlo, že filtry pro jednotlivá data nejsou

zarovnány do řádku, ale do sloupce. Poté se ale zorientoval a pomocí kalendáře nastavil 1.1.2018. I tento uživatel přepínal měsíc po měsíci od aktuálního data, aby se dostal k lednu 2018.

d) Velikost nastavil v pořádku.

e) S nastavením vyhledávání pouze pro „Externí HDD“ jsem musel pomoc.

f) Po spuštění vyhledávání program nic nenalezl. Uživatel nevěděl, kde je problém. Po provedení kontroly filtrů jsme zjistil, že uživatel zadal příponu „.pdf“ s mezerou na konci a proto program nic nenalezl. To je chyba na mé straně, měl bych odstraňovat přebytečné mezery na začátku a konci vstupů.

f. Uživatel seřadí výsledky vzestupně podle velikosti.

Uživatel nezná používané řešení u tabulek, kde stačí kliknout na název sloupce pro seřazení dat. Na tento problém nebudu brát zřetel, souvisí s počítačovými dovednostmi uživatele.

g. Uživatel vybere soubor Bila_nemoc.pdf a otevře jeho umístění ve výchozím správci souborů.

Uživatel dvakrát poklepal na správný soubor, což nic neudělalo. Nepřišel na to, že musí kliknout pravým tlačítkem myši. Jedná se o opětovný problém, bude se muset vyřešit.

h. Uživatel smaže tento soubor a následně aktualizuje index pro paměťové médium (E:) Seagate HDD.

V pořádku.

i. Uživatel změní lokalizaci programu na angličtinu. Pokud chce, může jazyk přepnout zpět na češtinu.

Umístění ikonek pro lokalizaci nalezl bez problému.

j. Uživatel odpojí externí disk (E:) Seagate HDD od počítače a aktualizuje seznam paměťových zařízení.

V pořádku.

k. Uživatel vybere Externí HDD v odpojených zařízeních a odstraní index.

V pořádku.

5.1.4 Test s uživatelem č. 3

- **Věk:** 31
- **Pohlaví:** muž
- **Dovednosti s počítačem:** mírně pokročilé

a. Uživatel připojí připravený externí disk k počítači a obnoví seznam připojených zařízení.

V pořádku.

b. Uživatel vybere připojené zařízení (E:) Seagate HDD a spustí vytváření indexu.

V pořádku.

c. Uživatel vytvářený index pojmenuje jako Externí HDD. V tomto indexu se budou indexovat metadata velikost, datum vytvoření a datum změny. Z indexu bude vyloučen adresář Osobní a Foto. Index nebude evidovat textové soubory (přípona .txt) a archivy ZIP (přípona .zip).

Celý tento proces proběhl v pořádku.

d. Po vyplnění předchozích údajů uživatel spustí vytváření indexu a počká na jeho dokončení.

Klikl na tlačítko „Vytvořit index“ a čekal na dokončení. Po zmizení ukazatele průběhu si nebyl jistý, zda je index úspěšně vytvořený.

e. Uživatel vyhledá všechny PDF soubory (přípona .pdf) změněné před rokem 2018 s velikostí větší jak 100 kB. Vyhledávání provede pouze na paměťovém médiu (E:) Seagate HDD [Externí HDD].

a) Uživatel nenašel kartu „Vyhledávání souborů“. Musel jsem poradit. Možná bych měl zvětšit písmo karet a zvýraznit je.

b) Velikost nastavil v pořádku.

c) Datum změny nastavil správně na filtr „před“. Zkusil napsat vstup „2018“, což pole nevzalo jako validní vstup. Následně klikl na kalendář a vybral správné datum.

d) Filtr pro název souboru nastavil správně.

e) U filtru pro výběr paměťových zařízení nevěděl, jak nastavit vyhledávání pouze pro „Externí HDD“. Zkoušel klikat na zaškrtávací pole u oddílu „C“, aby zrušil zaškrtnutí, ale měl nastavený přepínač na prohledání všech paměťových zařízení. Přepínačů si nevšiml, musel jsem ho popostrčit. Poté filtr nastavil v pořádku.

f) Tlačítko „Vyhledat“ našel ihned.

f. Uživatel seřadí výsledky vzestupně podle velikosti.

V pořádku.

g. Uživatel vybere soubor Bila_nemoc.pdf a otevře jeho umístění ve výchozím správci souborů.

Označil správný soubor. Chvíli hledal tlačítko pro otevření umístění, ale následně klikl pravým na řádek se souborem a otevřel jeho umístění.

h. Uživatel smaže tento soubor a následně aktualizuje index pro paměťové médium (E:) Seagate HDD.

V pořádku.

i. Uživatel změní lokalizaci programu na angličtinu. Pokud chce, může jazyk přepnout zpět na češtinu.

V pořádku.

j. Uživatel odpojí externí disk (E:) Seagate HDD od počítače a aktualizuje seznam paměťových zařízení.

V pořádku.

k. Uživatel vybere Externí HDD v odpojených zařízeních a odstraní index.

Klikl na tlačítko „Odebrat index“ a chvíli nevěděl, zda probíhá nějaká akce. Měl bych přidat ukazatel postupu pro situace, kdy se všechny soubory z databáze neodstraní ihned a akce trvá několik sekund.

5.1.5 Test s uživatelem č. 4

- **Věk:** 54
- **Pohlaví:** žena
- **Dovednosti s počítačem:** základní

a. Uživatel připojí připravený externí disk k počítači a obnoví seznam připojených zařízení.

Nebyla si jistá, zda tlačítko pro obnovení seznamu dělá to, co má. Uvítala by text u tlačítka.

b. Uživatel vybere připojené zařízení (E:) Seagate HDD a spustí vytváření indexu.

V pořádku.

c. Uživatel vytvářený index pojmenuje jako Externí HDD. V tomto indexu se budou indexovat metadata velikost, datum vytvoření a datum změny. Z indexu bude vyloučen adresář Osobní a Foto. Index nebude evidovat textové soubory (přípona .txt) a archivy ZIP (přípona .zip).

a) Pojmenování proběhlo v pořádku. Možná jsem měl rozepsat všechny body zvlášť, byla chvíli zmatená z celého bodu zmatená.

b) Indexovaná metadata vybrala správně.

c) Nevěděla, jak odstranit z indexování adresáře. Navedl jsem ji k tlačítku „Procházet“. Tam následně chtěla vybrat oba adresáře *Osobní* a *Škola* najednou, což ji program neumožnil. Následně celou akci provedla správně. Nepochopila ale smysl seznamu vyloučených adresářů a chtěla je pomocí tlačítka „Odstranit“ odebrat. Musel jsem vysvětlit, jak je to myšleno.

d) Testovaná uživatelka neví, co je to přípona souboru. Musel jsem pomoci a vysvětlit, co je tím myšleno. Úkol s vyloučením přípon z indexování nebudu brát v zřetel.

d. Po vyplnění předchozích údajů uživatel spustí vytváření indexu a počká na jeho dokončení.

V pořádku.

e. Uživatel vyhledá všechny PDF soubory (přípona .pdf) změněné před rokem 2018 s velikostí větší jak 100 kB. Vyhledávání provede pouze na paměťovém médiu (E:) Seagate HDD [Externí HDD].

a) Správně přešla na kartu „Vyhledávání souborů“, ale nevěděla, jak nastavit vyhledávání pouze „.pdf“ souborů. Nevšimla si ani menu s filtry, možná by chtělo zvýraznit.

b) U data správně vyplnila filtr na „před“ a napsala rok 2018, ale to pole pro datum nepřijímá jako validní vstup. Následně datum nastavila pomocí kalendáře.

c) Velikost a filtr pro paměťové médium nastavila správně.

f. Uživatel seřadí výsledky vzestupně podle velikosti.

Nevěděla, jak seřadit výsledky. Není jí známá standardní funkce tabulek a jejich řazení kliknutím na název sloupce. Nebudu brát v potaz kvůli základním zkušenostem s počítači.

g. Uživatel vybere soubor Bila_nemoc.pdf a otevře jeho umístění ve výchozím správci souborů.

V pořádku. Nejprve zkusila kliknutí levým tlačítkem myši, ale následně zkusila pravé tlačítko myši a otevřela umístění souboru.

h. Uživatel smaže tento soubor a následně aktualizuje index pro paměťové médium (E:) Seagate HDD.

Uživatelka správně použila tlačítko „Aktualizovat index“, bohužel ale měla zrovna vybráno jiné paměťové médium. Možná by bylo dobré více zvýraznit vybrané paměťové zařízení v jejich seznamu. Po upozornění akci správně zrušila a aktualizovala správný index.

i. Uživatel změní lokalizaci programu na angličtinu. Pokud chce, může jazyk přepnout zpět na češtinu.

V pořádku.

j. Uživatel odpojí externí disk (E:) Seagate HDD od počítače a aktualizuje seznam paměťových zařízení.

Uživatelka hledala tlačítko pro obnovení seznamu vedle popisku „Odpojená zařízení“. Možná by bylo dobré přidat ikonku pro obnovení seznamu i tam.

k. Uživatel vybere Externí HDD v odpojených zařízeních a odstraní index.

V pořádku.

5.1.6 Vyhodnocení uživatelského testování

Uživatelské testování proběhlo na čtyřech členech mé rodiny. Dva z nich mají naprosto základní a další dva mírně pokročilejší zkušenosti s počítači. To je výhoda pro zjištění, zda programu rozumí i naprostí laici. Na druhou stranu na to musím brát u některých bodů zřetel, protože dvěma uživatelům dělaly problémy i základní operace s počítačem. Nicméně i díky nim jsem zjistil spoustu užitečných informací a způsobů, jak program vylepšit a zpříjemnit jeho používání. Takové informace bych u zkušeného uživatele počítače nezískal. Program není ani zdaleka dokonalý a v následujících řádcích uvádím seznam nejčastějších chyb a jejich řešení.

Tlačítko pro obnovení seznamu

K tlačítku pro obnovení seznamu paměťových zařízení by se měla přidat vysvětlivka, která se zobrazí po najetí na tlačítko. Druhá možnost je přidat text „Obnovit seznam“ vedle ikonky tlačítka. Problém byl i s tím, že toto tlačítko je jen vedle popisku „Připojená zařízení“ a chybí vedle popisku „Odpojená zařízení“, kde ho hledala jedna z testovaných uživatelek. Bylo by dobré ho přidat i tam.

Formulář pro vytvoření indexu

S pojmenováním paměťového média a nastavením indexovaných metadat neměl problém žádný uživatel. Problém byl s vyloučením adresářů, kde uživatelé nejprve přidali adresář do seznamu vyloučených, ale pak ho z něj chtěli ve dvou případech odebrat, protože neporozuměli, k čemu tlačítko „Odebrat“ slouží. Řešením by mohlo být udělat ho méně nápadné nebo upřesnit popisek tlačítka na „Odebrat ze seznamu“. To by mohlo vysvětlit, že tím uživatel neodstraní adresář z indexování, ale jen ze seznamu. Stejná změna by byla provedena i u tlačítka „Odebrat“ u seznamu přípon.

Hláška o úspěšném dokončení indexování

Tři uživatelé si nebyli jistí, zda indexování skončilo úspěšně. Po dokončení indexování by se tak měla zobrazit hláška s informací o úspěšném dokončení akce. To by se vztahovalo k vytváření, aktualizaci a odebrání indexu.

Kalendář

Použitý kalendář není optimální, všichni uživatelé přecházeli k lednu 2018 pomocí přepínání měsíců od aktuálního data místo vybrání roku a měsíce. Řešením je vybrat kalendářovou komponentu s intuitivnějším rozhraním.

Otevření umístění souboru

Dva uživatelé zkoušeli otevřít umístění souboru pomocí dvojího kliknutí levým tlačítkem myši a nenapadlo je použít pravé tlačítko myši. Nemyslím si, že je dobré řešení otevřít umístění pomocí dvojího kliknutí levým tlačítkem myši, protože tato akce více evokuje otevření souboru. Ideální by bylo přidat do rozhraní tlačítko, kterým lze otevřít umístění daného souboru a zachovat kontextovou nabídku pomocí pravého tlačítka myši.

Odstranění indexu a vyhledávání

U obou z těchto akcí se může stát, že trvají několik sekund, pokud program zpracovává velké množství dokumentů. Kvůli tomu byli někteří uživatelé nervózní a nevěděli, zda probíhá nějaká akce po kliknutí na tlačítko a například zkusili kliknout znovu. Řešením je u těchto akcí zobrazit stejný ukazatel postupu jako při vytváření a aktualizaci indexu.

Zvýraznění menu s filtry

Dva nezkušení uživatelé měli problém nalézt menu s filtry na obrazovce pro vyhledávání. Řešením je změnit styl tlačítek tak, aby bylo jasné, že se jedná o položky menu. Tlačítka by měla reagovat na přejetí myši. Po vybrání jednoho z filtrů by mělo tlačítko zůstat zvýrazněné, aby uživatel věděl, v jakém filtru se nachází.

Popisky u filtrů
Uživatelé u jednotlivých filtrů nevěděli, co které pole znamená. U každého filtru bych měl přidat popisky typu „Výběr podmínky“, „Hodnota“ či u data a času přidat popisky „Datum“ a „Čas“.

Změnit zarovnání filtrů pro datum a čas
Uživatelům by bylo přirozenější, kdyby u filtru pro datum a čas byly jednotlivé kolonky zarovnány do řádku a ne do sloupce. Řešením je změnit zarovnání nebo více zvýraznit oddělení polí filtrů pro datum vytvoření, změny a posledního přístupu.

Ošetření vstupů
V případě, že uživatel místo data zadá pouze rok, vypsat informační hlášku s formátem validního data. Pokud uživatel do některého z polí zadá vstup s mezerami na začátku nebo na konci, tak při zpracování vstupu tyto mezery odstranit.

Zvýraznit názvy karet
Jeden z uživatelů si nevšiml názvů karet pro správu indexování a vyhledávání, takže nevěděl, jak začít vyhledávat soubory. Řešením je zvětšit a zvýraznit názvy karet.

Výběr paměťových zařízení k prohledání
Někteří uživatelé měli problém s nastavením prohledávání pouze některých paměťových zařízení. Měli aktivovaný přepínač pro vyhledávání ve všech paměťových zařízeních, ale přesto se snažili klikat do deaktivovaného seznamu zařízení. Řešením je vypsat informační hlášku, že uživatel musí nejprve přepnout přepínač na „Vybraná paměťová zařízení“ a až pak může zvolit, která chce a která nechce prohledávat.

Výraznější zvýraznění vybraného paměťového zařízení
Na kartě „Správa indexování“ měli uživatelé občas problém s tím, že provedli akci s jiným paměťovým zařízením, než měli. Nevšimli si totiž, jaké zařízení mají aktuálně vybrané. Řešením je větší zvýraznění aktuálně vybraného paměťového zařízení, a to i v případě, kdy není seznam aktivně vybraný.

Závěr

Cílem práce bylo navrhnout a implementovat *open-source* nástroj pro snadnou evidenci různých souborových úložišť. Při vývoji byl uplatněn klasický cyklus vývoje softwaru. Analýzou existujících řešení a průzkumem u potenciálních uživatelů z cílové skupiny jsem sestavil funkční a nefunkční požadavky na program.

Na základě požadavků jsem definoval případy užití, kterými jsem popsal způsob fungování programu. Následně jsem v návrhové části vytvořil drátový model programu vycházející z případů užití. Definoval jsem architekturu a způsob uložení dat. Stanoveného návrhu jsem se držel při implementaci. Rozdíly mezi drátovým modelem a finální podobou programu lze porovnat v příloze A a B.

Implementované řešení problému pokrývá všechny požadované funkce plynoucí ze zadání. Ke každému paměťovému zařízení lze vytvořit index. Následně lze v obsahu paměťových zařízení vyhledávat, a to i v případě, kdy médium není připojeno k počítači. Takto lze snadno zjistit, na kterém médiu se hledaný soubor nachází, což výrazně zkrátí čas hledání. Vyhledávání lze upřesnit pomocí několika filtrů. Filtry se zaměřují na práci s názvem a velikostí souboru a daty (datum vytvoření, datum změny, datum posledního přístupu). U každého paměťového média lze nastavit, co se bude a co se nebude ukládat do indexu, včetně volitelných metadat. Z indexování lze vyloučit jak adresáře, tak typy souborů.

Po dokončení implementace jsem vytvořil dokumentaci kódu. Také jsem vytvořil scénář typického použití programu, který mi umožnil provést uživatelské testování se čtyřmi uživateli. Díky výsledkům testování jsem objevil problémy, které program má, a navrhl jsem jejich řešení.

Do budoucna by bylo dobré implementovat vícevláknové vytváření indexu pro zvýšení rychlosti indexování. Seznam připojených a odpojených zařízení by se mohl automaticky aktualizovat při odpojení zařízení od počítače, či naopak. Program by také mohl implementovat procházení stromové struktury spravovaných indexů, pokud si člověk nepamatuje název souboru, který hledá.

Literatura

[1] TCHELIDZE, D.: Historie datových úložišť: od děrných štítků po SSD [online]. *cnews.cz*, Prosinec 2010, [cit. 2020-02-18]. Dostupné z: https://www.cnews.cz/historie-datovych-ulozist-od-dernych-stitku-po-ssd/

[2] File Definition [online]. Říjen 2007, [cit. 2020-02-18]. Dostupné z: https://techterms.com/definition/files

[3] CHAPPLE, M.: What Is Metadata? [online]. *Lifewire*, Leden 2020, [cit. 2020-02-18]. Dostupné z: https://www.lifewire.com/metadata-definition-and-examples-1019177

[4] HOFFMAN, C.: What Is a File System, and Why Are There So Many of Them? [online]. *How-To Geek*, Září 2016, [cit. 2020-02-18]. Dostupné z: https://www.howtogeek.com/196051/htg-explains-what-is-a-file-system-and-why-are-there-so-many-of-them/

[5] Časté otázky k indexování vyhledávání ve Windows 10 [online]. Duben 2018, [cit. 2020-01-22]. Dostupné z: https://support.microsoft.com/cs-cz/help/4098843/windows-10-search-indexing-faq

[6] XML Tutorial [online]. [cit. 2020-02-10]. Dostupné z: https://www.w3schools.com/xml/

[7] soumya08: Version Control Systems [online]. [cit. 2020-02-10]. Dostupné z: https://www.geeksforgeeks.org/version-control-systems/

[8] Git [online]. [cit. 2020-02-10]. Dostupné z: https://git-scm.com/

[9] Apache Subversion FAQ [online]. [cit. 2020-02-10]. Dostupné z: https://subversion.apache.org/faq.html

[10] Indie game [online]. Duben 2017, [cit. 2020-02-10]. Dostupné z: https://www.computerhope.com/jargon/i/indie-game.htm

[11] Cloud Storage [online]. [cit. 2020-02-10]. Dostupné z: https://aws.amazon.com/what-is-cloud-storage

[12] BOUANANI, O.: How to Fund Your Games By Creating and Selling Game Assets [online]. Červenec 2015, [cit. 2020-02-11]. Dostupné z: https://gamedevelopment.tutsplus.com/articles/how-to-fund-your-games-by-creating-and-selling-game-assets--cms-24380

[13] File manager [online]. Listopad 2019, [cit. 2020-01-24]. Dostupné z: https://www.computerhope.com/jargon/f/filemana.htm

[14] File Explorer [online]. Leden 2020, [cit. 2020-01-22]. Dostupné z: https://www.computerhope.com/jargon/e/explorer.htm

[15] FISHER, T.: What Is a SEARCH-MS File? [online]. *Lifewire*, Listopad 2019, [cit. 2020-01-24]. Dostupné z: https://www.lifewire.com/search-ms-file-2622249

[16] History of Total Commander [online]. Prosinec 2009, [cit. 2020-01-26]. Dostupné z: https://www.ghisler.ch/wiki/index.php?title=History_of_Total_Commander

[17] GIBBS, M.: What is an FTP Connection? [online]. *Study.com*, [cit. 2020-01-26]. Dostupné z: https://study.com/academy/lesson/what-is-an-ftp-connection.html

[18] Everything [online]. [cit. 2020-01-28]. Dostupné z: https://www.voidtools.com/faq/#what_is_everything

[19] Lightning Fast File Search [online]. [cit. 2020-02-16]. Dostupné z: https://www.fileseek.ca/

[20] FileSeek Free vs Pro [online]. [cit. 2020-02-16]. Dostupné z: https://www.fileseek.ca/Compare/

[21] UltraSearch Features [online]. [cit. 2020-02-17]. Dostupné z: https://www.jam-software.com/ultrasearch/features.shtml

[22] ČÁPKA, D.: Lekce 4 - UML - Doménový model [online]. *ITNetwork*, Prosinec 2012, [cit. 2020-02-25]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-domenovy-model-diagram

[23] What is wireframing? [online]. [cit. 2020-03-01]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/

[24] 2020 Java Technology Report [online]. Leden 2020, [cit. 2020-02-05]. Dostupné z: https://www.jrebel.com/blog/2020-java-technology-report

[25] WARREN, D.: Java 8 vs Java 11 – What are the Key Changes? [online]. *IDRsolutions*, Květen 2019, [cit. 2020-02-06]. Dostupné z: https://blog.idrsolutions.com/2019/05/java-8-vs-java-11-what-are-the-key-changes/

[26] ŠTECHMÜLLER, P.: Lekce 1 - Úvod do JavaFX [online]. *ITNetwork*, Březen 2019, [cit. 2020-02-09]. Dostupné z: https://www.itnetwork.cz/java/javafx/uvod-do-javafx

[27] SINGH, C.: Java AWT tutorial for beginners [online]. *BeginnersBook*, [cit. 2020-02-09]. Dostupné z: https://beginnersbook.com/2015/06/java-awt-tutorial/

[28] Introduction to Java Swing [online]. [cit. 2020-02-09]. Dostupné z: https://www.zentut.com/java-swing/introduction-to-java-swing/

[29] JavaFX [online]. [cit. 2020-02-09]. Dostupné z: https://www.knowledgehut.com/tutorials/java-tutorial/java-fx

[30] Embedded Database [online]. [cit. 2020-02-09]. Dostupné z: https://www.techopedia.com/definition/30660/embedded-database

[31] Apache Derby [online]. [cit. 2020-02-09]. Dostupné z: https://db.apache.org/derby/

[32] MILLINGTON, S.: The Difference Between JPA, Hibernate and EclipseLink [online]. *Baeldung*, Prosinec 2018, [cit. 2020-02-11]. Dostupné z: https://www.baeldung.com/jpa-hibernate-difference

[33] JPA Performance Benchmark [online]. [cit. 2020-02-11]. Dostupné z: https://www.jpab.org/Hibernate/Derby/embedded/EclipseLink/Derby/embedded.html

[34] KOTAČKA, V.: Gradle, moderní nástroj na automatizaci [online]. *Zdroják*, Červen 2013, [cit. 2020-03-12]. Dostupné z: https://www.zdrojak.cz/clanky/gradle-moderni-nastroj-na-automatizaci/

[35] Project Lombok [online]. [cit. 2020-02-10]. Dostupné z: https://projectlombok.org/

[36] Operating System & Hardware Information [online]. [cit. 2020-02-10]. Dostupné z: https://github.com/oshi/oshi

[37] Harawata: AppDirs [online]. [cit. 2020-03-15]. Dostupné z: https://github.com/harawata/appdirs

[38] Simple Logging Facade for Java (SLF4J) [online]. [cit. 2020-02-11]. Dostupné z: http://www.slf4j.org/

[39] Apache Log4j 2 [online]. [cit. 2020-02-11]. Dostupné z: https://logging.apache.org/log4j/2.x/

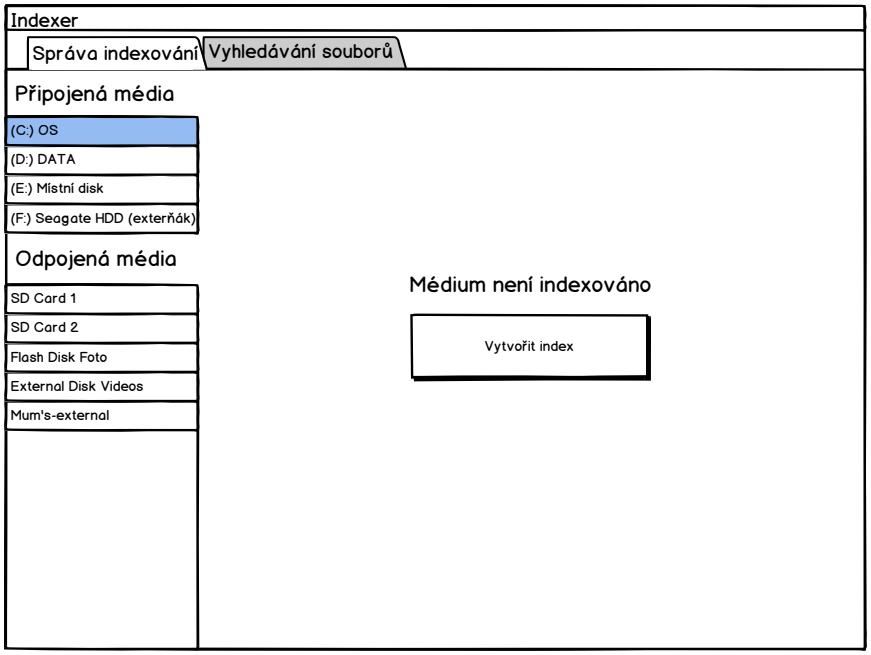
[40] HORDĚJČUK, V.: JPA (Java Persistence API) [online]. *voho*, [cit. 2020-02-11]. Dostupné z: http://voho.eu/wiki/java-jpa/

[41] Internationalization in JavaFX [online]. [cit. 2020-04-02]. Dostupné z: https://javafxpedia.com/en/tutorial/5434/internationalization-in-javafx

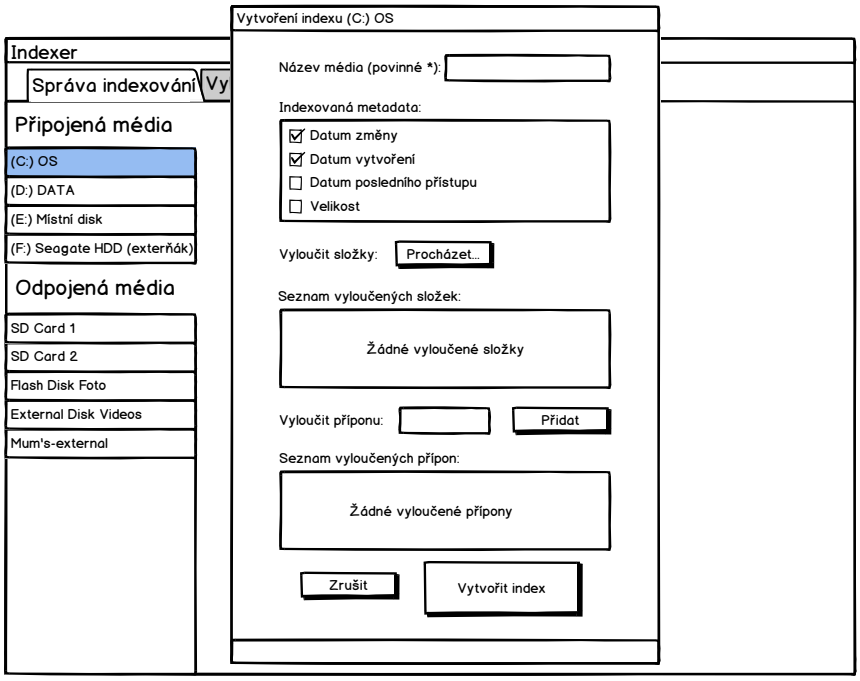
Příloha A

Drátový model programu

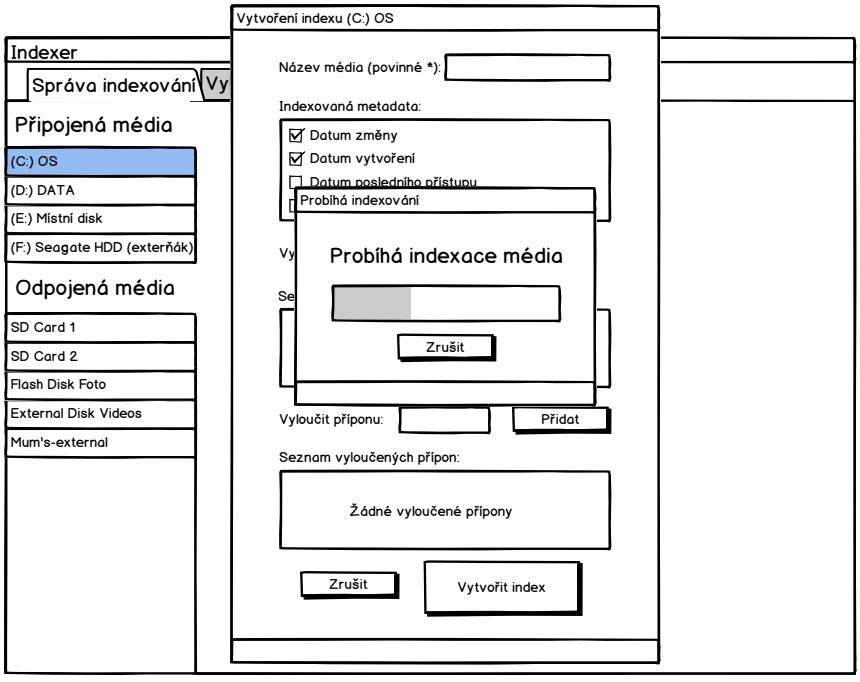
Kompletní drátový model programu, který vychází z případů užití a požadavků na program.



Obrázek A.1: Hlavní okno s neindexovaným médiem

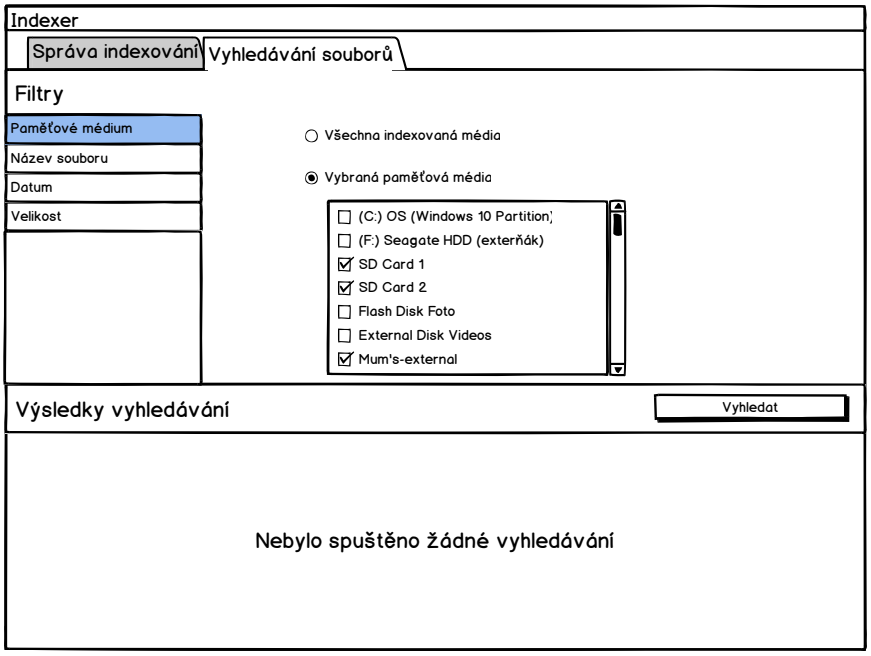


Obrázek A.2: Okno s možnostmi vytvoření indexu

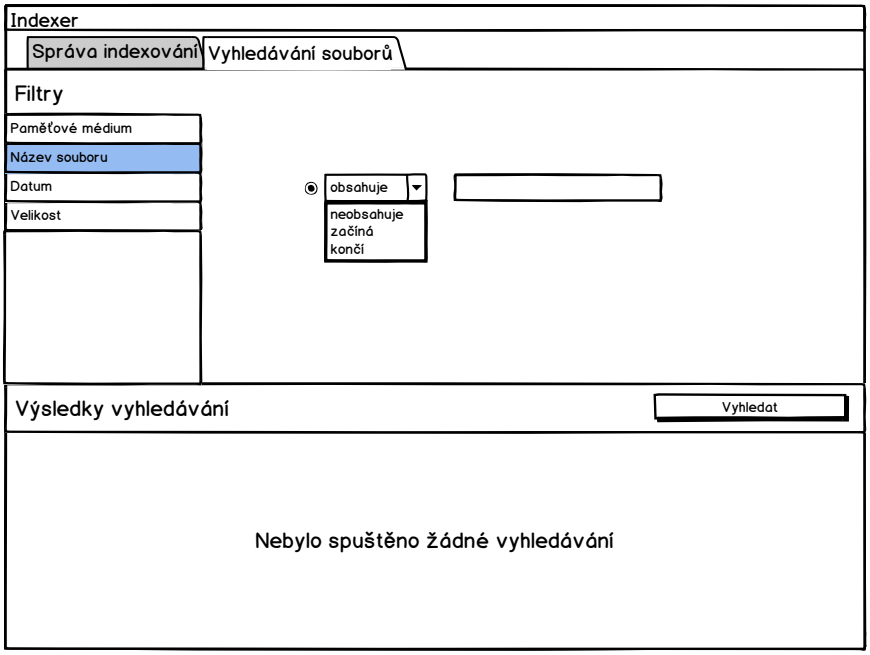


Obrázek A.3: Okno s průběhem tvorby indexu

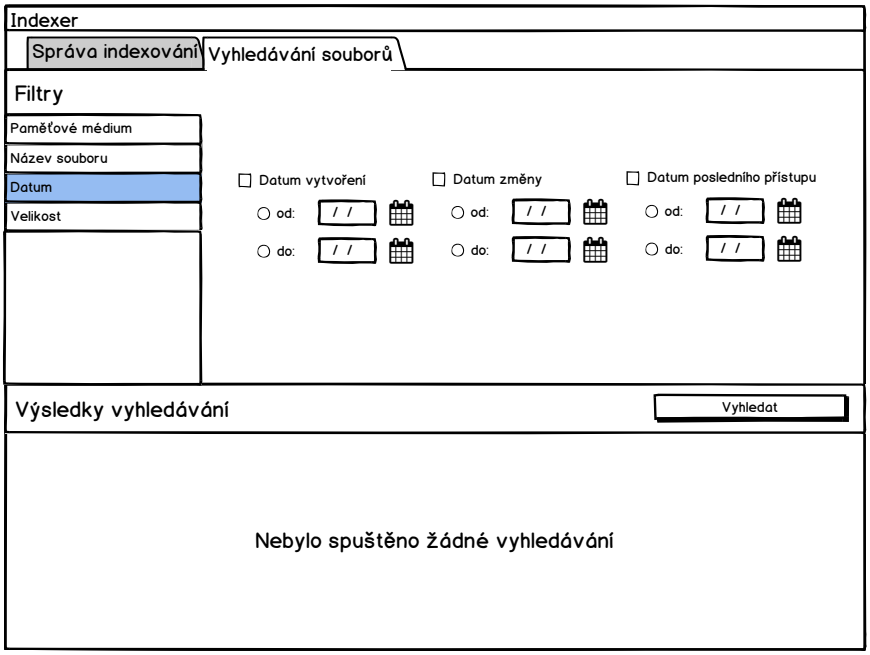
Obrázek A.4: Hlavní okno s indexovaným médiem



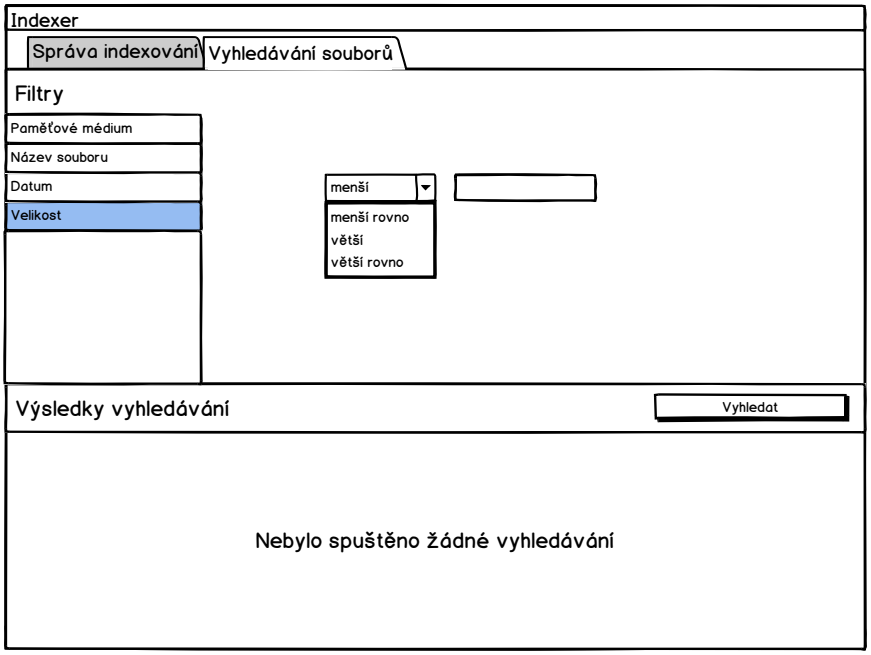
Obrázek A.5: Vyhledávací okno - filtr paměťových médií



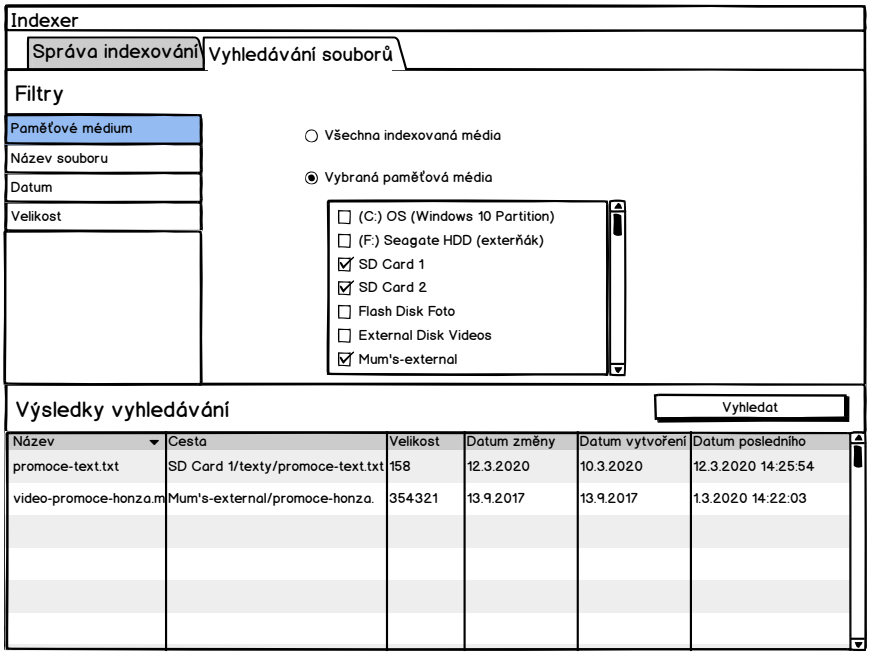
Obrázek A.6: Vyhledávací okno - filtr názvů souborů



Obrázek A.7: Vyhledávací okno - filtr pro datum



Obrázek A.8: Vyhledávací okno - filtr pro velikost souborů

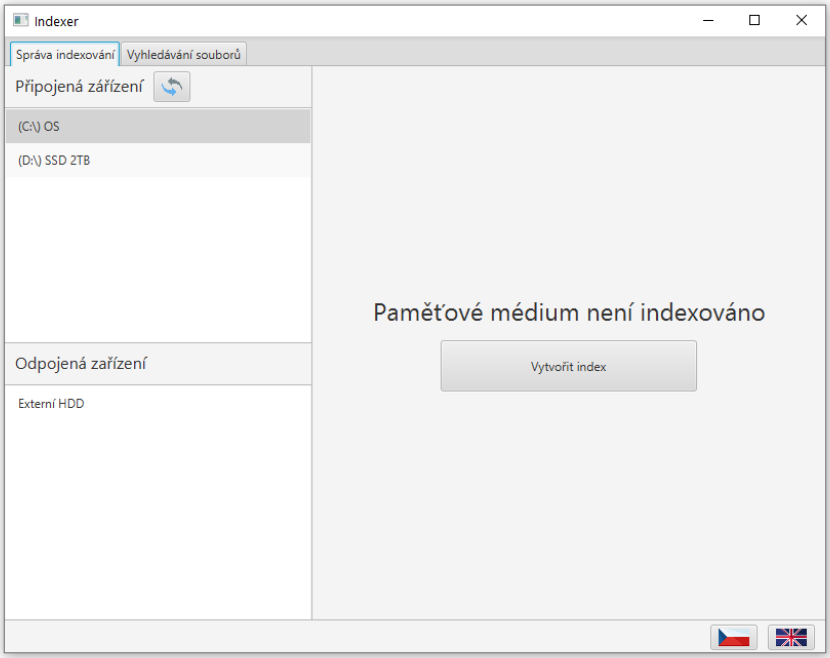


Obrázek A.9: Vyhledávací okno - výsledky vyhledávání

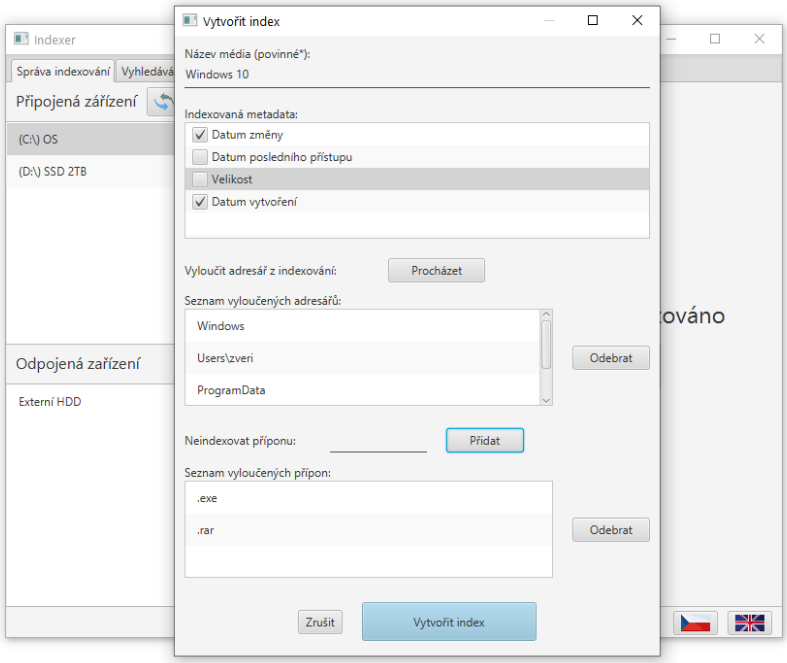
Příloha B

Finální podoba programu

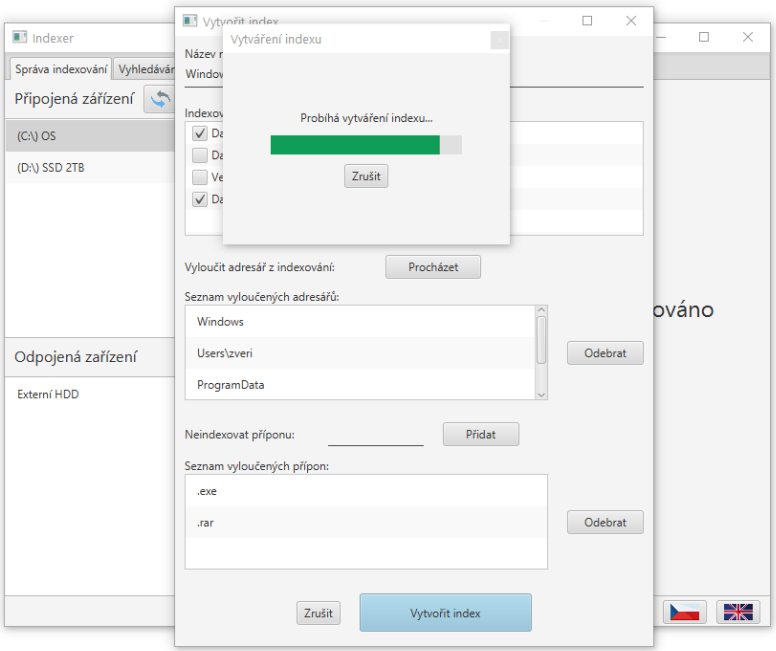
V této příloze lze vidět sadu obrázků s finální podobou implementovaného programu *Indexer*.



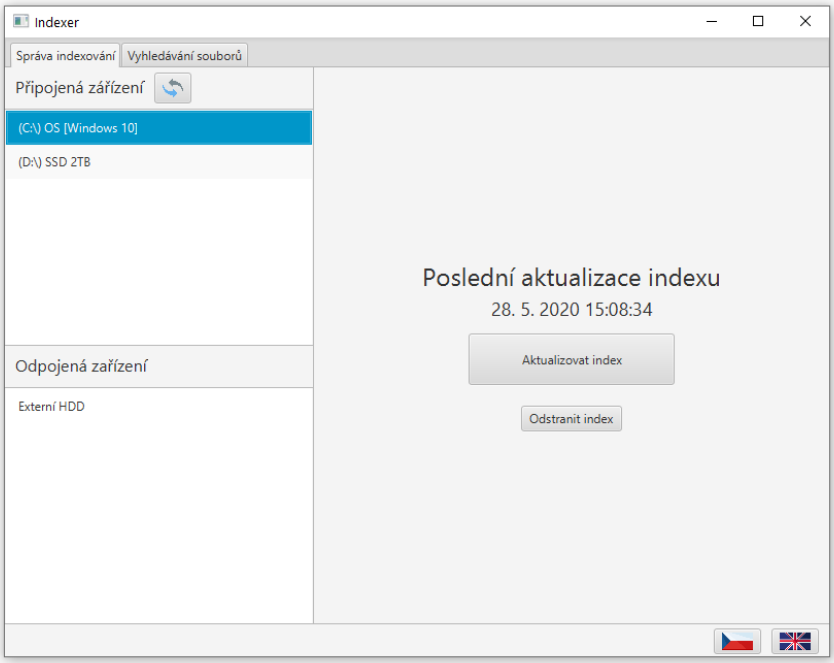
Obrázek B.1: Hlavní okno s neindexovaným médiem



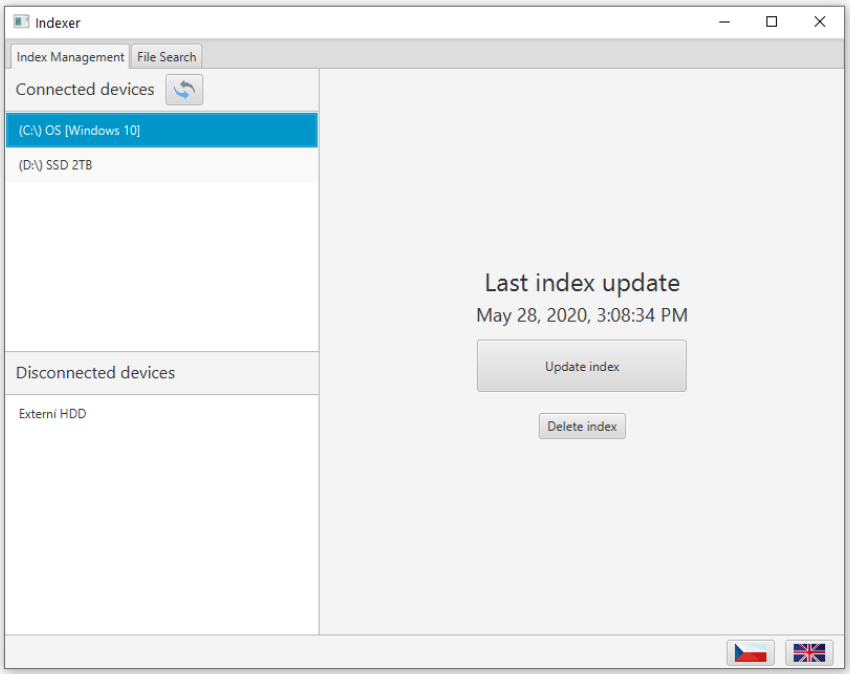
Obrázek B.2: Dialogové okno s možnostmi pro vytvoření indexu



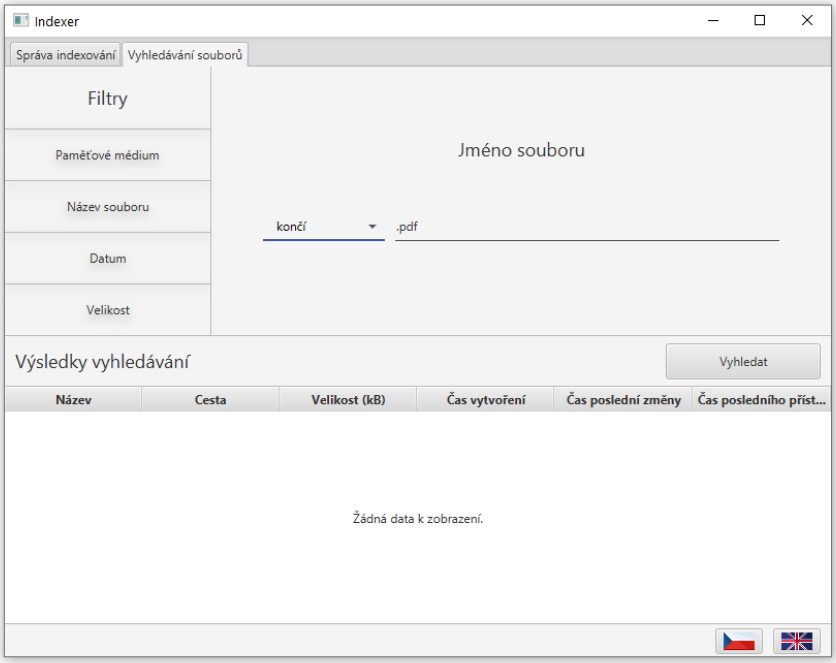
Obrázek B.3: Dialogové okno s ukazatelem průběhu



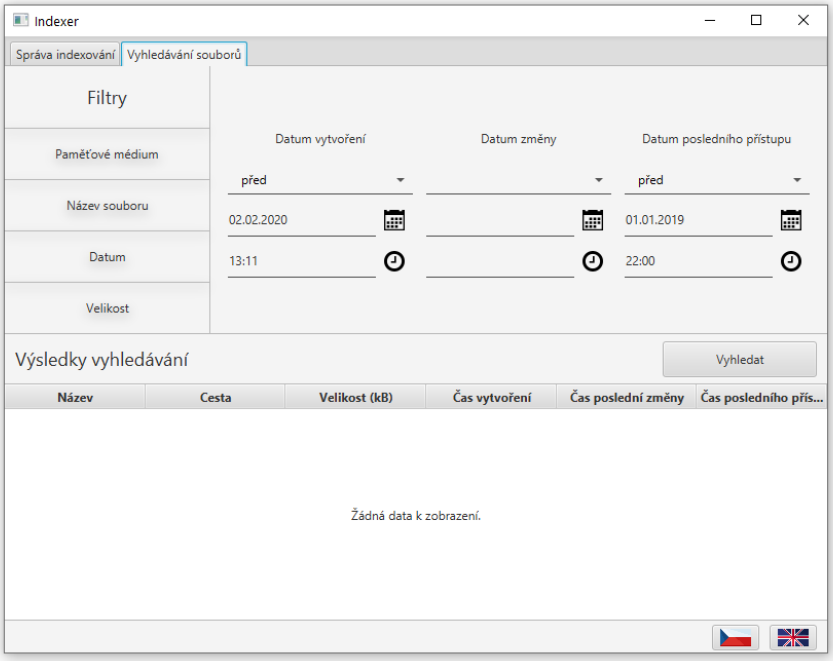
Obrázek B.4: Hlavní okno s indexovaným médiem v české lokalizaci



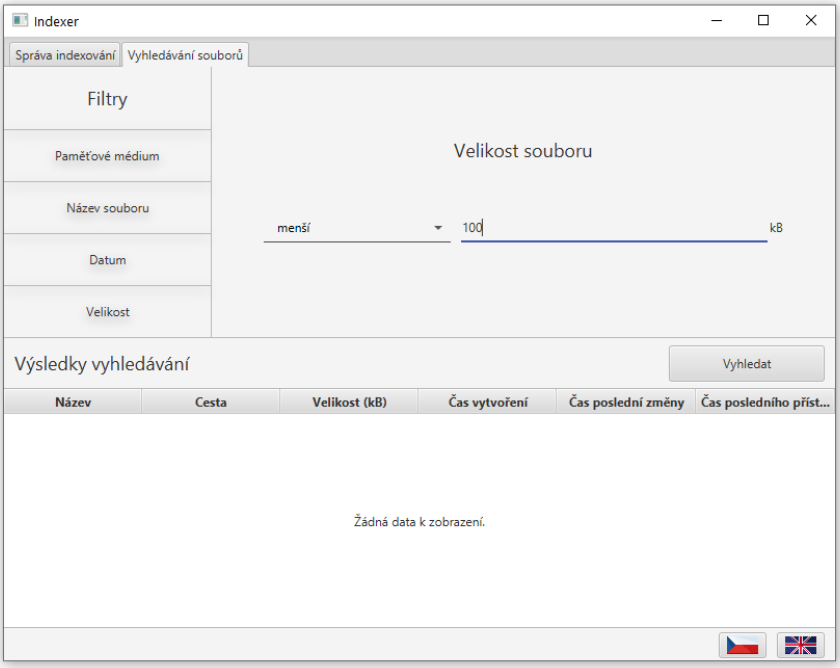
Obrázek B.5: Hlavní okno s indexovaným médiem v anglické lokalizaci



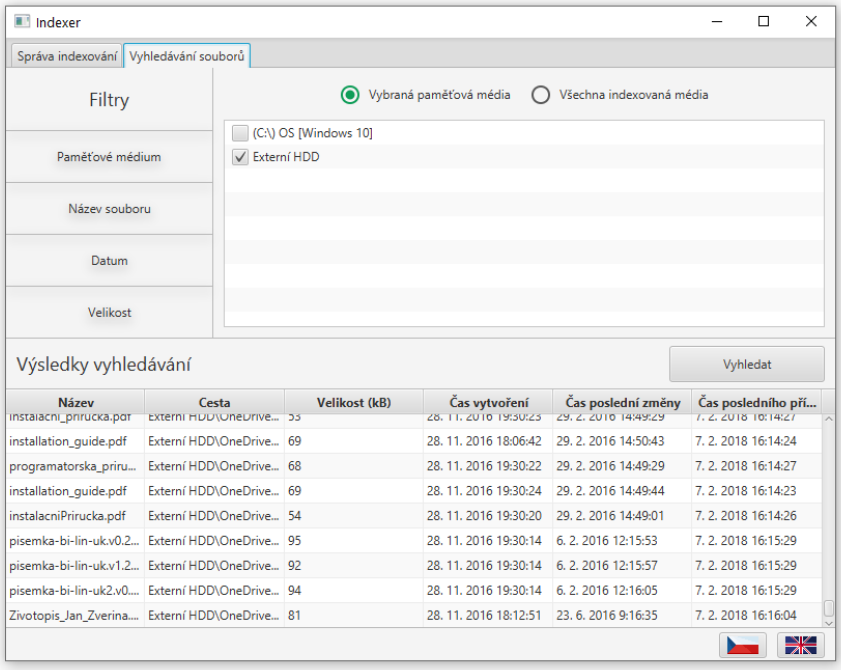
Obrázek B.6: Vyhledávací okno - filtr názvů souborů



Obrázek B.7: Vyhledávací okno - filtr pro datum



Obrázek B.8: Vyhledávací okno - filtr pro velikost souborů



Obrázek B.9: Vyhledávací okno s výsledky - filtr paměťových médií

Příloha C

Seznam použitých zkratek

CD Compact Disc

CRUD Create, Read, Update, Delete

DVD Digital Versatile Disc

ext4 Extended File System 4

FAT32 File Allocation Table 32

FTP File Transfer Protocol

GUI Graphical User Interface

HDD Hard Disk Drive

HFS+ Hierarchical File System+

IDE Integrated Development Environment

JDBC Java Database Connectivity

NTFS New Technology File System

SD Card Secure Digital Card

SQL Structured Query Language

SŘBD Systém Řízení Báze Dat

SSD Solid-State Drive

SVN Subversion

XML eXtensible Markup Language

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `executable_file`
  - `Indexer.jar` ..... spustitelný program Indexer ve formátu JAR
  - `indexer.bat` ..... dávkový soubor pro alternativní spuštění programu
  - `instalacni_prirucka.txt` ..... instalační příručka
- `javadoc` ..... dokumentace tříd
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF