Retrospektivní bibliografie české literární vědy

Implementační dokumentace

inSophy s.r.o.

22. května 2012

Obsah

Kapitola 1

Úvod

1.1 Cíl projektu

Tento dokument vznikl jako implementační dokumentace projektu digitalizace a online zpřístupnění kartotéky retrospektivní bibliografie české literární vědy. Projekt má za cíl především usnadnit vyhledávání a listování existující kartotékou bez nutnosti fyzického přístupu k lístkům. Využitím moderních technologií je možné práci s těmito lístky umožnit všem, kteří online katalog navštíví přes internet. Významným přínosem a součástí projektu je i provedení OCR rekognoskace při importu dat a možnost fulltextového vyhledávání v těchto datech. Shromážděné lístky bude dále možné v průběhu času spravovat a informace na nich obsažené stále přesněji strukturovat.

Projekt je výsledkem projektu „Digitalizace lístkového katalogu Retrospektivní bibliografie české literární vědy 1775–1945“ (VZ09004), který v rámci programu INFOZ v letech 2009–2011 laskavě finančně podpořilo Ministerstvo školství, mládeže a tělovýchovy České republiky.

Vlastní projekt je možné rozdělit do několika částí:

- návrh procesu plnění a údržby katalogu (částečně jde mimo tuto dokumentaci, protože je v režii zaměstnanců ÚČL);
- vytvoření jednoúčelových nástrojů pro zpracování a import obrázků do databáze tak, aby odpovídaly navrženému procesu;
- vytvoření webové aplikace určené k procházení a správě nashromážděných dat.

Poměrně přesné zadání projektu (včetně všech původních požadavků ze strany ÚČL) lze najít v zadávací dokumentaci. V průběhu času se tyto požadavky samozřejmě lehce měnily, usměrňovaly a stabilizovaly, jak se projekt vyvíjel.

1.2 Dostupnost

Zdrojové kódy aplikace byly předány na datových nosičích a jejich záloha je umístěna na verzovacím systému SVN společnosti inSophy s.r.o. Veškerý software od třetích stran potřebný k sestavení a provozování systému je šířen zdarma či pod svobodnou licencí. Celý proces kompilace včetně stažení potřebných závislostí a knihoven je řízen systémem Maven a je popsán v následujících kapitolách.

Kapitola 2

Instalace a oživení systému

2.1 Platforma

Systém Retrobi je *centralizovaný*. Jeho centrálním prvkem je *server*, ke kterému se připojují klienti pomocí HTTP. Klienty se zde rozumí jednoúčelové *aplikace pro import dat* (připojují se přímo k databázi) a návštěvnící ze sítě internet (připojují se k webové aplikaci). Celý systém je multiplatformní, ale otestován je pod systémem Ubuntu Linux (server) a Windows (klienti).

2.1.1 Hardware

Minimální konfigurace serveru je sestava se dvěma procesorovými jádry na 2 GHz (64 bit), 2 GiB paměti RAM a diskové pole 2 TiB (pro cca 1,5 milionu lístků s rezervou). Na této konfiguraci současně běží databázový, vyhledávací i webový server. V dokumentaci se dále předpokládá tato minimalistická varianta, ačkoliv by bylo teoreticky možné jednotlivé součásti oddělit a po korektní změně nastavení je provozovat na několika fyzicky oddělených strojích.

Výkon celého systému je závislý především na rychlosti přístupu na pevný disk. Doporučujeme proto využít např. technologie RAID, a to i kvůli vyšší spolehlivosti a odolnosti proti ztrátě dat. Další významné zvýšení výkonu by přinesl přechod na technologii SSD (solid state drive). Takový disk by byl zajímavý zejména pro umístění indexů vyhledávacího enginu, které nejsou příliš velké, ale na druhou stranu často využívané.

Jakoukoliv paměť RAM navíc je možno využít pro zvýšení maximální velikosti haldy (heap size) pro Java Virtual Machine (JVM). Doporučená velikost haldy pro webový server

je 2 GiB. Velikost haldy a další možnosti virtuálního stroje se specifikují v souboru **/etc/default/tomcat7** touto konfigurační direktivou:

```
JAVA_OPTS="-Djava.awt.headless=true -Xmx2g -XX:+UseConcMarkSweepGC"
```

2.2 Automatická instalace

Skript pro automatickou instalaci systému Retrobi nám laskavě posytl pan Mgr. Martin Krsek. Zabalený skript a jeho součásti naleznete v repositáři SVN v podadresáři **install**. Pro použití skriptu celou tuto složku nakopírujte na server, kde chcete systém Retrobi nainstalovat. Poté spusťte hlavní skript. Ten provede veškerou instalaci, konfiguraci a kompilaci systému Retrobi, takže již stačí jen spustit potřebné systémové služby a provést deploy webové aplikace do serveru Tomcat (viz následující kapitoly). Balíček WAR potřebný pro deploy se nachází ve složce **/opt/RETROBI**. Během běhu skriptu vzniknou logy s ladícími informacemi.

2.3 Manuální instalace

Poznámka: Toto je doporučený postup instalace. Pokud jste zkušený administrátor a víte co děláte, můžete samozřejmě instalaci a konfiguraci provést jinak, než je zde uvedeno.

K instalaci systému Retrobi není třeba žádných speciálních programátorských znalostí, ale provádět by ji měl spíše zkušenější správce. K instalaci je potřeba internetové připojení, možnost získat rootovská oprávnění (např. příkazem **sudo**), schopnost editovat textové soubory a obecně pracovat se soubory na disku (kopírovat, přesouvat, mazat, vytvářet). Správce musí znát základy práce s terminálem. Znalost všech příkazů ale není nutná, ty jsou popsány v dokumentaci.

Také se předpokládá znalost principů systémových oprávnění a jejich změny, například pomocí příkazů `chmod` a `chown`.

2.3.1 Operační systém

Základem systému Retrobi je operační systém Linux v distribuci Ubuntu (edice Server). Otestována byla verze **11.10** (64 bit). Tu lze stáhnout z tého adresy:

http://www.ubuntu.com/download/server/download

Operační systém se nainstaluje standardním způsobem, není nutné nic speciálního nastavovat. Po instalaci operačního systému je třeba zprovoznit připojení k internetu (mimo rozsah této dokumentace). Připojení většinou proběhne automaticky, pokud je server připojen do sítě s DHCP serverem.

2.3.2 Potřebné balíčky

Verze uvedené u balíčků nejsou přesné a závazné, ale minimální možné. Automatické aktualizace operačního systému Ubuntu zajistí, že budou vždy použity nejnovější verze se všemi dostupnými opravami chyb a vylepšeními. Číslování verzí je hierarchické. To například znamená, že program verze 1 může být aktualizován na verzi 1.0.1, poté na 1.1 a nakonec na verzi 1.2.5. Pokud vyjde verze 2, nemusí být zpětně kompatibilní. Toto ale posuzuje správce balíčků (na straně Ubuntu), který se stará o jejich bezproblémovou aktualizaci a kompatibilitu.

Nejprve doporučujeme nainstalovat správce souboru **mc** (Midnight Commander), který umožňuje procházet disk i editovat textové konfigurační soubory (klávesa F4). Dále je nutné nainstalovat **OpenJDK** pro Javu 1.6, servletový kontejner **Apache Tomcat**, databázi **CouchDB**, poštovní (e-mailový) server **Postfix**, buildovací systém **Maven** a systémy pro správu verzí **Subversion** a **Git**.

1. **sudo apt-get install mc maven2 subversion git**
2. **sudo apt-get install openjdk-6-jdk tomcat7-admin**
3. **sudo apt-get install couchdb postfix python**

Otestované verze:

- OpenJDK 1.6.b23
- Apache Tomcat 7.0.23
- CouchDB 1.0.1
- Postfix 2.8.7
- Maven 2.2.1

Adresář	Zkratka	Zapisující aplikace	Odhad velikosti
data CouchDB	CESTA_DB_DATA	CouchDB	2 TiB
indexy pro Lucene	CESTA_INDEX	CouchDB-Lucene	5 GiB
CSV logy	CESTA_CSV	webová aplikace	1 GiB
konfigurační soubory	CESTA_CONFIG	webová aplikace	1 MiB

Tabulka 2.1: Tabulka adresářů a jejich velikostí

- svn 1.6.12
- git 1.7.1
- Python 2.3

2.3.3 Adresářová struktura

Pro vyšší přehlednost je vhodné připravit adresáře uvedené v tabulce 2.1 a zajistit, aby k nim odpovídající aplikace měly potřebná oprávnění pro zápis. Je možné využít symlinků a jednotlivé adresáře umístit na jiné fyzické disky či stroje. V tabulce jsou uvedeny jednotlivé adresáře a jejich předpokládaná velikost po naplnění všemi daty. Proto rozvažte, na který disk který soubor umístíte.

2.4 Konfigurace

2.4.1 Databáze CouchDB

Databáze CouchDB se konfiguruje souborem, který je standardně umístěný na cestě **/etc/couchdb/local.ini** (pozor, druhý soubor **default.ini** obsahuje výchozí konfiguraci a je přepsán při každé aktualizaci CouchDB). V souboru **local.ini** je třeba nastavit tyto parametry:

```
[couchdb]
database_dir = CESTA_DB_DATA
view_index_dir = CESTA_DB_DATA
os_process_timeout = 1800000
```

```
delayed_commits = false
[httpd]
bind_address = 0.0.0.0
[couch_httpd_auth]
require_valid_user = false
[log]
file = CESTA_DB_DATA/couchdb.log
level = error
[external]
fti=/usr/bin/python CESTA_LUCENE/tools/couchdb-external-hook.py
[httpd_db_handlers]
_fti = {couch_httpd_external, handle_external_req, <<"fti">>}
```

Databázový server CouchDB je dále nutné zabezpečit např. firewallem tak, aby se k němu nemohl připojit žádný klient kromě vybraných počítačů, na kterých budou provozována „udělátka“ a webová aplikace (většinou není nutné speciálně řešit, protože webová aplikace běží na stejném stroji – *localhost*).

2.4.2 Vyhledávací servlet CouchDB-Lucene

Vyhledávací servlet se konfiguruje pomocí **.ini** souboru, který se nachází na tomto umístění:

```
retrobisrc/couchdb-lucene/src/main/resources/couchdb-lucene.ini
```

Před kompilací vyhledávacího servletu CouchDB-Lucene je v něm potřeba nastavit tyto parametry:

```
[lucene]
# cesta k vyhledavacim indexum
dir=CESTA_INDEX
# maximalni doba dotazu [ms]
# (nastavime na 30 sekund)
timeout=30000
# vychozi limit na pocet vysledku
limit=10
[local]
# zakladni URL databaze CouchDB
url=http://localhost:5984/
```

2.4.3 Server Tomcat

Ve výchozím nastavení je server Tomcat možná až příliš „upovídaný“ a podrobně loguje různé akce a přístupy. Aby do budoucna nebyl problém s nedostatkem místa na systémovém disku, je doporučeno snížit podrobnost logování a to v tomto souboru:

```
/etc/tomcat7/server.xml
```

Stačí vyhledat a zakomentovat XML uzel *Valve* související s logováním přístupů (access log), který se nachází pod uzlem *Host*:

```
<Host name="localhost" appBase="webapps"
      unpackWARs="true" autoDeploy="true">
...
<!--
<Valve className="org.apache.catalina.valves.AccessLogValve"
       directory="logs" pattern="%h %l %u %t &quot;%r&quot; %s %b"
       prefix="localhost_access_log." suffix=".txt" />
!-->
...
</Host>
```

2.4.4 Webová aplikace

Soubor web.xml

V projektu se nachází dva soubory **web.xml**: jeden pro vyhledávací servlet CouchDB-Lucene (ten měnit nemusíme) a druhý pro webovou aplikaci systému Retrobi. Tento druhý soubor se nachází mezi zdrojovým kódem v následujícím podadresáři:

```
(retrobisrc)/retrobi-web/src/main/webapp/WEB-INF/web.xml
```

Ve zmíněném konfiguračním XML souboru je obsaženo několik důležitých nastavení, které bude pravděpodobně nutné před první kompilací a spuštěním webové aplikace změnit. Po změně je vždy nutné znovu provést kompilaci a *deployment* webové aplikace (viz další kapitoly). Proměnné obsažené v konfiguračním souboru jsou shrnuty a popsány v tabulce 2.2.

Název (klíč)	Výchozí hodnota	Popis
debug	false	true = zapnout ladící režim, false = vypnout
daily_maintenance_hour	3	hodina spuštění denní údržby (0 - 23)
catalog_cache_timeout	86400000	doba platnosti cache modelu katalogu (jednotka = milisekundy)
csv_log_dir	/mnt/csv/	adresář kam budou ukládány CSV logy; musí být zapisovatelný; nemusí končit lomítkem
attribute_file	/mnt/settings/attribute.json	soubor s definicí položkového rozpisu; cíl musí být zapisovatelný; když soubor neexistuje, je vytvořen výchozí
index_file	/mnt/settings/index.json	soubor s definicí informací o indexech; cíl musí být zapisovatelný; když soubor neexistuje, je vytvořen výchozí
server_url	http://retrobi.ucl.cas.cz/	absolutní URL adresa webu použitelná např. do e-mailů
many_basket_cards	35000	nejnižší počet lístků ve schránce, při kterém se bude zobrazovat varování a budou blokovány hromadné operace; závisí na velikosti haldy; pro 1 GiB je doporučeno 35000
many_cards	100	nejvyšší počet lístků, pro který bude dlouhá úloha spuštěna ihned; tato hodnota by měla být vyšší než limit schránky návštěvíka
web_stats_url	http://retrobi.ucl.cas.cz/awstats/	Absolutní URL adresa webu se statistikami přístupu
source_email	retrobi@ucl.cas.cz	výchozí e-mailová adresa, která bude odesílatelem systémových e-mailů

Tabulka 2.2: Seznam nastavení v souboru web.xml

Název	Popis	Hodnoty	Výchozí hodnota
key	systémový název	řetězec z abecedy, podtržítek a čísel	povinné!
title	lidový název	řetězec	povinné!
repeat	opakovatelnost atributu	true, false	false
role	minimální uživatelská role	GUEST, USER, EDITOR, ADMIN	GUEST
value	výchozí hodnota	řetězec	prázdné
children	podatributy	pole [atribut, atribut, . . .]	prázdné
index	indexy	pole [řetězec, řetězec, . . .]	prázdné

Tabulka 2.3: Tabulka atributů pro zápis položkového rozpisu

Definice položkového rozpisu a indexů

Název a přesné umístění souboru s definicí položkového rozpisu a indexů je určeno v konfiguračním souboru webové aplikace **web.xml** (viz výše). Pokud soubor s definicemi na zadaném umístění není, je při spuštění webové aplikace automaticky vytvořen soubor výchozí.

Položkový rozpis slouží k podrobnějšímu rozepsání jednotlivých údajů do uživatelsky definované stromové struktury polí.

Struktura položkového rozpisu včetně zařazení jednotlivých položek do indexů je při startu webové aplikace načtena z textového souboru, obsahujícího její popis ve strukturovaném textovém formátu JSON (viz níže). Tyto definice lze měnit v libovolném textovém editoru, přičemž je nutné zachovávat předdefinovanou strukturu souboru. Po uložení je nutné buď počkat do další automatické údržby systému, spustit ruční obnovu v administrační sekci „Nástroje“, nebo webovou aplikaci restartovat.

Podrobný popis jednotlivých atributů se nachází v tabulce 2.3.

Řetězce musí být z obou stran uzavřeny v uvozovkách (”) a tyto uvozovky se v samotném řetězci objevit nesmí. Doporučujeme sledovat stávající obsah souboru a způsob zápisu „odkoukat“.

Zde je ukázka definice jednoho atributu.

```
{
  "key":"zahlavi",
  "repeat":false,
```

Název	Popis	Hodnoty	Výchozí hodnota
key	systémový název	řetězec z abecedy, podtržítek a čísel	povinné!

Tabulka 2.4: Tabulka atributů pro zápis informací o indexech

```
  "role":"GUEST",
  "title":"Záhlaví",
  "value":"",
  "children": [
    {...},
    {...},
    {...}
  ]
}
```

Definice dodatečných vlastností indexů

Název a přesné umístění souboru s definicí dodatečných vlastností indexů je určeno v konfiguračním souboru webové aplikace **web.xml** (viz výše). Pokud soubor s definicemi na zadaném umístění není, je při spuštění webové aplikace automaticky vytvořen soubor výchozí.

Hlavním účelem dodatečných definic vlastností indexů je pojmenovat všechny indexy lidsky čitelným názvem namísto krátkých systémových názvů (např. **Osoby autorské** místo systémového **person_author**.

Definice indexů má podobná specifika jako definice atributů, jen struktura definic a umístění souboru je jiné. Ve struktuře je možné definovat „lidový“ název indexu, jeho pořadí v rozbalovací nabídce a minimální roli, která je k použití indexu nutná. Přesné popisy atributů jsou uvedeny v tabulce. Pokud nějaký index záznam v tomto souboru nemá, nevadí – jsou použity výchozí hodnoty.

```
[
  {
    "name":"person_author",
    "order":2,
    "role":"GUEST",
```

```
    "title":"Osoby autorské"
  }, {
    ...
  }
]
```

2.5 Oživení

2.5.1 Kompilace

Systém Retrobi

Oživení systému začíná stažením aktuálního zdrojového kódu. Zdrojový kód je umístěn na verzovacím systému SVN, který běží na serveru společnosti inSophy. K tomuto zdroji má ÚČL přístup pro čtení (uživatel **retrobi**). Do konzole napište tyto příkazy (pro jistotu je spouštějte jako obyčejný uživatel, nikoliv jako root):

```
svn export https://retrobi@insophy.cz/svn/retrobi/trunk retrobisrc
cd retrobisrc
mvn
mv retrobi-web/target/retrobi-web-1.0.0-SNAPSHOT.war ../retrobi.war
cd ..
rm -r retrobisrc
```

Nyní popíšeme, co uvedené příkazy vykonají. Nejprve dojde k přihlášení do systému SVN a ke stažení zdrojového kódu do nového adresáře **retrobisrc**. Dalším krokem je kompilace, tedy vytvoření binárního balíčku ze zdrojových kódů. Tu plně automaticky provádí systém Maven. Ten vypisuje podrobný protokol o své činnosti, ze kterého je možné vidět, jak proces kompilace probíhá. Po úspěšném dokončení kompilace vznikne v adresáři **retrobi-web/target** soubor **retrobi-web-1.0.0-SNAPSHOT.war**, který se pro jednoduchost přesune o adresář výše do souboru **retrobi.war** a později se umístí na server Tomcat. Na závěr už se jen smaže dočasný adresář se zdrojovými soubory **retrobisrc** a z celého procesu zbyde už jen výsledný balíček **retrobi.war**.

Vyhledávací servlet CouchDB-Lucene

Spolu se zdrojovým kódem systému Retrobi se stáhne i zdrojový kód vyhledávacího servletu CouchDB-Lucene, což je servlet spravující indexy a provádějící vyhledávání. Tento servlet běží paralelně s databází CouchDB, která mu posílá všechny provedené změny dokumentů. CouchDB-Lucene tyto změny zpracovává a udržuje vyhledávací indexy podle definic, které si načte také. Pokud se definice vyhledávacího indexu změní, musí se index přepočítat, což může trvat až několik hodin. Pokud je s indexy nějaký vážný problém (např. chyba v souboru), je možné je zcela smazat (tzn. smazat veškerý obsah adresáře CESTA_INDEX) a restartovat CouchDB-Lucene. Indexy budou znovu vytvořeny v řádu několika hodin.

Jedná se o projekt s otevřeným kódem, do kterého byly námi zaneseny drobné úpravy pro účely použití v systému Retrobi. Tyto úpravy jsou následující:

- Podpora booleanovského parametru **qsplit**, který určuje, zda se bude dotaz rozdělovat po čárkách. Toto chování je v systému Retrobi nežádoucí, protože čárky mohou být součástí dotazu. Výchozí hodnota parametru je **true** (z důvodu kompatibility), ale systém Retrobi používá hodnotu **false**.
- Podpora booleanovského parametru **allow_leading_wildcard**, který umožňuje dotaz začínající hvězdičkou. Výchozí hodnota je **false** (z důvodu kompatibility), ale systém Retrobi používá hodnotu **true**.
- Podpora booleanovského parametru **lowercase_expanded_terms**, který automaticky převádí velká písmena na malá při prefixovém dotazu (s hvězdičkou na začátku). Výchozí hodnota je **true**, systém Retrobi tuto hodnotu mění dle nastavení vyhledávacích parametrů.
- Podpora řetězcového parametru **include_fields**, který určuje, které uložené hodnoty budou vráceny spolu s výsledky. Systém Retrobi vyžaduje pouze **_id**. Tato nová funkce je již součástí oficiální větve CouchDB-Lucene.
- Přidána podpora **WhitespaceAnalyzer**, který systém Retrobi používá.

Zdrojové kódy vyhledávacího servletu CouchDB-Lucene se nachází v podadresáři **couchdb-lucene**. Kompilaci opět zajistí systém Maven.

```
cd retrobisrc
cd couchdb-lucene
```

```
mvn
cd target
unzip couchdb-lucene-0.9.0-SNAPSHOT-dist.zip
mv couchdb-lucene-0.9.0-SNAPSHOT CESTA_LUCENE
```

Nyní popíšeme, co uvedené příkazy vykonají. Po vstupu do adresáře **couchdb-lucene** se spustí kompilace příkazem **mvn**. Po dokončení vznikne složka **target**, ve které se nachází soubor ZIP, který se rozbalí a celý jeho obsah zkopíruje na požadované umístění.

Dále je třeba nastavit servlet CouchDB-Lucene jako systémovou službu, aby se zjednodušilo jeho spouštění, zastavování a restartování. Nejprve tedy vstoupíme do adresáře se zdrojovými kódy CouchDB-Lucene a vyhledáme skript na tomto umístění:

```
src/main/tools/etc/init.d/couchdb-lucene
```

V tomto souboru je nutné změnit cestu ke skriptu, který spouští CouchDB-Lucene, aby byla platná i na vašem systému (viz `CESTA_LUCENE`). Jedná se zhruba o dvacátý řádek, který vypadá takto:

```
DAEMON=/opt/couchdb-lucene/bin/run
```

V něm změňte hodnotu parametru **DAEMON** na platnou cestu ke skriptu **run**, který se nachází v této složce:

```
CESTA_LUCENE/bin/run
```

Po úspěšné úpravě tohoto skriptu je nutné jej nakopírovat do složky `/etc/init.d`, aby jej šlo použít jako systémovou službu. K tomu potřebujete rootovská oprávnění.

```
cd retrobisrc
cd couchdb-lucene
cd src/main/tools/etc/init.d
sudo cp couchdb-lucene /etc/init.d/
```

Služba se pak spustí tímto způsobem:

```
sudo service couchdb-lucene start
```

Dalším možným parametrem příkazu je *stop* (zastavení) a *restart* (restart). Služba by měla běžet vždy, když běží webová aplikace systému Retrobi, jinak nebude vyhledávání fungovat.

Občas se vyskytuje podivné chování, kdy skript začne vypisovat svůj standarní výstup přímo do konzole, ze které byla spuštěna systémová služba. Konzole ale dál reaguje na příkazy a tento výstup lze ignorovat. Příčina je pravděpodobně v logovacím kódu CouchDB-Lucene, který není schopen svůj výstup v případě systémové služby správně přesměrovat.

Spuštění webové aplikace (deploy)

Nejprve se ujistíme, že jsou nainstalovány všechny potřebné balíčky a server Tomcat běží. Po zadání adresy **http://localhost:8090/** by se měla ukázat stránka s textem „It works!“. Ovládací konzole se nachází na této adrese:

```
http://localhost:8080/manager/html
```

Po přihlášení se otevře administrátorská ovládací konzole (viz obrázek 2.1), ve které je seznam spuštěných aplikací. Pokud webová aplikace Retrobi již běží, je nutné ji nejprve zastavit tlačítkem **Undeploy**. Spuštění webové aplikace systému Retrobi poté zajistíme odesláním zkompilovaného balíčku **retrobi.war** (viz předchozí krok) na server pomocí formuláře, který se nachází pod seznamem aplikací. Tomcat automaticky zajistí všechny operace nutné ke spuštění aplikace a poté již musíte jen vyčkat, než webová aplikace systému Retrobi nastartuje (může trvat od jednotek minut až po jednotky hodin v případě potřeby aktualizace všech indexů v databázi). Všechny parametry (URL aplikace, apod.) se konfigurují v konzoli serveru Tomcat.

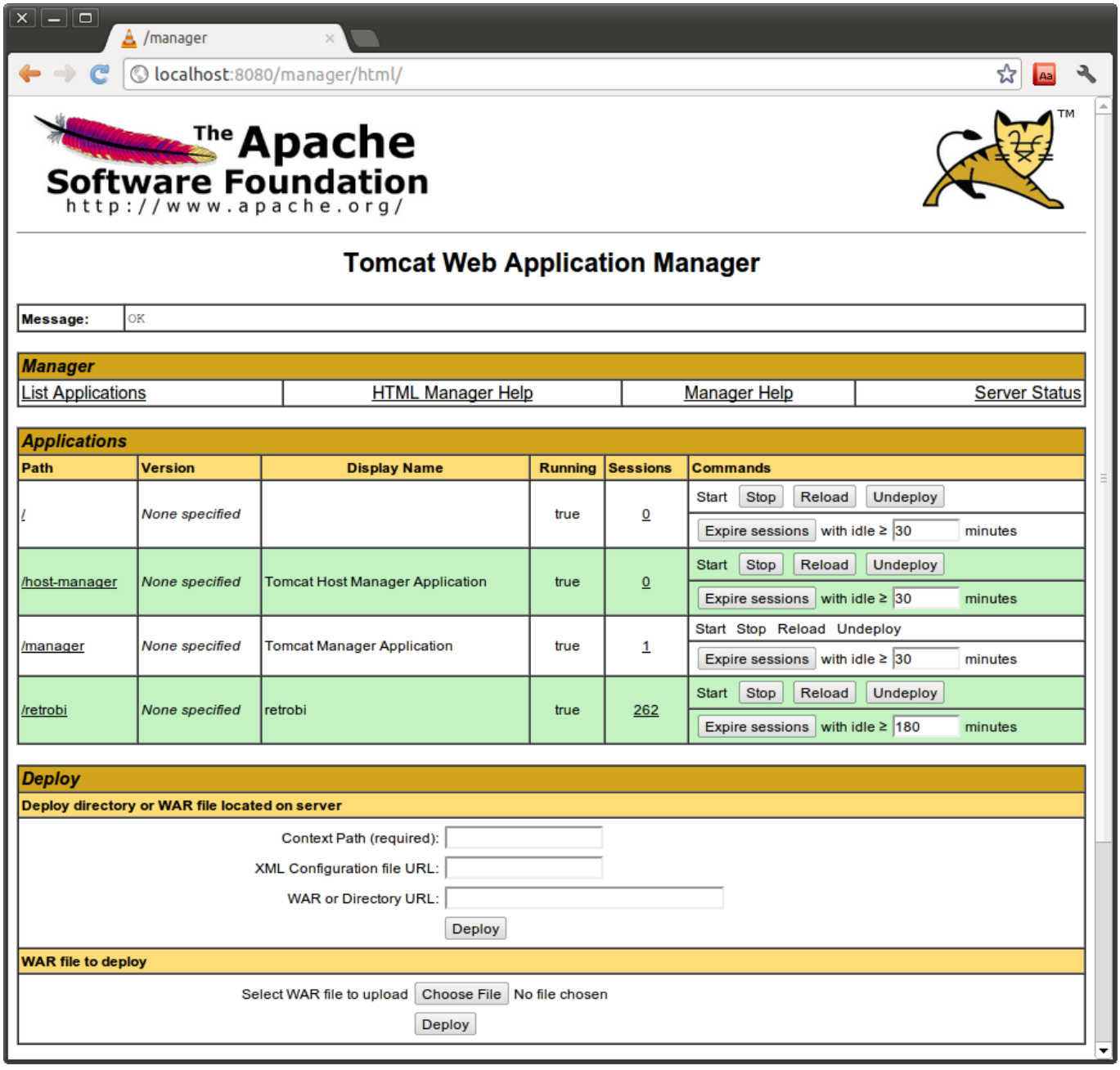
2.5.2 Protokol aplikace

Protokol webové aplikace systému Retrobi je možné sledovat následujícím příkazem (stisknutím **Shift+F** se dostanete na konec souboru a prohlížeč bude automaticky skrolovat na nové záznamy):

```
less /var/log/tomcat7/catalina.out
```

Více podrobností o možnostech instalace a spouštění aplikací v serveru Tomcat lze najít na této adrese:

```
http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html
```



Obrázek 2.1: ovládací konzole serveru Tomcat

Kapitola 3

Vývojářská dokumentace

3.1 Technologie

Celý systém je naprogramován v rozšířeném objektově orientovaném jazyce *Java* (verze 1.6) a drobné části (např. databázové pohledy) ve skriptovacím jazyce *JavaScript*. Systém používá dokumentově orientovanou databázi *CouchDB*, přístupnou přes rozhraní HTTP (REST API). Spolupráce s databází je zajištěna open-source knihovnou *jcouchdb*. Webová aplikace je postavena na frameworku *Apache Wicket*. Produkční verze webové aplikace běží v kontejneru *Tomcat*. Pro fulltextové vyhledávání je použit servlet *CouchDB Lucene*, který v sobě integruje vyhledávací engine *Lucene* a stará se o jeho napojení na databázi CouchDB. Pro čtení obrázků typu TIFF používáme knihovnu *Apache Sanselan*, o kompilaci a sestavení se stará *Apache Maven*. Zdrojový kód byl vyvíjen v prostředí *Eclipse*, nastavení automatického formátování je přiloženo v souboru *eclipse_formatter.xml*.

Zde jsou domovské odkazy na každý zmíněný produkt (platné k datu 1. dubna 2012):

- **http://download.oracle.com/javase/**
- **http://www.tvorba-webu.cz/javascript/**
- **http://code.google.com/p/jcouchdb/**
- **https://github.com/rnewson/couchdb-lucene**
- **http://commons.apache.org/sanselan/**
- **http://couchdb.apache.org/**

- **http://wicket.apache.org/**
- **http://maven.apache.org/**
- **http://www.eclipse.org/**

Během programování bylo dbáno na to, aby byl zdrojový kód přehledný, čitelný, pochopitelný a dobře okomentovaný. Proto v tomto dokumentu nebudeme zacházet do velkých implementačních detailů, protože ty by měly být zřejmé při studiu konkrétní části zdrojového kódu. Implementace zde bude popsána na úrovni konceptů a jednotlivých tříd.

Poznámka: Podrobnější dokumentaci k jednotlivým třídám, metodám a proměnným je možné najít v samostatném dokumentu, který je generován přímo z komentářů ve zdrojovém kódu (JavaDoc) a je přiložen jako součást dokumentace.

3.1.1 Formát JSON

Klíčovým formátem dat používaným na různých místech systému Retrobi je strukturovaný textový formát JSON (Javascript object notation). Jak již název napovídá, jedná se o velmi omezenou podmnožinu jazyka Javascript určenou k zápisu strukturovaných dat.

Jazyk JSON je velmi jednoduchý, jeho syntaxe je postavena na dvou základních strukturách: *množina párů klíč–hodnota* a *seznamu hodnot*. Zde je vypsána jeho gramatika:

- objekt ::= {} | { páry }
- páry ::= pár | pár , páry
- pár ::= řetězec : hodnota
- pole ::= [ ] | [prvky]
- prvky ::= hodnota | hodnota , prvky
- hodnota ::= řetězec | číslo | objekt | pole | true | false | null

A zde je příklad krátkého dokumentu:

```
{
  "jmeno" : "Karel Čapek",
  "narozen" : {"den" : 9, "mesic" : 1, "rok" : 1890},
  "zamestnani" : ["spisovatel", "novinář", "dramatik"]
}
```

3.1.2 CouchDB

CouchDB je dokumentově orientovaná databáze přístupná přes HTTP REST API implementovaná ve funkcionalním programovacím jazyce Erlang.

Narozdíl od relačních databází nejsou v CouchDB žádné relace, tabulky, transakce, integritní omezení, ani datové definice. Relace mezi záznamy jsou v aplikaci Retrobi zajištěny až na úrovni aplikace – vazební proměnné jsou tedy řetězce typu **String** (případně **Set**<**String**>) a je v nich uložen klíč vázaného dokumentu (řetězec obsahující **_id** nebo prázdná hodnota **null**).

Integrita dat tedy není nijak kontrolována na úrovni databáze. To klade vyšší nároky na kvalitu klientské aplikace, která ale není omezena v tom, co může do databáze uložit. Vedle sebe tak mohou ležet komentáře k lístkům, lístky, uživatelé, schránky a další typy dokumentů. Systém Retrobi nemá datový model tak košatý, aby to nějak zásadně vadilo a převažují výhody zvoleného řešení, kterými jsou jednoduchost, rychlost a schopnost skladovat strukturovaná data s přiloženými obrázky, což přesně lístková kartotéka v současném stavu je.

Dokumenty jsou v databázi CouchDB logicky (nikoliv fyzicky) uloženy ve formátu JSON (Javascript Object Notation), který byl popsán výše. Adresní prostor databáze je plochý, každý záznam má právě jeden unikátní klíč – řetězec 32 alfanumerických znaků. Dokument v databázi CouchDB má tři speciální proměnné (poznají se tak, že začínají podtržítkem):

_id: povinné; primární klíč dokumentu (automaticky generovaný nebo ručně zadaný);

_rev: povinné; revize (slouží pro kontrolu verzí a prevenci update konfliktů);

_attachments: nepovinné; každý dokument u sebe může mít přiložen libovolný počet pojmenovaných souborů ve formě přílohy (například u lístků jejich naskenované obrázky); tyto přílohy se nenačítají přímo s dokumentem, ale na vyžádání (kvůli jejich velikosti).

Dokument je z databáze možné načíst buď dle primárního klíče (tedy řetězce **_id**), nebo využít tzv. *pohledy*. Pohledy nad databází CouchDB jsou postaveny na principu *map–reduce* a každý pohled je zároveň indexem. Každý databázový pohled se skládá ze dvou funkcí: funkce *map* a funkce *reduce*, které jsou definovány v jazyce JavaScript.

- **Mapovací funkce (*map*)** je spuštěna pro každý jeden dokument a vyhodnocena; v jejím těle lze zavolat speciální funkci **emit(klíč, hodnota)**, kterou lze na základě dokumentu vytvořit libovolný počet párů *klíč–hodnota*.

- **Redukční funkce (*reduce*)** je nepovinná funkce, která je spuštěna (pokud je přítomna) pro každý unikátní klíč a množinu hodnot k tomuto klíči přiřazenou; jako vstup přijímá dvojice *klíč–hodnota* vytvořené v mapovací funkci *map* a jako výstup vrací jednu hodnotu (skalár/objekt/pole), která je poté ke klíči přiřazena místo všech hodnot původních.

Pohledy jsou uloženy v tzv. *design dokumentech*, což jsou speciální systémové dokumenty, jejichž klíč začíná řetězcem **_design/** a respektují strukturu, ve které je dotazovací podsystém CouchDB schopen definice pohledů najít. Vstupním bodem do této definice je hodnota atributu **views**, kde jsou v další podstruktuře uvedeny definice jednotlivých pohledů (těla jejich funkcí *map* a *reduce*). Tato podstruktura vypadá zhruba takto:

```
"views": {
  "VIEW NAME 1": {
    "map": "MAP FUNCTION",
    "reduce": "REDUCE FUNCTION"
  }
}
```

Dotaz na databázi je v podstatě jen otevření platné URL adresy (požadavek typu GET), která obsahuje tyto údaje:

- název *design dokumentu* - klíč dokumentu obsahujícího definici požadovaného pohledu;
- název pohledu;
- klíč (nepovinné), který omezí výsledky na zadaný klíč;
- počáteční klíč (nepovinné), který omezí výsledky na ty, jejichž klíče jsou „větší nebo rovny“ zadanému počátečnímu klíči (princip řazení je definován v dokumentaci CouchDB);
- konečný klíč (nepovinné), který omezí výsledky na ty, jejichž klíče jsou „menší“ než zadaný konečný klíč (lze kombinovat s počátečním klíčem);
- maximální počet výsledků;
- počet výsledků, které se mají od začátku přeskočit.

Po otevření této adresy se CouchDB nejprve podívá, zda je požadovaný pohled aktuální (tedy jsou v něm obsaženy všechny změny dokumentů od jeho poslední aktualizace a nezměnila se jeho definice).

Pokud pohled aktuální není, je nutné jej přepočítat (toto přepočítání je možné vypnout pomocí URL parametrem **stale=ok**). Přepočet nějakou chvíli trvá, obzvláště na databázích s velkým počtem dokumentů a pohledů. Pokud se změnila definice pohledu (funkce *map* nebo *reduce*), je nutné jej přepočítat pro všechny dokumenty v databázi. Pokud se ale změnil jen obsah databáze (tedy dokumenty), přepočítá se jen část pohledu obsahující změněné dokumenty. I proto jsou funkce *map* a *reduce* stavěny tak, jak jsou – aby je bylo možné nezávisle spustit pro každý dokument zvlášť.

V případě, že je pohled aktuální či byl právě úspěšně dokončen jeho přepočet, databáze jako odpověď zobrazí výsledek dotazu ve formátu JSON. Tento výsledek obsahuje jak samotná data, tak i několik informací o pohledu samotném.

Příklad pohledu 1 (bez redukce)

Mapovací funkce:

```
function(doc) {
  if (doc.catalog) {
    emit([doc.catalog, doc.batch, doc.nr], doc._id);
  }
}
```

Dotaz a výsledek:

```
http://127.0.0.1:5984/retrobi/_design/views/_view/cards?limit=5
```


```
{"total_rows":36,"offset":0,"rows":[
{"id":"1781ccfb6570416e646db60b1d0024ab",
  "key":["A","BATCH1",1],
  "value":"1781ccfb6570416e646db60b1d0024ab"},
{"id":"1781ccfb6570416e646db60b1d00314b",
  "key":["A","BATCH1",2],
  "value":"1781ccfb6570416e646db60b1d00314b"},
```

```
{"id":"1781ccfb6570416e646db60b1d003180",
  "key":["A","BATCH1",3],
  "value":"1781ccfb6570416e646db60b1d003180"},
{"id":"1781ccfb6570416e646db60b1d003e7c",
  "key":["A","BATCH1",4],
  "value":"1781ccfb6570416e646db60b1d003e7c"},
{"id":"1781ccfb6570416e646db60b1d004b49",
  "key":["A","BATCH1",5],
  "value":"1781ccfb6570416e646db60b1d004b49"}
]}
```

Příklad pohledu 2 (s redukcí)

Mapovací funkce:

```
function(doc) {
  if (doc.TAG_card) {
    emit('total card count', 1);
  }
}
```

Redukční funkce:

```
function(key, values) {
  return sum(values);
}
```

Dotaz a výsledek:

```
http://127.0.0.1:5984/retrobi/_design/views/_view/count_cards?group=true
```

```
{"rows":[
  {"key":"card count","value":36}
]}
```

3.2 Součásti projektu

Projekt je rozdělen do tří modulů (podprojektů). Každý tento modul bude popsán zvlášť.

- společná knihovna tříd (retrobi-common);
- jednoúčelové nástroje (retrobi-tools);
- webová aplikace (retrobi-web).

3.2.1 Společná knihovna tříd

Společná knihovna tříd se nachází v podprojektu *retrobi-common*. Všechny ostatní části systému Retrobi ji využívají. Obsahuje datový model, třídy pro práci s ním a různé pomocné knihovny funkcí. Tyto součásti budou dále popsány.

Databázový podsystém

Datovým modelem zde rozumíme modelovanou realitu, entitou jeden prvek datového modelu, tedy ohraničenou část modelované reality.

Základní entitou v databázi CouchDB je *dokument*, který má svůj jednoznačný identifikátor (ID) a číslo revize (REV). Všechny entity určené k uchování v databázi musí být převedeny na dokumenty. Pokud je chceme načíst a opět s nimi pracovat, musíme je z dokumentů převést zpět na instance určitých objektů. Databázové dokumenty jsou interně uchovávány ve formátu JSON.

Tyto základní operace zajišťuje knihovna *jcouchdb*, která pomocí HTTP spojení komunikuje s databázovým serverem CouchDB a automaticky (pomocí introspekce) převádí instance objektů na dokumenty a zpět. Aplikace postavená na knihovně *jcouchdb* tedy může definovat doménový model jako klasické třídy s doplňujícími anotacemi.

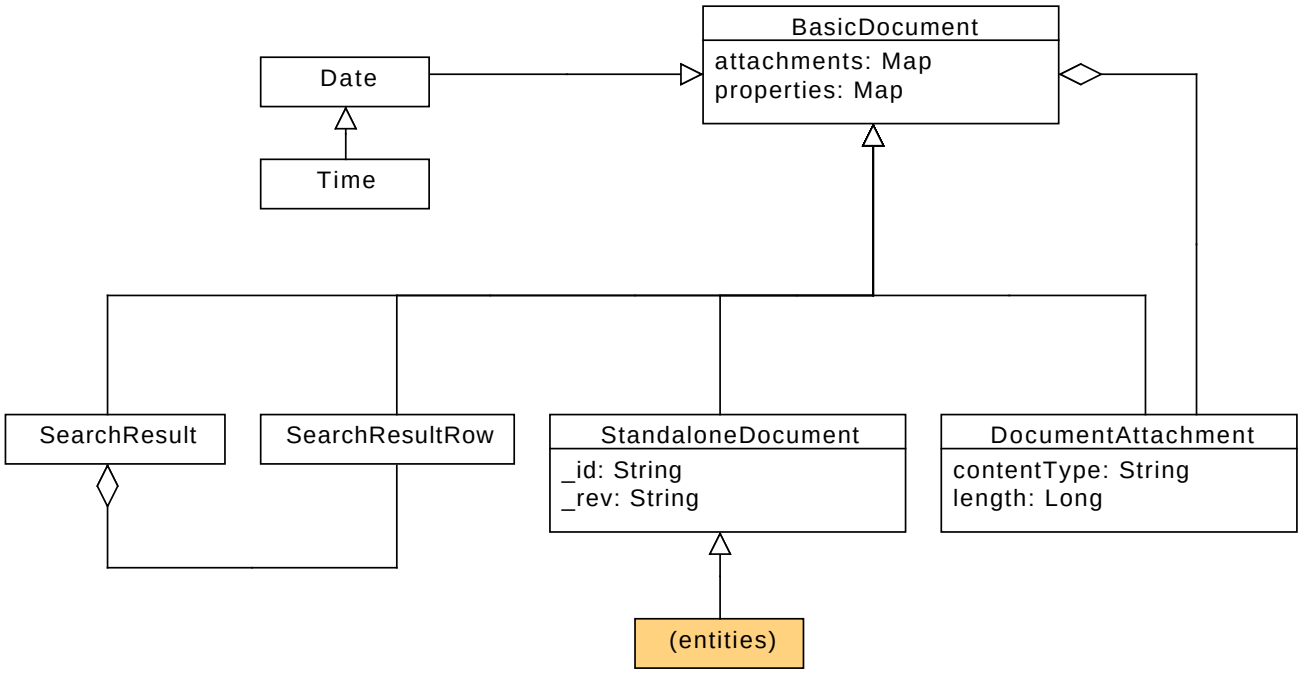
Nejdůležitějšími metodami třídy **Database** knihovny *jcouchdb* jsou tyto:

- **createDocument()** - vytváření dokumentů,
- **updateDocument()** - úprava dokumentů,

- **getDocument()** - načtení dokumentu dle ID,
- **delete()** - mazání dokumentů,
- **createAttachment()** - přiložení přílohy,
- **updateAttachment()** - úprava přílohy,
- **getAttachment()** - načtení přílohy,
- **deleteAttachment()** - odstranění přílohy,
- **queryView()** - spuštění dotazu na pohled.

Načítání konkrétních dat z databáze je implementováno v jednotlivých repositářích různě, ale všechny tyto metody mají podobnou kostru: 1) vytvoření hraničních klíčů; 2) příprava objektu s parametry pro pohled – především počáteční a koncový klíč, omezení na počet výsledků a číslo prvního řádku; 3) provedení vlastního dotazu na pohled; 4) vytvoření výsledné kolekce a sběr převedených výsledků. Typickou kostru takovéto metody pro ilustraci uvádíme níže. Co se týče načítání vlastních dat, většinou se v prvním průchodu pro jednoduchost načítají pouze klíče (ID) objektů a vlastní objekty jsou získány až později v případě potřeby. Z tohoto důvodu také mnoho metod přijímá jako parametry právě tyto klíče a jejich seznamy.

```
final Object startKey = START KEY;
final Object endKey = END KEY;

final Options options = new Options().
  startKey(startKey).endKey(endKey).skip(SKIP).limit(LIMIT);

final ViewResult<String> result = this.queryView(
  VIEW NAME, String.class, options);

final List<DOCUMENT> list = new LinkedList<DOCUMENT>();

for (final ValueRow<String> row: result.getRows()) {
  list.add(getDocument(CLASS, row.getValue()));
}

return Collections.unmodifiableList(list);
```



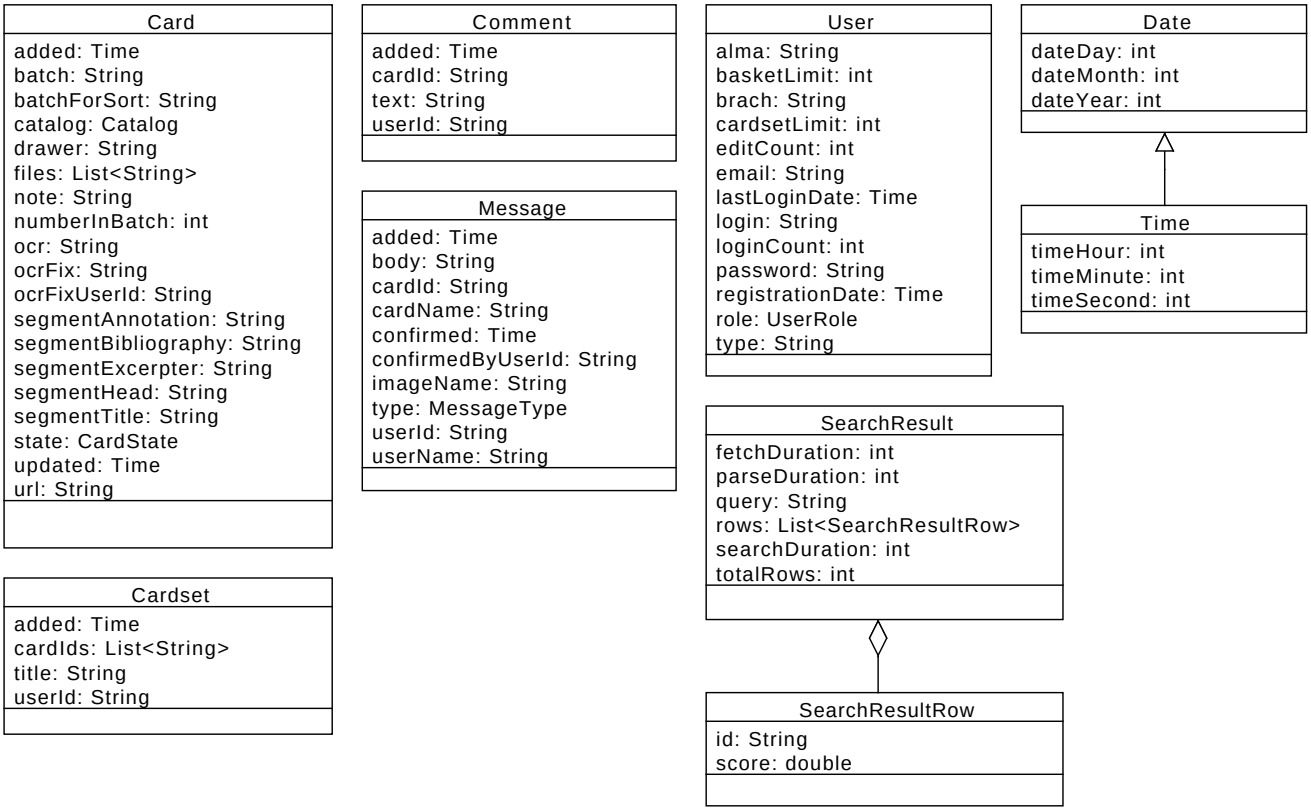
Obrázek 3.1: hierarchie dokumentů a entit (UML)

Základy datového modelu

Jak již bylo řečeno, základní a obecnou entitou je dokument. Dokument je předkem všech entit a každá entita je tedy zároveň i specifickým druhem dokumentu. Proto v datovém modelu existuje silná hierarchie, jejímž vrcholem je třída **BasicDocument**. Ta je nejobecnější reprezentací dokumentu a všechny entity obsažené v databázi je na ní možné pomocí knihovny *jcouchdb* převést. Taková entita ale není ve své obecnosti příliš užitečná. Dalším stupněm je třída **StandaloneDocument**, která přidává vlastnosti **_id** (identifikátor) a **_rev** (revize). Takto již máme jednoznačně určený dokument a jeho revizi. Od této třídy dědí všechny entity (**entities**), které přidávají další konkrétní vlastnosti.

Existují však i nesamostatné entity (často součásti jiných entit), které nevyžadují celou složitost třídy **StandaloneDocument** a jako předek jim stačí jednodušší **BasicDocument**. S nimi může knihovna *jcouchdb* pracovat, ale neobsahují jednoznačné ID a číslo revize. Těmito nesamostatnými entitami je příloha dokumentu **DocumentAttachment**, datum **Date**, čas **Time**, výsledek vyhledávání **SearchResult** a jeho řádek **SearchResultRow**.

Celá tato hierarchie dokumentů je vyznačena na diagramu 3.1.



Obrázek 3.2: seznam entit (UML)

Datový model

Datový model představuje modelovanou realitu, v tomto případě katalog lístků a podpůrné entity v databázi (uživatele, schránky, komentáře, hlášení, atd.). Jednotlivé složky datového modelu na sobě nejsou přímo závislé, vazby jsou realizovány přes dokumentové ID a jsou tedy typu **String**. Pokud v takové vazbě existuje hodnota **null**, vazba neexistuje. Databáze CouchDB sama žádnou kontrolu integritních omezení neobsahuje, vše obsluhuje datový podsystém aplikace.

Datový model systému Retrobi je uveden na diagramu 3.2.

Následuje seznam tříd a krátký popis reality, kterou modelují.

Card: lístek (souhrn papírových kartiček a informací na nich obsažených);

Cardset: uložená uživatelská schránka;

Comment: uživatelský komentář k lístku;

Message: hlášení systémové (událost) či uživatelské;

User: registrovaný uživatel;

SearchResult: výsledek vyhledávání;

SearchResultRow: řádek výsledku vyhledávání;

Date: datum (den, měsíc, rok);

Time: datum a čas (den, měsíc, rok, hodina, minuta, sekunda).

Stav lístku

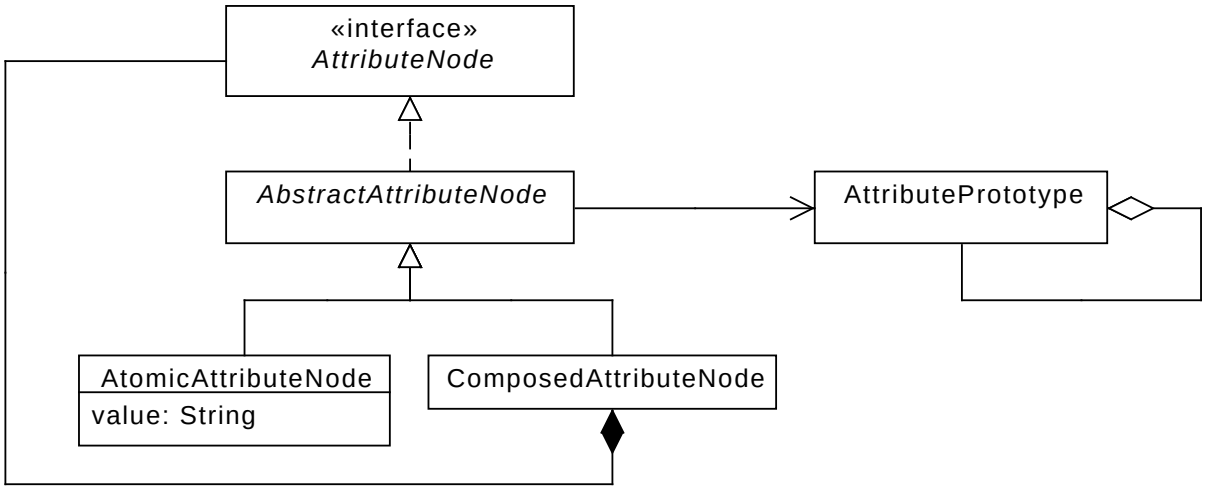
Důležitou vlastností lístku je jeho stav, který vyjadřuje kvalitu a míru přepisu. Prvním a tedy nejnižším stavem je stav „nový“, posledním a tedy nejvyšším stavem je stav „uzavřený“. Každý lístek je v jednom z následujících stavů:

1. nový (lístek byl nově uložen do databáze),
2. přepsaný (lístek byl editován uživatelem a přepis zatím neprošel redakcí),
3. segmentovaný (lístek prošel redakcí a je pro další úpravy uzamčen),
4. strukturovaný (lístek obsahuje segmentaci a položkový rozpis),
5. uzavřený (lístek je kompletně ověřený, přepsaný a uzavřený).

Obrázky lístku

Obrázky lístku (včetně syntetických) jsou uloženy jako přílohy jednotlivých dokumentů reprezentujících lístky v databázi CouchDB. S těmito přílohami pracuje výhradně repositář **CardImageRepository**, který se stará o jejich načítání, ukládání a mazání. Přílohy dokumentu v databázi CouchDB se nenačítají automaticky s dokumentem samotným kvůli úspoře místa. Proto je načítání obrázků nějakého lístku dodatečná operace.

Každá příloha je jednoznačně daná klíčem (ID) svého rodičovského dokumentu a svým názvem. Název přílohy musí být unikátní v rámci jednoho dokumentu, jinak dochází k nechtěnému přepsání.



Obrázek 3.3: atributy lístku (UML)

Uživatelské atributy lístku

Uživatelské atributy lístku je množina hierarchicky uspořádaných hodnot, jejichž struktura se definuje ve zvláštním konfiguračním souboru (viz výše). Tato struktura je po startu systému naparsována a na jejím základě je vytvořen tzv. *strom prototypů* (objektů třídy **AttributePrototype**). Každý prototyp se svým okolím je pak předlohou pro jeden či více skutečných atributů (implementujících rozhraní **AttributeNode**). Atributy jsou uspořádány ve stromové struktuře, ve které je každý uzel teoreticky vícenásobný atribut. Atributy jsou buď kompozitní (**ComposedAttributeNode**) s podatributy, nebo atomické (**AtomicAttributeNode**). Atomické atributy (listy stromu) jsou pak nositeli konkrétní řetězcové hodnoty. Strom atributů je přiřazen k lístku.

U každého konkrétního atributu na lístku je možné jednoznačně určit, zda jej lze smazat či klonovat.

Hierarchie atributů a jejich prototypů je uvedena na diagramu 3.3.

Správa entit v databázi

Databázový podsystém kromě entit obsahuje i třídy, které se o ně starají, tzv. *repositáře*. Repositáře obsahují i definice (kódy Javascript + názvy) souvisejících *databázových pohledů*, které slouží jako různé reprezentace dat uložených v databázi CouchDB. Každý repositář spravuje určitou podmnožinu datového modelu.

Základním kamenem repositářů je třída **AbstractProtoRepository**, která obsahuje napojení na knihovnu *jcouchdb* a implementuje s její pomocí základní operace CRUD (create, read, update, delete – vytvořit, načíst, upravit, smazat). Dalším patrem je třída **AbstractRepository**, která od ní dědí. Přidává k ní další a pokročilejší funkce pro dotazování databáze, jako je např. načítání několika entit najednou. Zjednodušuje zkrátka dalším patrům načítání dat a tak zkracuje jejich kód.

Dalšími patry jsou pak již jednotlivé repositáře:

AnalystRepository: analýza lístků v databázi, načítání statistických údajů, rejstřík hodnot;

CardImageRepository: správa obrázků u lístků;

CardRepository: správa lístků a zjišťování struktury katalogu, přečíslování skupin;

CardSearchRepository: spolupráce s vyhledávacím servletem *couchdb-lucene* a správa indexů;

CardsetRepository: správa schránek v databázi;

CommentRepository: správa uživatelských komentářů;

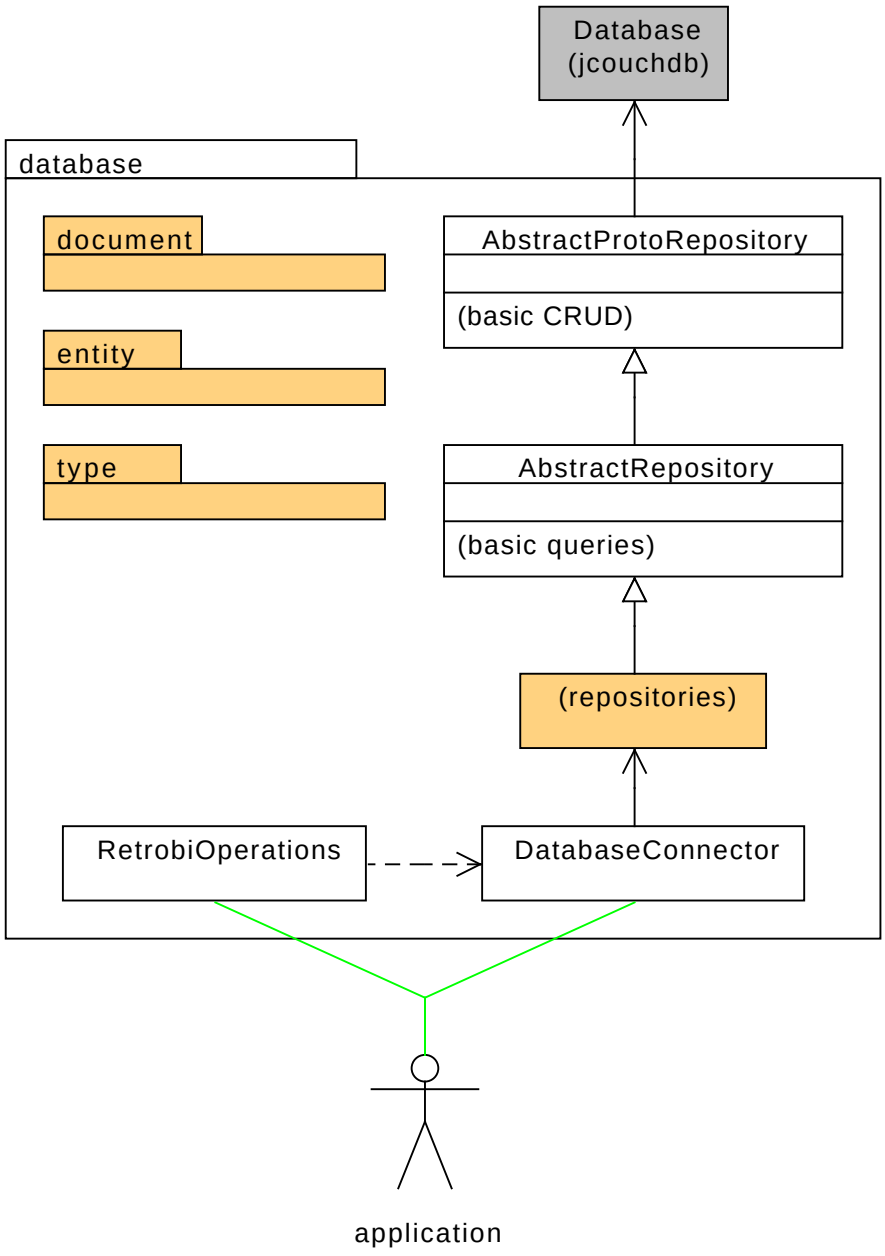
MessageRepository: správa uživatelských a systémových hlášení, zápis událostí do CSV logu, záloha a mazání starých hlášení, potvrzování;

TextRepository: správa textů na webu;

UserRepository: správa uživatelů, ověřování přihlašovacích údajů, změna hesla, kontrola duplicitních údajů, žebříček přepisovatelů.

Některé operace s entitami v systému Retrobi jsou poměrně komplikované a vyžadují zapojení několika různých repositářů. Takové operace se nachází ve speciální třídě **RetrobiOperations**, která je přístupná ze všech částí aplikace, podobně jako **DatabaseConnector**. Tyto operace jsou:

Registrace uživatele: Před registrací se provede kontrola, zda stejný login či e-mail v databázi již neexistuje. Poté proběhne registrace a nakonec se novému uživateli odešle e-mail s přihlašovacími údaji. Událost se zaloguje.



Obrázek 3.4: správa entit v databázi (UML)

Odstranění uživatele: Odstraní schránky a komentáře uživatele, poté smaže uživatele, událost se zaloguje a čerstvě odstraněnému uživateli se odešle informační e-mail. Pozor, zde neprobíhá reset uživatelských přepisů, tato funkcionalita je oddělená.

Změna segmentace: Nejdříve se zkontroluje, zda již lístek není uzavřený. Poté se spustí segmentační algoritmus a identifikují se jednotlivé segmenty. Pokud se segmentace povede (nebo je-li segmentace zadána ručně), lístek se v databázi upraví a změna se zaloguje. Poté je stav lístku převeden na „segmentováno“, což je samostatná komplexní operace popsaná níže.

Změna přepisu OCR: Nejdříve se zkontroluje, zda již lístek není uzavřený. Pokud není, k lístku se přiřadí nový přepis včetně autora a lístek je uložen. Událost je zalogována a stav lístku je změněn na „přepsáno“. Změna stavu lístku je samostatná komplexní operace popsaná níže.

Změna stavu lístku: Nejprve je změněn stav lístku a lístek je uložen. Poté se podle minulého a nového stavu rozhodne, které další operace se provedou. Je-li nový stav vyšší než minulý (tzn. lístek má blíže k uzavření), událost se zaloguje a potvrdí se všechna otevřená hlášení u daného lístku. Pokud je nový stav lístku nižší než minulý (tzn. lístek má dále k uzavření), smažou se krycí lístky. V případě, že je nový stav lístku „segmentováno“ a vyšší, vytvoří se nové krycí lístky. Hlášení se neukládá, pokud se stav nemění.

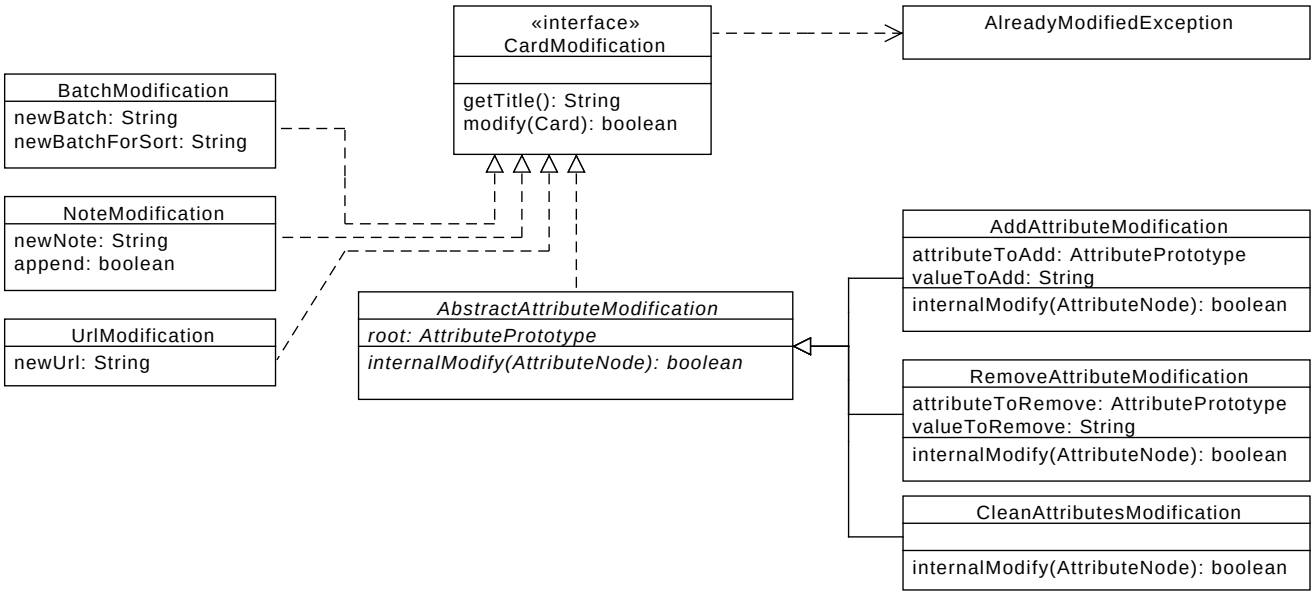
Změna údajů na lístku: Lístek je změněn a událost je zalogována. Pokud navíc dojde ke změně poznámky či URL, vytvoří se další události navíc.

Založení lístku: Vytvoří se nový výchozí lístek na zadaném místě a událost se zaloguje.

Odstranění lístku: Odstraní komentáře k danému lístku a potom i vlastní lístek. Událost se zaloguje a skupina je automaticky přečíslována.

Odstranění přepisů uživatele: Nejprve nalezne všechny lístky, které jako poslední přepsal daný uživatel. Reset přepisu se provede u podmnožiny těchto lístků, které mají stav „přepsáno“ (reset = smazání přepisu, autora přepisu, celé segmentace a převod stavu lístku na „nový“). Poté se počítadlo přepisů daného uživatele sníží o počet resetovaných lístků.

Přepnutí stavu potvrzení u hlášení: Potvrzené hlášení převede na nepotvrzené a zpět.



Obrázek 3.5: hromadné operace nad lístky (UML)

Hromadné operace

Hromadné operace jsou implementovány jako hierarchie tříd implementující rozhraní **CardModification**. Každá tato změna implementuje metodu **modify()**, která obdrží lístek určený k modifikaci, provede úpravu a vrátí logickou hodnotu oznamující, zda úprava proběhla úspěšně, či nikoliv. Fakt, že úprava neproběhla úspěšně, neznamená chybu, ale vznikne při nemožnosti provést změnu nad daným lístkem. Pokud například hromadná operace přidává atribut k lístku, neproběhne úspěšně v případě, že je atribut neopakovatelný a již nějakou hodnotu obsahuje.

Hromadné operace se dále dělí na obyčejné operace a operace s atributy. Operace s atributy jsou dvě: přidat atribut a odebrat atribut. Obě operace pracují s položkovým rozpisem lístku. Pokud tento neexistuje, je automaticky vytvořen výchozí (dle definice v souboru, viz výše). Operace dědí od abstraktního předka **AbstractAttributeModification**, který zajišťuje načtení a uložení položkového rozpisu před a po konkrétní operaci.

Všechny hromadné operace mají možnost „odmítnout“ úpravu nad konkrétním lístkem – a to v případě, že lístek zadanou modifikaci **nevyžaduje**. Pokud například nastavujeme novou skupinu lístku, který v této skupině již je, není změna nutná a bylo by zbytečné lístek ukládat a zatěžovat tak bezdůvodně databázi. Z tohoto důvodu může metoda **modify()** skončit s výjimkou **AlreadyModifiedException**, kterou nadřazená vrstva zachytí a lístek přeskočí.

Vyhledávání

Pro vyhledávání je použita univerzální platforma Lucene od nadace Apache. Tato platforma je podobně jako CouchDB založena na dokumentech, ale mezi dokumenty CouchDB a Lucene je nutný určitý netriviální převod. Hodnoty uložené v indexu se totiž mohou lišit od hodnot v databázi. Vše záleží na definici indexů – některé indexy mohou obsahovat jen přepis OCR, jiné zase skupiny, atd.

Tvorbu a obsluhu indexů zajišťuje servlet CouchDB Lucene, který je neustále spuštěný a očekává požadavky na samostatném portu.

Databáze CouchDB je nastavena tak, že hlásí všechny změny tomuto indexovacímu servletu a ten svůj index postupně aktualizuje. Platforma Lucene tedy není využita přímo, ale přes zmíněný servlet. Vyhledávání je na databázi nezávislé a vyhledávání jako takové nezatěžuje databázi, jen samotný indexovací server (za předpokladu, že jsou servery fyzicky oddělené).

Vyhledávací dotaz se (podobně jako dotazy databázové) spustí požadavkem na určitou URL adresu, na které vyhledávací servlet poslouchá. Jako parametr je v ní specifikován uživatelský dotaz, název indexu a také název pole, ve kterém se má hledání provést. Lucene podporuje širokou škálu různých vyhledávacích technik, například fuzzy dotaz, kombinovaný dotaz, dotaz na číselný rozsah, použití divokých karet a další.

Výsledky vyhledávání jsou pak předány zpět webové aplikaci, která dotaz spustila, a obsahují metainformace (čas vyhledávání, čas načítání, celkový počet výsledků bez stránkování) a řádky, ve kterých je klíčová hodnota ID obsahující ID nalezeného dokumentu a související hodnoty. V těchto hodnotách pak lze provést zvýraznění výsledků.

Společné knihovny

Společná knihovna metod je skupina několika nezávislých tříd, které obsahují různé užitečné statické metody. Každá třída realizuje určitý okruh funkcionalit a zpravidla jsou k ní napsány unit testy, které ověřují korektnost metod. Následuje seznam těchto tříd se stručným popisem jejich zodpovědnosti:

DefaultAttributePrototyper: Vytváří výchozí definici položkového rozpisu (stromu atributů) a indexů, pokud nejsou nalezeny žádné uživatelsky definované.

SimpleAttributeUtils: Pracuje s položkovým rozpisem – vyhledává, přidává a odebírá atributy, podporuje import a export struktury do souboru a získání i vložení vyplněné struktury do dokumentu.

SimpleDatabaseUtils: Zajišťuje vytvoření design dokumentu zadaných repositářů v databázi a obsahuje metody pro práci s identifikátory (**_id**) design dokumentů.

SimpleExportUtils: Obsahuje metody pro export lístků do RTF a ZIP.

SimpleFileUtils: Obsahuje metody pro práci se soubory – extrahování různých informací z jejich názvů, jejich načítání a zápis, validaci, přesun a kopírování.

SimpleFrameUtils: Obsahuje metody pro zobrazení různých dialogů: potvrzení, zpráva, chybové hlášení a výběr souboru.

SimpleGeneralUtils: Obecné metody pro práci s objekty a jejich kolekcemi. Obsahuje i poměrně důležitý algoritmus pro přesun objektu v rámci nějaké kolekce.

SimpleImageUtils: Klíčová knihovna pro práci s obrázky. Obsahuje metody pro načítání a zápis obrázků, jejich konverzi, vytváření náhledů, škrtání, otočení, detekci prázdných stran a generování syntetických obrázků.

SimpleIndexUtils: Obsahuje jen jednu veřejnou metodu pro vytvoření vyhledávacích indexů. Tato funkce je ale natolik složitá a zároveň ostře ohraničená, že byla přesunuta do vlastní třídy.

SimpleSearchUtils: Obsahuje pomocné metody pro získávání seznamu lístků z výsledku vyhledávání, pro zvýrazňování hledaných výrazů ve fragmentech a pro normalizaci vyhledávacího dotazu.

SimpleSegmentUtils: Obsahuje segmentační algoritmus a metody pro extrakci jednotlivých segmentů dle určených formátovacích pravidel z přepisu obsahujícího segmentační znaky. Také převádí lístky do textové reprezentace, která se využívá v různých částech webové aplikace.

SimpleStringUtils: Knihovna metod pro práci s řetězci, která patří mezi ty nejrozsáhlejší. Obsahuje metody pro kódování a dekódování (base64, hashe, CSV, JSON), získávání náhodného řetězce, skloňování, validaci, typografickou opravu, zalamování řádků a různé netriviální úpravy řetězce. Také obsahuje různé malé sémantické funkce pro zjednodušení kódu webové části.

Vyhledávání

O vyhledávání se stará třída **CardSearchRepository**, která posílá správné požadavky na vyhledávací servlet CouchDB-Lucene. Takový požadavek může vypadat například takto:

```
/retrobi/_fti/_design/index/basic_ocr_best
?allow_leading_wildcard=true&lowercase_expanded_terms=true
&qsplit=false&skip=0&limit=1
&q=%2Bdefault_lc%3A%28*psac%C3%AD*+%2B+*stroj*+%281870+to+1920%29%29
```

Tento dotaz byl zavolán na indexu **basic_ocr_best** (nejlepší dostupný textový přepis) a jedná se o dotaz necitlivý na velikost písmen. Z výsledků bude vybrán jen první lístek (skip = 0, limit = 1). Dotaz je „*psací* *stroj* [1870 TO 1920]“.

Rozsah lístků

Rozsah lístků je zde chápán jako interval lístků v katalogu. Tento interval může být libovolně velký, vždy je však nutné z něj vybrat několik *reprezentantů*, podle kterých se uživatel bude orientovat a rozsah měnit. To si můžeme představit jako listování šuplíkem, který si rozdělíme např. na deset stejně velkých částí. Pak se podíváme na první lístek každé části a tak zjistíme, ve které části najdeme lístek požadovaný.

Rozsah lístků se používá při procházení katalogu (část / skupina), ale i schránky a výsledků vyhledávání. U výsledků vyhledávání je ale rozsah omezen pouze na jednotlivé lístky (z výkonnostních důvodů).

K reprezentaci rozsah lístku nestačí dvě čísla (offset a limit), jak je zvykem v běžných stránkovacích aplikacích. Může totiž existovat i poměrně velký rozsah lístků, ze kterého je viditelných jen několik reprezentativních lístků. Například pro rozsah 1–1000 a počet lístků 10 jsou viditelné lístky 1, 101, 201, 301, . . . , 901. Rozsah tedy musí obsahovat informaci, kolik lístků má načíst celkem, z čehož dopočítá, kolik lístků se musí pokaždé přeskočit.

Pro ilustraci uvádíme v tabulce 3.1 několik příkladů. V každém řádku tabulky jsou tři vstupní údaje: počet lístků, číslo prvního lístku a velikost skoku mezi dvěma reprezentanty. Za nimi jsou v řádku uvedené čísla reprezentantů a odpovídající rozsahy. Každý viditelný lístek tedy vlastně reprezentuje další rozsah lístků (až do úrovně jednotlivých lístků, kdy každý lístek reprezentuje právě sám sebe).

Počet lístků	První	Skok	= Reprezentanti	= Rozsahy
54	1	10	1, 11, 21, 31, 41, 51	1–10, 11–20, 21–30, 31–40, 41–50, 51–54
54	7	10	7, 17, 27, 37, 47	7–16, 17–26, 27–36, 37–46, 47–54
842	3	100	3, 103, 203, 303, 403, 503, 603, 703, 803	3–102, 103–202, 203–302, 303–402, 403–502, 503–602, 603–702, 703–802, 803–842
842	1	1000	1	1–842
842	124	1000	124	124–842
4321	7	1000	7, 1007, 2007, 3007, 4007	7–1006, 1007–2006, 2007–3006, 3007–4006, 4007–4321

Tabulka 3.1: příklady rozsahů (intervalů) lístků a jejich reprezentantů

Z tabulky 3.1 vyplývají výpočty a různé způsoby reprezentace rozsahů.

Rozsah lístků je reprezentován dvěma *neměnnými* (konstantními) třídami: **CardRange** a **CardCatalogRange**. První z nich reprezentuje obecný rozsah, zatímco ta druhá rozsah v určitém katalogu a skupině. Druhá třída je postavena na první, a tak je dostačující věnovat se jí.

Třída **CardRange** obsahuje následující stavové proměnné, které se nikdy nemění (při změně zobrazeného intervalu je vždy vytvořen na základě starého intervalu nový):

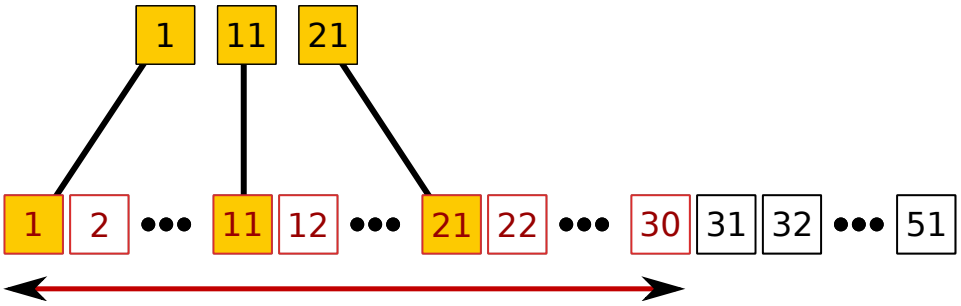
count: celkový počet lístků v procházené podmnožině (ne v celém katalogu);

offset: počáteční offset intervalu (přesah), tedy číslo prvního lístku bez jedné (přesah je číslován od 0, lístky od 1);

range: velikost intervalu, tedy počet lístků reprezentovaných tímto intervalem (např. pro lístky 1–99 má hodnotu 100);

bigstep: počet reprezentantů, tedy počet najednou zobrazených lístků (tato hodnota ovlivňuje i počet a členitost stromu intervalů);

flat: příznak plochého intervalu s jednou úrovní – pokud je příznak platný, neexistuje vyšší ani nižší interval, pohybovat se lze jen vlevo a vpravo;



Obrázek 3.6: ilustrace k rozsahu lístků: počet lístků = 51, počet reprezentantů = 3, interval = 1–30, výslední reprezentanti = 1, 11, 21

type: typ intervalu (pouze pro zobrazení správné „šipky“).

Dále obsahuje tovární metody pro vytváření dalších intervalů:

createFirst(): interval obsahující jen první lístek;

createLast(): interval obsahující jen poslední lístek;

createNext(): následující interval na stejné úrovni;

createPrevious(): předchozí interval na stejné úrovni;

createLower(int): nižší interval (parametr určuje který);

createUpper(): vyšší interval;

createForOtherCount(int): podobný interval pro jiný počet lístků.

Tovární metody vytváří na základě aktuálního intervalu jiné intervaly, které zachovávají jeho základní nastavení, tedy počet reprezentantů a celkový počet lístků. Při změně počtu reprezentantů nebo lístků se musí vytvořit nový základní interval, který bude tento počet respektovat, a to i v dalších jím vytvořených intervalech.

Příklad ilustruje obrázek 3.6.

Ostatní třídy

RetrobiLocker: Objekt obsahující rozličné zámky, na kterých probíhá synchronizace skupin proměnných.

CSVHistoryLogger: Třída, která zapisuje data ve formátu CSV do souboru, jehož jméno je dáno aktuálním datumem. Druhého ledna roku 2000 bude například zápis probíhat do souboru **2000-01-02.csv**. Tato třída slouží k průběžnému logování důležitých operací systému.

CzechAlphabet: Třída obsahující českou abecedu a metody pro řazení řetězců. Také zahrnuje funkcionalitu pro získávání prvního písmene skupiny a generování výchozí skupiny pro řazení.

DataLoader: Rozhraní objektu, který dokáže načíst uspořádanou podmnožinu položek zadané maximální velikosti od určité pozice. Tento objekt se používá ve spolupráci s třídou **CardRange**, která jej umí využít jako zdroj dat k získání reprezentantů, tedy např. lístků z intervalu.

DataToExport: Nastaví exportu schránky.

Triple: Uspořádaná trojice.

Tuple: Uspořádaná dvojice.

Systémová hlášení

Systémovým hlášením se v systému Retrobi rozumí speciální záznam v databázi. Tento záznam je reprezentován entitou **Message** a jeho nejdůležitějšími vlastnostmi je datum vložení, typ, obsah a potvrzení. Typ hlášení nabývá hodnot ze dvou kategorií: systémové události (events) a hlášení o problémech (problems). Systémové události přidává systém sám, zatímco hlášení o problémech jsou přidávána ručně uživateli. Administrátor může tato hlášení prohlížet ve Správě a potvrzovat ty, které vzal na vědomí, případně vyřešil.

V této části dokumentace se budeme zabývat jen systémovými událostmi. Existují tyto druhy systémových událostí:

EVENT_BATCH_UPDATED: Změna skupiny.

EVENT_OCR_UPDATED: Změna OCR.

EVENT_SEGMENTS_UPDATED: Změna segmentace.

EVENT_NOTE_UPDATED: Změna poznámky.

EVENT_URL_UPDATED: Změna URL.

EVENT_STATE_UPDATED: Změna stavu lístku.

EVENT_CARD_CREATED: Lístek založen.

EVENT_CARD_REMOVED: Lístek smazán.

EVENT_CARD_MODIFIED: Hromadná operace.

EVENT_MULTIPLE_CARDS_MODIFIED: Hromadná operace (více lístků).

EVENT_CARD_MOVED: Hromadný přesun.

EVENT_MULTIPLE_CARDS_MOVED: Hromadný přesun (více lístků).

EVENT_USER_REGISTERED: Registrace uživatele.

EVENT_USER_REMOVED: Uživatel smazán.

EVENT_IMAGE_CROSS_ON: Zaškrtnutí lístku.

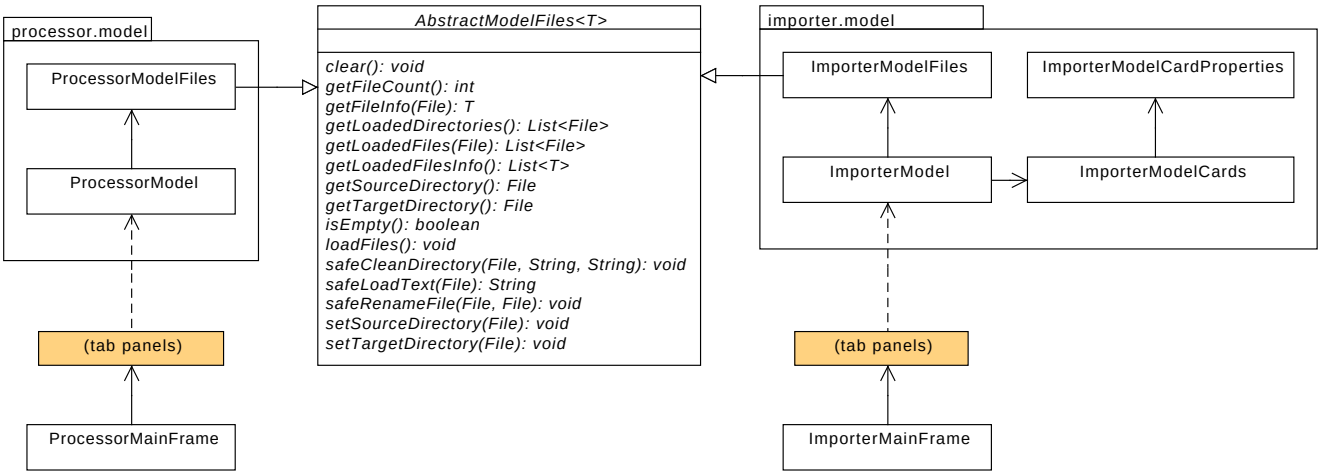
EVENT_IMAGE_CROSS_OFF: Zrušeno zaškrtnutí lístku.

Všechna hlášení se ukládají jako nepotvrzená až na několik výjimek. Hlášení o změně segmentace (EVENT_SEGMENTS_UPDATED) se po vložení automaticky uzávírá právě tehdy když je stav lístku vyšší než Přepsáno (REWRITTEN), tedy segmentováno a výše. Hlášení o změně stavu (EVENT_STATE_UPDATED) se uzavírá právě tehdy když je nový stav lístku vyšší než původní stav a zároveň je přihlášen editor či vyšší.

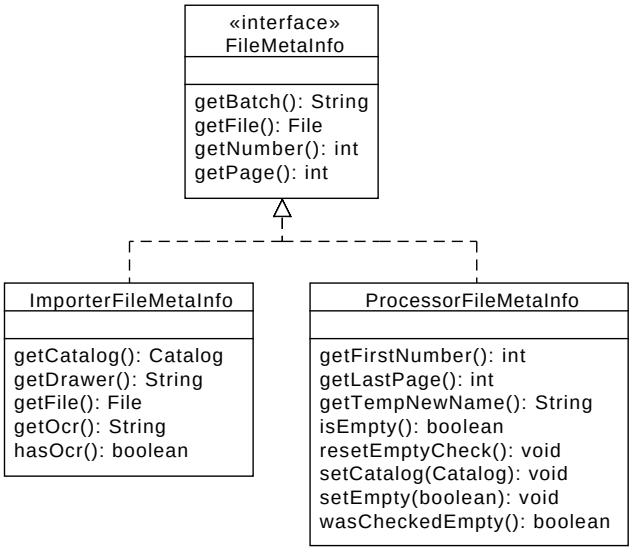
3.2.2 Jednoúčelové nástroje

Systém Retrobi obsahuje dva nástroje pro zpracování a import naskenovaných dat do databáze. Technicky jsou zde označeny jako *processor* (přejmenování, konverze, detekce prázdných stran lístků) a *importer* (zmenšení, zařazení, upload zpracovaných obrázků). Spouštějí se přes webové rozhraní (v sekci *Správa* / *Nástroje*). Technicky vzato je lze spustit i ručně ze zkompilovaného JARu (*retrobi-tools.jar*), ale nedoporučuje se to - spuštění přes webové rozhraní (technologie JNLP) totiž doplní jejich konfigurační parametry, což při manuálním spuštění chybí.

Oba nástroje (procesor i importer) pracují především s obrázkovými soubory. Proto oba obsahují kolekce pro uchování těchto souborů (**ProcessorModelFiles**, **ImporterModelFiles**). Společná či zobecněná funkcionalita shrnuta ve společném předkovi **AbstractModelFiles**.



Obrázek 3.7: základní struktura nástrojů (UML)



Obrázek 3.8: hierarchie metasouborů (UML)

Tento předek uchovává soubory spolu s meta-informacemi (**ProcessorFileMetaInfo**, **ImporterFileMetaInfo**), které jsou extrahovány po načtení nové sady souborů. Oba nástroje obsahují hlavní model, který se dále dělí na podmodely. Model poskytuje funkcionalitu a data nad ním postavenému grafickému rozhraní. Následuje detailnější popis obou modelů.

U procesoru neexistuje spojení s databází a vše se děje na úrovni souborů (např. přejmenování, přesun, detekce prázdných stránek, konverze). Proto je jeho model jednodušší a obsahuje pouze kolekci souborů.

U importeru existuje spojení s databází a z obrázkových souborů se vytváří lístky. Proto je jeho model rozdělen na kolekci souborů (**ImporterModelFiles**) a a kolekci lístků (**ImporterModelCards**). Kolekce lístků dále obsahuje model pro tvorbu a uchování jejich vlastností (**ImporterModelCardProperties**).

Oba nástroje svou činnost logují do souboru CSV.

Detekce prázdných stránek

Detekce prázdných stránek v aplikaci Retrobi není nic složitého, jen aplikace středoškolské statistiky a několik jednoduchých nápadů. Provádí se jednou, a to při zpracování lístku po skenování. Detekce se skládá z těchto kroků:

1. Z obrázku se vytvoří menší náhled, aby zpracování netrvalo tak dlouho.
2. Oříznou se hrany, které mohou obsahovat nepřesné, pomačkané či špinavé okraje lístku. Tím se zvýší účinnost rozpoznávání.
3. Z tohoto fragmentu se vypočítá barva papíru (barva, jejíž složky jsou mediány odpovídajících barevných složek všech pixelů – algoritmus: do tří polí se postupně uloží barevné složky všech pixelů, pole se seřadí a z každého se vezme prostřední prvek).
4. Vypočítá se světelnost barvy papíru (0.3 * r + 0.59 * g + 0.11 * b).
5. Světelnost barvy papíru se vynásobí malou konstantou a vyjde prahová světelnost, tedy nejvyšší světelnost, kterou může mít inkoust na daném lístku.
6. Spočítá se, kolik pixelů má podprahovou světelnost (= kolik inkoustu tam je).
7. Pokud relativní počet tmavých pixelů přeroste určitou mez, je lístek označen jako neprázdný, jinak je považován za prázdný.

Během procesu je použito několik parametrů, které byly náhodným testováním vyladěny tak, aby detekce dávala co nejlepší výsledky na testované množině.

3.2.3 Webová aplikace

Webová aplikace je postavena na frameworku *Wicket*. Nejprve zde stručně popíšeme základní rysy frameworku a způsob, jakým jsou v něm generovány stránky. Domovská stránka frameworku *Wicket* je **http://wicket.apache.org/**, kde se nachází dokumentace a příklady použití.

Hlavní třída webové aplikace systému Retrobi se jmenuje **RetrobiWebApplication**. Tato třída dědí od třídy **WebApplication** specifikované ve frameworku Wicket a je jakýmsi vstupním bodem do celého webového podsystému. Spravuje vytváření a obsluhu webových požadavků, řídí ukládání uživatelských relací (sessions), spouští dlouhé úlohy, pravidelně spouští systémovou údržbu a obsahuje názvy všech parametrů URL. Také se zde při spuštění aplikace (metoda **init()** inicializují všechny parametry frameworku Wicket tak, aby jeho chování odpovídalo požadavkům systému Retrobi. Při startu aplikace napsané nad frameworkem Wicket se také připojují všechny *zdroje*, což jsou v případě systému Retrobi soubory JNLP pro spuštění jednorázových importovacích nástrojů (tzv. „udělátek“, zdroj implementován třídou **JnlpResource**) a obrázky jednotlivých lístků (zdroj implementován třídou **CardImageResource**). Podobně se mapují i jednotlivé webové stránky – ke každé stránce je připojena „lidová“ adresa, na které ji lze najít.

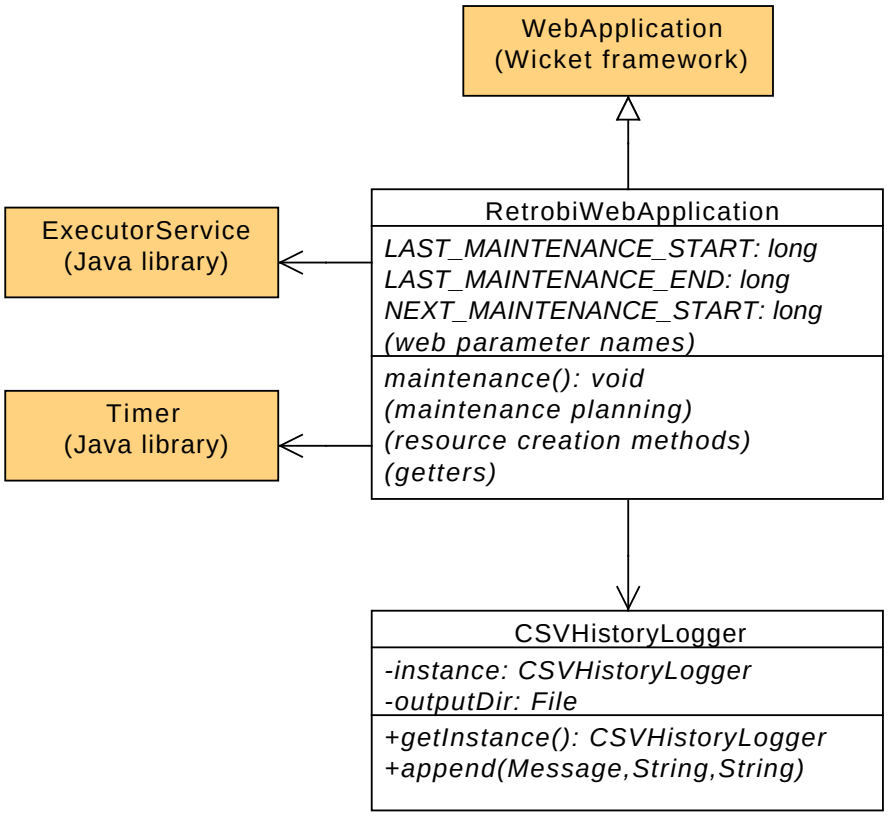
Stránka, komponenta, model

Základním objektem frameworku *Wicket* je *stránka*. Na stánce jsou umístěny komponenty. Komponenty lze seskupovat do větších celků pomocí tzv. *kontejnerů* – speciálních komponent, které mohou obsahovat jiné komponenty. Kontejnerem je například již zmíněná stránka, formulář, panel a další. Vzhled stránek a rozmístění komponent je definován pomocí lehce modifikovaného jazyka XHTML, který zde budeme označovat jako *Wicket HTML*. Ten navíc obsahuje tagy pro umisťování komponent.

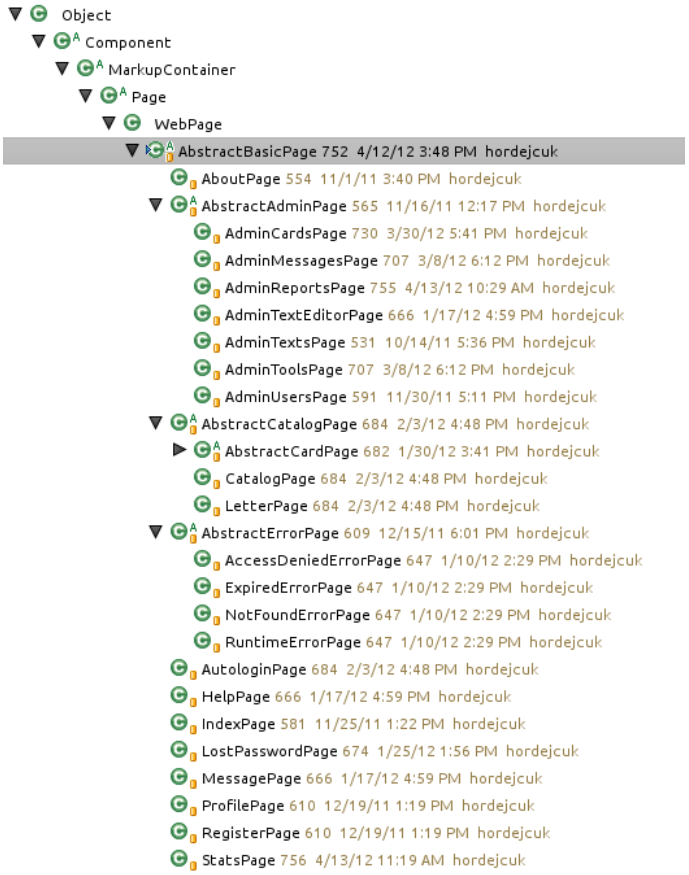
Při požadavku na danou stránku vezme framework správnou šablonu XHTML, zpracuje ji a postupně ke speciálním tagům přiřadí skutečné instance komponent.

```
Katalog: <h3 wicket:id="nadpis"></h3>
<form wicket:id="formular">...</form>

add(new Label("nadpis", catalog.getName()));
add(new Form("formular"));
```

Základními komponentami jsou třídy **Label** (jednoduchý text), **Panel** (panel obsahující samostatnou skupinu spolu souvisejících komponent), **Form** (formulář s ovládacími prvky), **Link** (hypertextový odkaz), **ListView** (panel s opakováním).



Obrázek 3.9: webová aplikace (UML)



Obrázek 3.10: hierarchie stránek

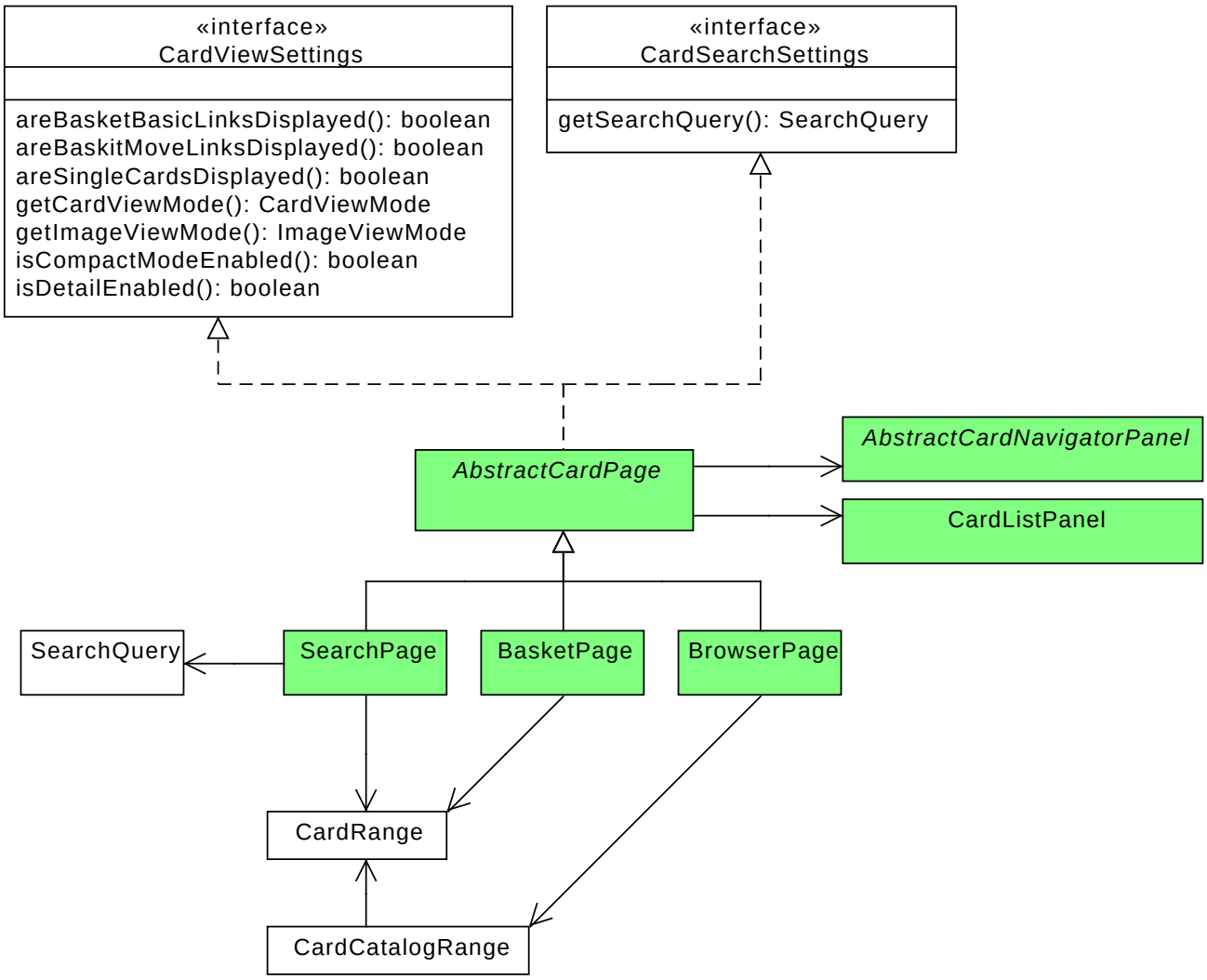
Hierarchie stránek

Uživatelská relace (Session)

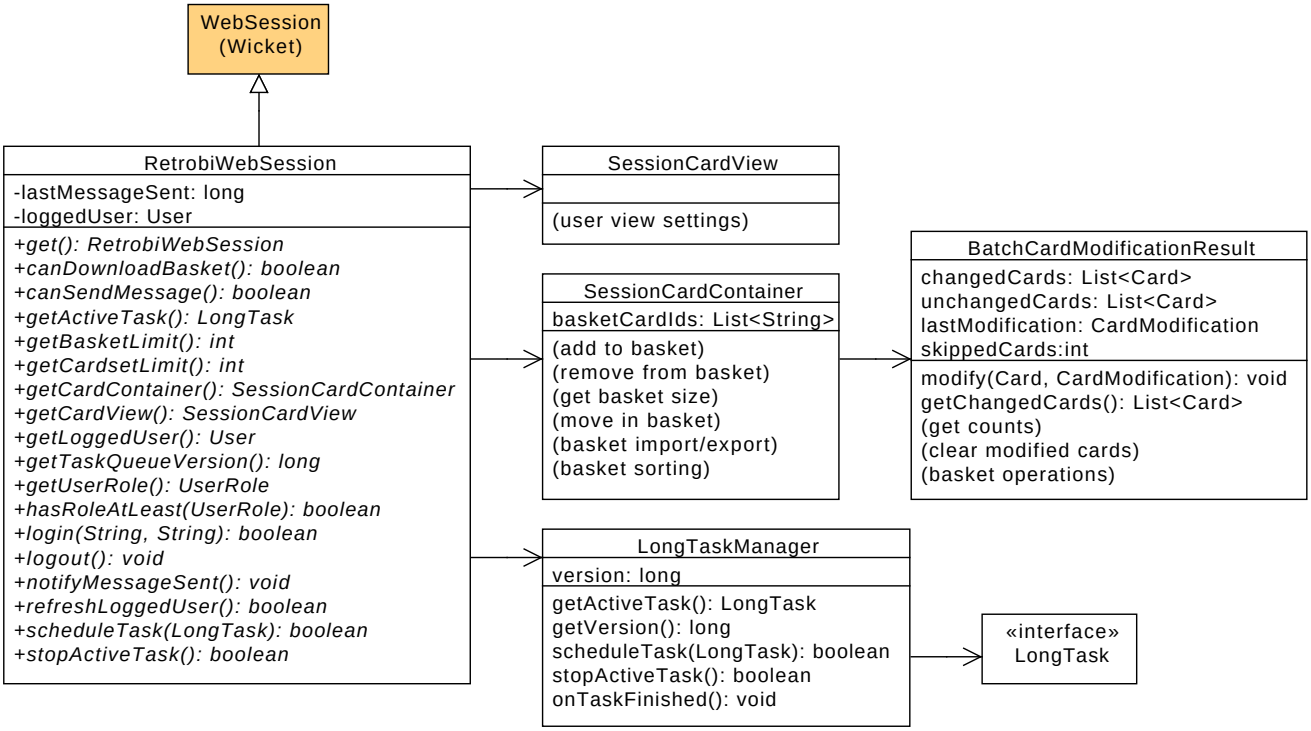
Ke každému návštěvníkovi je přiřazena tzv. *relace*, což je zjednodušeně řečeno jeho osobní paměťový prostor na serveru. Tato relace je vytvořena, pokud se návštěvník na server připojuje poprvé nebo po dlouhé době.

Každá nová návštěva uživatele vyústí ve vytvoření tzv. *relace* (session). Tato relace je následně svázána s konkrétním návštěvníkem (pomocí Cookies) a uchovává se, dokud nevyprší (session timeout, výchozí nastavení je 3 hodiny) nebo není webovým serverem zrušena.

Klíčovou funkcí relace je uchovávání přihlášení daného uživatele a jeho schránky. Relace dále podporuje dlouhé operace.



Obrázek 3.11: podmnožina tříd vztažených k hlavním stránkám s lístky (zeleně jsou vyznačeny komponenty, tedy vizuální prvky)



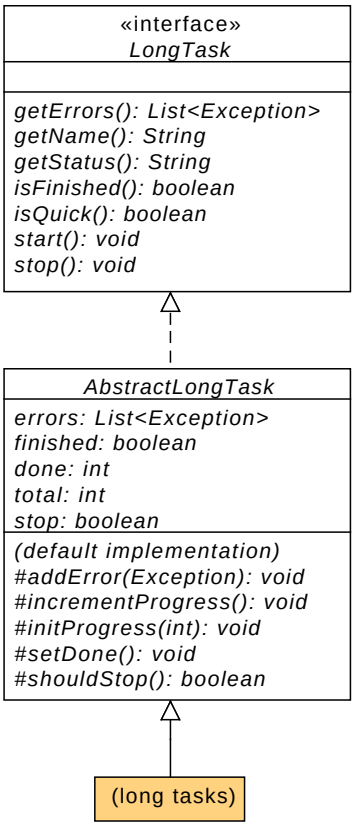
Obrázek 3.12: uživatelská relace (UML)

Model katalogu

Model katalogu je společný pro všechny uživatelské relace, aby se nemusel načítat pokaždé. Model katalogu je jakási cache, obsahující seznam katalogů, skupin, písmen a jejich vzájemné přiřazení. Načtení těchto informací trvá zhruba 5 minut. Model katalogu se proto aktualizuje jen na vyžádání (viz záložka Nástroje ve správě) nebo automaticky při každé denní údržbě.

Abeceda, řazení a ekvivalence písmen je ovlivnitelná jedinou položkou v nastavení, a to pravidly pro řazení písmen. Tato pravidla specifikují, která písmena jsou si „rovna“ a jaké je jejich vzájemné pořadí.

Při obnovování katalogu jsou načteny všechny skupiny, seřazeny dle jejich vlastnosti „skupina pro řazení“ a dále přiřazeny k jednotlivým písmenům. Vlastnost „skupina pro řazení“ je vytvořena při importu lístků do katalogu pomocí nástroje importer a automaticky převádí název skupiny na velká písmena a odstraňuje nepotřebné předložky v příjmení (van, von, aus, de a další).



Obrázek 3.13: dlouhé úlohy (UML)

Dlouhé úlohy

V systému Retrobi existuje několik operací, které obvykle trvají příliš dlouho, než aby byly vykonány během jednoho HTTP požadavku. Pro podobné úlohy byl vytvořen speciální systém, který tyto úlohy řadí do fronty a asynchronně spouští. Webová aplikace obsahuje komponentu, na které je zobrazen stav aktuální úlohy a velikost fronty. Odtud je možné sledovat průběh úlohy, případně ji zastavit. Komponenta se překresluje v pravidelném intervalu pomocí technologie AJAX. Celá stránka se automaticky obnoví po dokončení každé dlouhé úlohy. Chyby vzniklé při provádění úlohy jsou zobrazeny jako klasické informační zprávy do hlavního okna.

Všechny dlouhé úlohy (tasks) v systému Retrobi implementují rozhraní **LongTask**, které specifikuje metody pro zjišťování stavu úlohy (název, dokončeno, status, chyby), dále pro její zahájení a ukončení. Výchozí implementace některých těchto metod je umístěna ve společném předkovi třídy **AbstractLongTask**, od kterého všechny dlouhé úlohy dědí. Většina dlouhých úloh je spjata s hromadnou editací lístků, která byla popsána výše. Webová aplikace systému Retrobi obsahuje následující dlouhé úlohy:

AddLetterToBasketTask: přidá písmeno do schránky;

AddSearchToBasketTask: přidá výsledek vyhledávání do schránky;

BatchModificationTask: provede hromadnou operaci (obsahuje instanci třídy implementující rozhraní **CardModification** (viz výše), která provede vlastní úpravu každého vybraného lístku);

DeleteBatchTask: smaže zadanou skupinu;

MultipleCardMoveTask: hromadně přesune vybrané lístky na určenou pozici v katalogu;

RemoveLetterFromBasketTask: vyjme písmeno ze schránky;

RemoveSearchFromBasketTask: vyjme výsledek vyhledávání ze schránky;

SaveBatchModificationTask: uloží lístky úspěšně změněné hromadnou operací do databáze;

SortBasketTask: seřadí schránku jako katalog.

Dlouhé úlohy obsahují i speciální metodu **isQuick()**, která zjišťuje, zda je daná dlouhá úloha nakonec „rychlá“ a bylo by možné ji stihnout během jednoho HTTP požadavku. Přesný

způsob zjišťování tohoto příznaku je pro každou úlohu jiný. Některé úlohy mohou například považovat za „rychlé“ takové instance úloh, které pracují s méně než sto lístky. Tento příznak je využitý v uživatelském rozhraní – pokud uživatel požádá o spuštění dlouhé úlohy a tato úloha je „rychlá“, není přidána do fronty, ale okamžitě vykonána. To šetří uživatelův čas a ve většina případů zvyšuje předvídatelnost uživatelského rozhraní.

Automatická údržba

Při spuštění a každý den v zadanou hodinu (viz konfigurace systému) se spouští tzv. *denní údržba*. Během této údržby se vykonají všechny náročnější operace, které z dlouhodobého hlediska vylepšují výkon systému a aktualizují všechna neaktuální data. Během údržby se provedou následující operace:

- Aktualizace položkového rozpisu a definice indexů.
- PING na databázové pohledy (zajistí důslednou aktualizaci všech pohledů).
- Aktualizace model katalogu (katalogy, skupiny a informace o nich).
- Vyčištění indexu od smazaných dokumentů.
- PING a optimalizace všech vyhledávacích indexů (zajistí důslednou aktualizaci všech vyhledávacích indexů).

Kapitola 4

Uživatelská příručka

4.1 Uživatelské role

V systému Retrobi existují čtyři uživatelské role. Tyto role jsou uspořádány lineárně do hierarchie, přičemž vyšší role má vždy vyšší oprávnění než role nižší.

Nejnižší rolí je *návštěvník* (GUEST). Tuto roli má každý člověk, který přijde na web, aniž by cokoliv musel dělat. Návštěvník má určitá omezení co se týče velikosti schránky a operací, které může provádět. To proto, aby každý náhodný návštěvník nemohl příliš zatížit systém a jeho paměť, protože těchto lidí bude pravděpodobně mnoho a mohli by zbytečně omezovat prostředky dostupné pro vážné zájemce či odborníky.

Další rolí je *uživatel* (USER). Uživatelem se lze stát po platné registraci do systému. Po přihlášení do systému Retrobi se uživatel vzdává své anonymity a jako náhradu získá vyšší limity pro velikost schránky a možnost provádět další operace. Zároveň si může ukládat např. své schránky do databáze. Uživatel může také začít přepisovat lístky, což mu bude započteno k dobru.

Ještě vyšší rolí je *editor* (EDITOR). Uživatele na editory povyšuje administrátor, například po schválení jeho kvalitních přepisů. Editor je tedy určitý privilegovaný uživatel, u kterého existuje jistá odbornost a zájem vylepšovat obsah databáze systému Retrobi. Editor může narozdíl od uživatele upravovat různé další parametry lístků a manipulovat s nimi v rámci katalogu. Také má přístup k některým částem správy - konkrétně k hromadné editaci a analýzám.

Nejvyšší rolí je *administrátor* (ADMIN). Administrátor může naprosto vše a jeho práci se neklade žádné omezení. Má například k dispozici celou sadu nástrojů na webovém rozhraní.

Může také spouštět nástroje pro import dat. Při prvním spuštění systému je vytvořen výchozí administrátor s přihlašovacím jménem (loginem) `admin` a heslem `admin`. Tohoto prvotního administrátora můžete například použít pro spuštění nástrojů pro import dat (viz níže), další uživatelé se již mohou registrovat pomocí webového rozhraní.

4.2 Skenování a import dat

4.2.1 Spuštění nástrojů

Nástroje pro import naskenovaných dat se spouštějí z webu, a to ze sekce *Správa* / *Nástroje*. Tam jsou umístěny dva odkazy: *Stáhnout Udělátko 1* (přejmenování, konverze, detekce prázdných stran lístků) a *Stáhnout Udělátko 2* (zmenšení, zařazení, upload zpracovaných obrázků). Nástroje může spouštet jen uživatel s rolí *administrátor*. Výchozí administrátor je při prvním spuštění systému vytvořen s přihlašovacím jménem (loginem) `admin` a heslem `admin`.

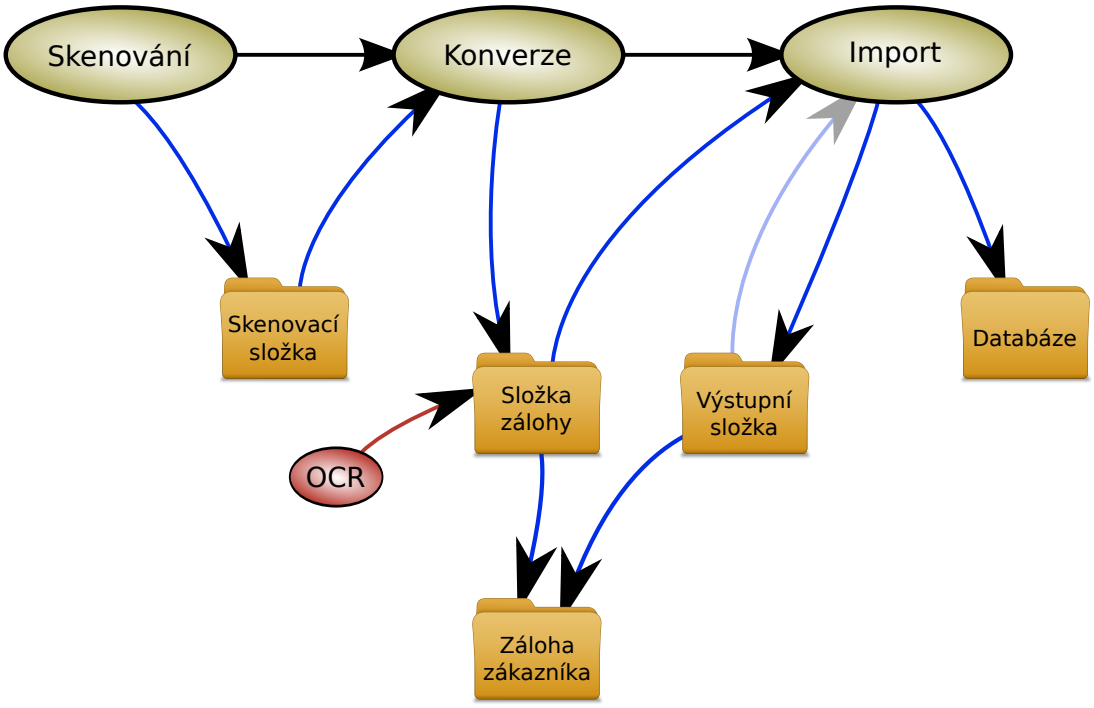
Technicky je spuštění programů řešené přes dynamicky generované JNLP (technologie firmy Oracle, která umožňuje jednoduchým XML souborem definovat spuštění libovolné Java aplikace). Během generování tohoto souboru JNLP dochází i k injekci konfiguračních parametrů, jako je adresa databázového serveru, jméno a heslo. Technicky je sice možné spustit nástroje i pomocí příkazové řádky, ale právě z důvodu chybějící konfigurace se to nedoporučuje. Pokud však přesto chcete otestovat, jestli se vůbec nástroje alespoň spustí, můžete použít tyto příkazy:

```
(U1) java -cp retrobi-tools.jar
 cz.insophy.retrobitool.processor.ProcessorMainFrame

(U2) java -cp retrobi-tools.jar
 cz.insophy.retrobitool.importer.ImporterMainFrame
```

4.2.2 Stručný proces

1. Skenování:

   (a) zapnout skener a nastavit dohodnuté parametry skenování (BMP, 24 bitů, bez vyhazování prázdných lístků);



Obrázek 4.1: Proces a tok dat během importovacího procesu

(b) naskenovat sadu lístků s názvy ve vzestupné číselné řadě a umístit je do složky, jejíž název odpovídá názvu sady;

(c) lístky přejmenovat podle toho, kolik kartiček (papírů) obsahují;

i. příklad lístku se třemi stránkami papíru:

- 5154x2.bmp = list 1, strana 1,
- 5155x2.bmp = list 1, strana 2,
- 5156x2.bmp = list 2, strana 1.

ii. příklad lístku se dvěma stránkami papíru:

- 10.bmp = list 1, strana 1,
- 11.bmp = list 1, strana 2.

2. Konverze a zmenšování obrázků:

(a) spustit nástroj pro konverzi obrázků;

(b) vybrat skenovací složku obsahující složky skupin;

(c) vybrat umístění složky zálohy;

(d) načíst soubory;

(e) spustit konverzi a počkat (jeden soubor cca 5 sekund);

(f) po skončení smazat označené prázdné stránky a makulatury.

3. Přiřazení OCR:
   (a) spustit nástroj pro vytvoření OCR;
   (b) výstupní textové soubory se musí jmenovat stejně jako obrázky (pouze přípona se změní na **.txt**);
   (c) tyto textové soubory se musí nacházet ve stejném adresáři jako zdrojové obrázky.
4. Import obrázků do databáze:
   (a) spustit nástroj pro import lístků;
   (b) vybrat složku se zálohou;
   (c) načíst soubory;
   (d) doplnit všechny známé údaje;
   (e) spustit upload a počkat.
5. Opakovat proces, dokud není katalog naskenovaný.

4.2.3 Podrobnější postup

Skenování lístků (skener a jeho obslužný software)

- **Vstup:** šuplík s připravenými lístky
- **Výstup:** naskenované obrázky z lístků v rostoucí číselné řadě s označenými vícestránkovými lístky

Skener musí být nastaven na formát TIFF a maximální kvalitu. Detekci prázdných lístků je nutné zakázat (tu provede náš nástroj později). Požadovaným výstupem jsou soubory TIFF v rostoucí číselné řadě (např. 1001.tif, 1002.tif, 1003.tif, . . .).

První číslo řady není důležité. Náš nástroj totiž zajímá pouze relativní pořadí, protože porovnává čísla vzájemně mezi sebou.

Ze šuplíku se vyjme ucelená skupina lístků (řádově 100–300 lístků, např. jeden autor, více autorů se stejnými počátečními písmeny, jedno období autora, . . .) a tato se vloží do skeneru. Spustí se skenování do správné odpovídající složky. Po dokončené této skupiny lze skenovat skupinu další, dokud není naskenován celý šuplík.

Dalším krokem je označení lístků, které se skládají z více karet. Byli jsme informování, že se objevují lístky i s pěti kartami. Aby o této skutečnosti nástroj věděl a přiřadil tak naskenované karty jednoho lístku správně k sobě, je nutné mu tuto informaci vložit do názvů souborů. Toho lze dosáhnout připojením řetězce **xN** za názvy všech souborů vícekartového lístku, kde **N** je počet jeho karet. Jednokartové lístky se nemusí řešit (tzn. **x1** přidávat nemusíte).

Pro vyšší názornost uvádíme několik příkladů:

- Jedna karta (není nutné označovat):
  - Po sobě jsou očekávány dva soubory (lichá a sudá strana).
  - příklad: `10512.tif`, `10513.tif`
- Dvě karty (přidat **x2** za název všech souborů lístku):
  - Po sobě jsou očekávány čtyři soubory (2 x lichá a 2 x sudá strana).
  - příklad: `100x2.tif`, `101x2.tif`, `102x2.tif`, `103x2.tif`
- Tři karty (přidat **x3** za název všech souborů lístku):
  - Po sobě je očekáváno šest souborů (3 x lichá a 3 x sudá strana).
  - příklad: `1x3.tif`, `2x3.tif`, `3x3.tif`, `4x3.tif`, `5x3.tif`, `6x3.tif`
- Více karet:
  - Obdobně, princip by měl být zřejmý.

Po naskenování celého šuplíku doporučujeme provést zbytek procesu tak, jak je tu popsáno, a až poté pokračovat šuplíkem dalším.

Zpracování lístků (Udělátko 1 / processor)

- **Vstup:** naskenované soubory lístků v adresářové struktuře
- **Výstup:** přejmenované soubory a označené prázdné strany

Po naskenování lístků spusťte první nástroj (udělátko 1).

Nejprve pomocí tlačítka „Procházet...“ vyberte složku, ve které jsou umístěny naskenované soubory TIF. Po vybrání cesty proveďte načtení pomocí tlačítka „Načíst soubory“. Program ověří, zda jsou názvy a soubory platné, a načte je do tabulky. V ní ověřte, že jsou vypočítané hodnoty správné (hlavně počet stránek). Poté vyberte složku se zálohou, opět pomocí odpovídajícího tlačítka „Procházet...“. Nyní je nastaven vstup i výstup, program může začít převádět soubory. Tento proces spustíte tlačítkem „Spustit zpracování“. V jeho průběhu se zobrazí ukazatel průběhu a tlačítko pro jeho zrušení.

Pokud dojde k chybě, neočekávanému vypnutí počítače nebo k jiné podobné zvláštní události, doporučujeme smazat ty soubory, které se před přerušením stačily nekompletním procesem vytvořit. Poté proces opakujte a pro jistotu stále sledujte a kontrolujte zobrazené hodnoty.

Po dokončení se v záloze vytvoří stejná adresářová struktura, jako ve zdrojové složce naskenovaných obrázků, obsahující přejmenované soubory. Název souborů je tvořen dle názvu skupiny, čísel kartiček a stránek.

Příklad: **.../skenovani/backup/suplik1/Alex/Alex-2-1.tif**

Zároveň s tím se detekují prázdné stránky (strany lístků bez textu), které se označí speciální předponou. Tyto soubory a makulatury se po řádném ověření musí smazat.

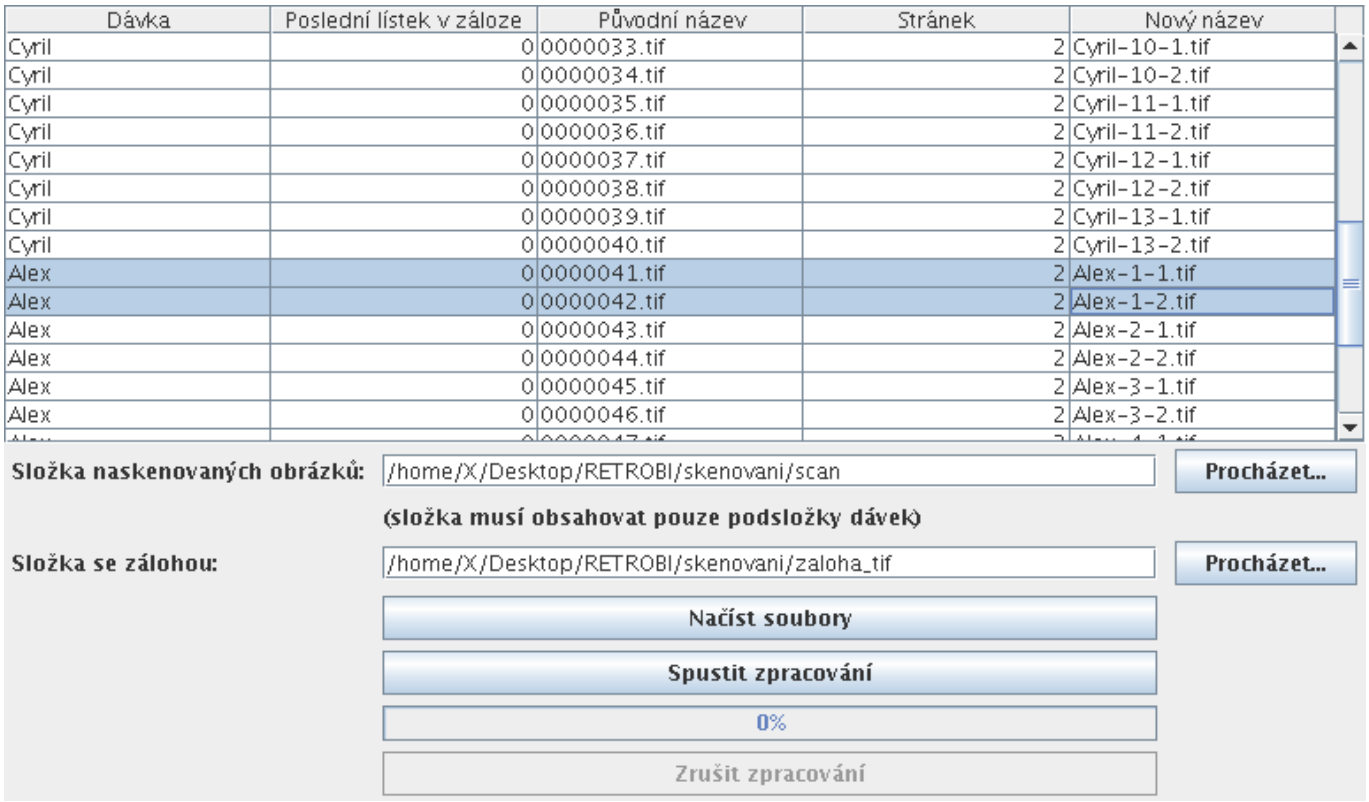
Příklad: **.../skenovani/backup/sss1/Alex/!PRAZDNY_Alex-3-2.tif**

Dalším krokem je načtení OCR. Spusťte program pro tvorbu OCR a zajistěte, aby vytvořil textové soubory s přepisem textu jednotlivých lístků. Tyto soubory musí být ve stejném adresáři jako zdrojové obrázky a musí se jmenovat stejně, jako zdrojové obrázky až na to, že končí příponou **.txt**.

Import lístků (Udělátko 2 / importer)

- **Vstup:** přejmenované soubory
- **Výstup:** lístky v databázi

Dalším velmi důležitým krokem je import naskenovaných lístků do databáze online katalogu. Naskenované soubory jsou z minulého kroku již přejmenované, prázdné stránky +



Obrázek 4.2: Seznam načtených souborů

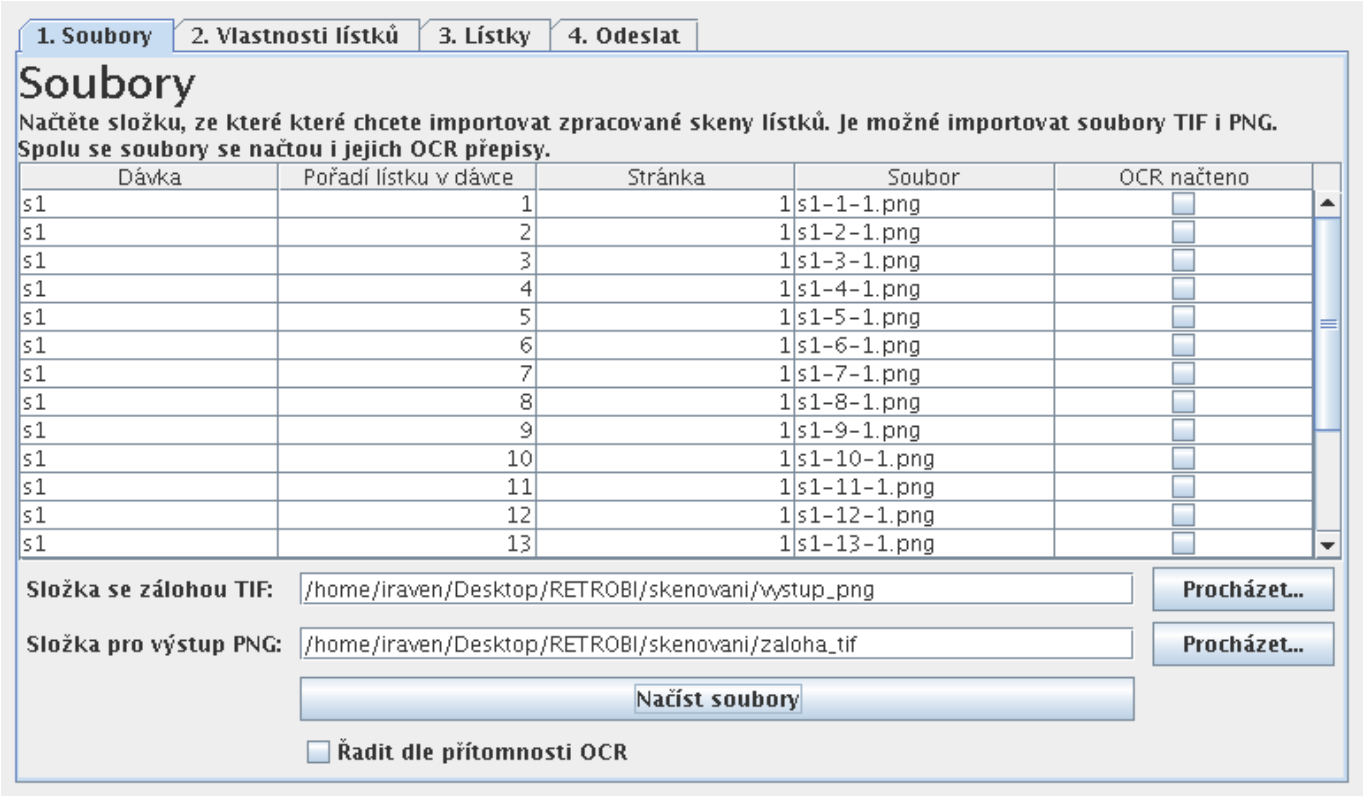
makulatury jsou smazané a u souborů se volitelně nachází textové soubory s jejich OCR přepisy.

Spusťte druhý nástroj (udělátko 2). Kořenovou složku zálohy vyberete kliknutím na tlačítko „Procházet...“. Poté vyberte složku, do které chcete ukládat výstupní soubory PNG, což jsou zmenšeniny původních obrázků. Do této výstupní složky se každý soubor zkopíruje s celou cestou, relativní k vybrané kořenové složce zálohy.

Nástroj kromě TIFFů dokáže načítat i zpracované PNG soubory.

Poté načtěte soubory tlačítkem „Načíst soubory“ a zkontrolujte, zda jsou údaje v tabulce správné a skutečně odpovídají lístkům, které chcete importovat. Indikuje se zde i přítomnost OCR přepisu. Soubory je možné podle tohoto příznaku i řadit (viz checkbox „Řadit dle přítomnosti OCR“ dole).

Po načtení souborů a kontrole přejděte na další záložku, to jest „Vlastnosti lístků“. Zde vyplňte všechny údaje, které jsou společné všem načteným lístkům. Zpravidla je to „Šuplík“. Dále zadejte své jméno jako hodnotu parametru „Vložil“. To usnadní případné dohledávání. Pomocí prvků v dolní části obrazovky můžete přidávat a odebírat parametry další. Parametry, u kterých je v hodnotě uvedeno „(automaticky)“, doplní nástroj automaticky. Tyto řádky ani nejdou odebrat.



Obrázek 4.3: Záložka se soubory

Nakonec nezapomeňte v dolní části vybrat část katalogu.

Po vyplnění všech hodnot se můžete přesunout na další záložku „Lístky“. Tato obrazovka slouží především ke kontrole. Jsou zde pod sebou vypsány jednotlivé lístky a konečné hodnoty všech parametrů. Pokud se vám zdá, že je něco špatně, vraťte se na předchozí záložky a hodnoty opravte. Po kontrole pokračujte na záložku poslední.

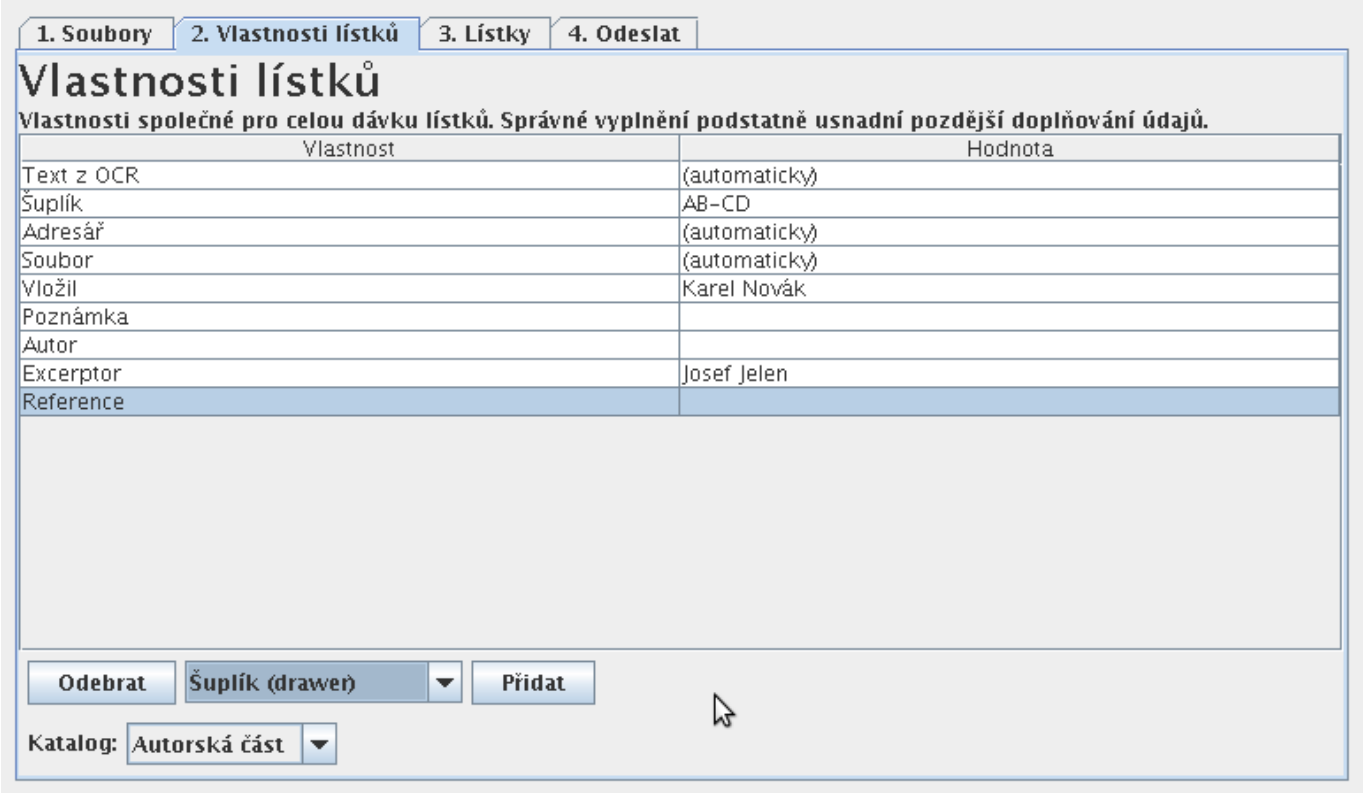
Na poslední záložce „Odeslat“ je tlačítko „Odeslat na server“, které spustí upload lístků do databáze. Tento proces může chvíli trvat, protože jsou lístky zmenšovány a konvertovány.

Nástroj u každého souboru ověří, zda se již v databázi nenachází. Pokud již soubor v databázi existuje, celý lístek se přeskočí.

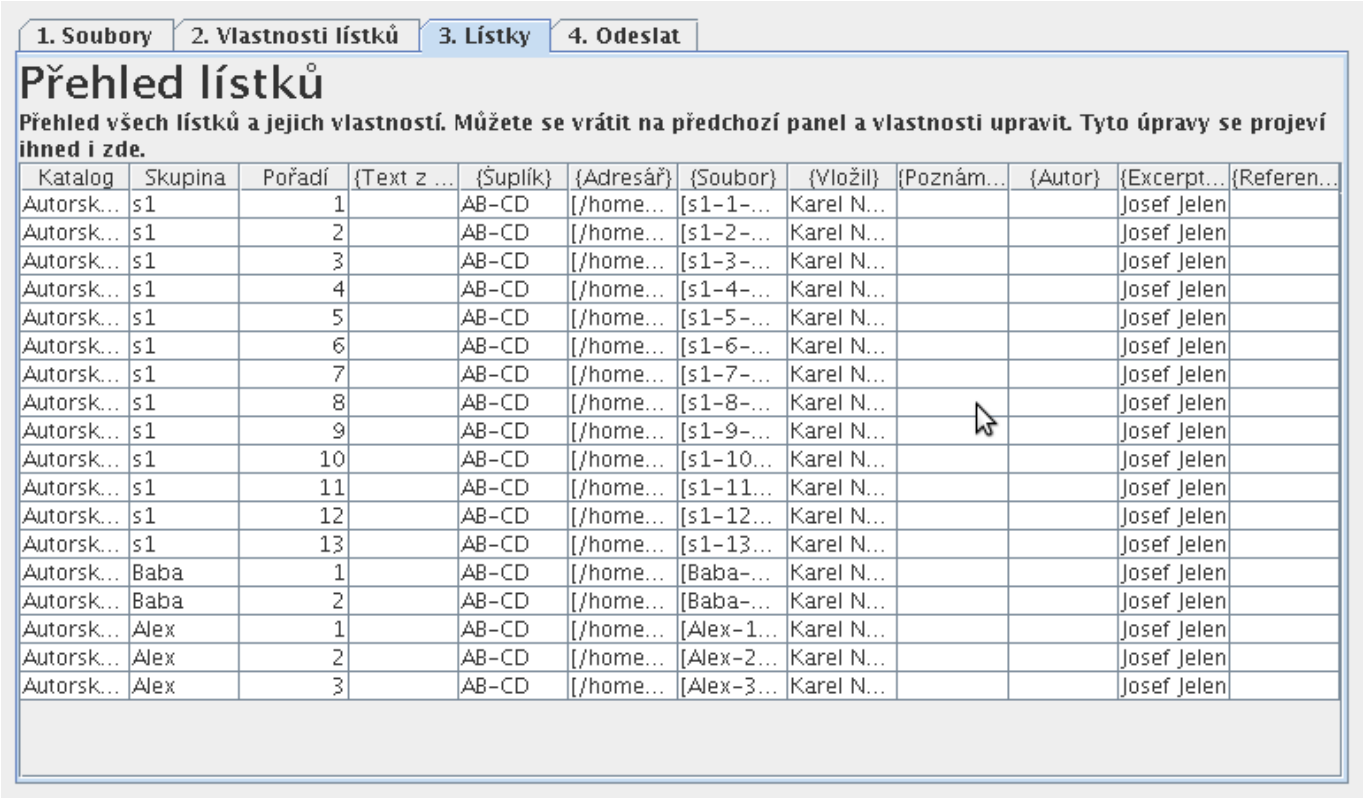
Dojde-li v tomto procesu k chybě nebo je počítač nečekaně vypnutý, budou se lístky v databázi nacházet jen částečně. Výše popsaný mechanismus detekce duplikátních souborů by měl zamezit opakovanému odeslání stejného lístku.

Po dokončení uploadu se zobrazí informační dialog, který obsahuje celkové počty lístků (odesláno, přeskočeno, chyb).

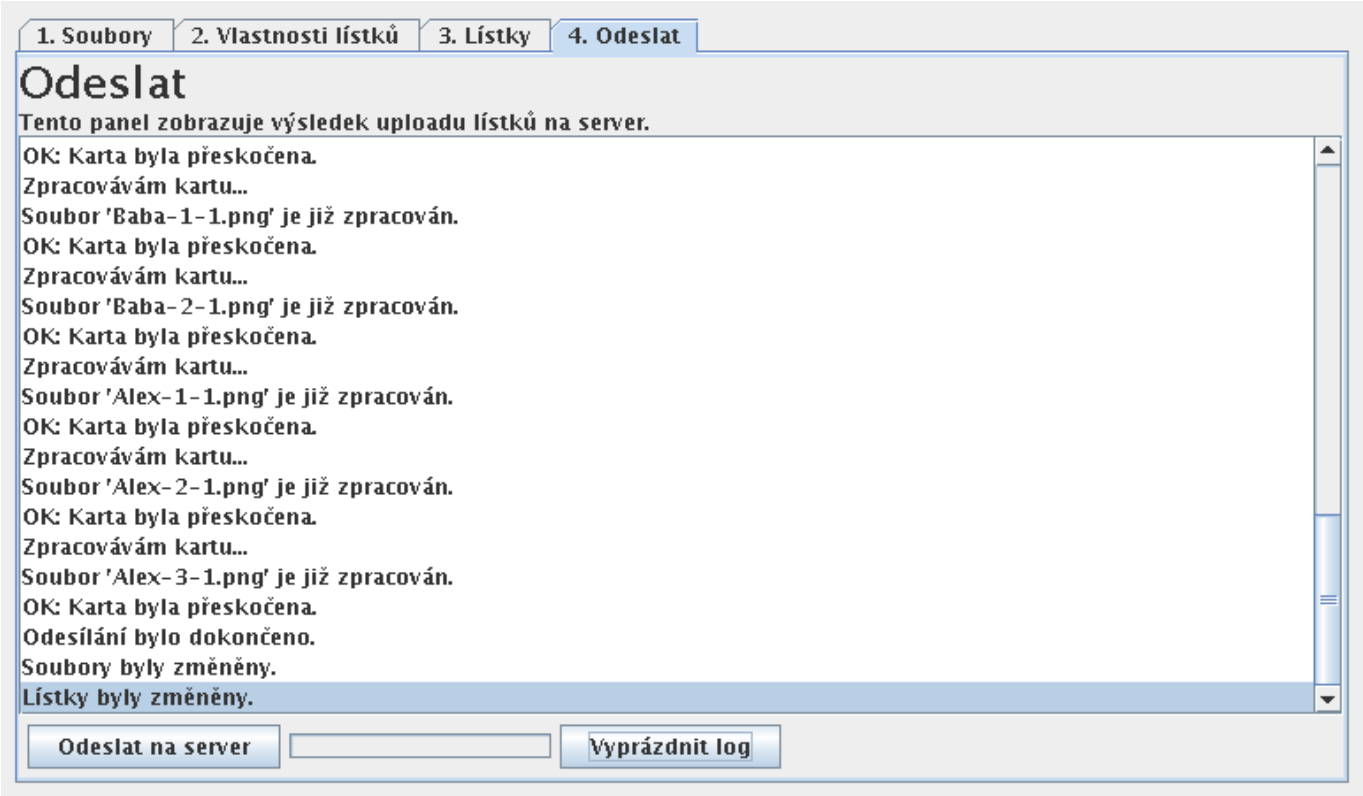
Tímto byl import lístků dokončen. Celý proces se opakuje, dokud není vše naskenováno.



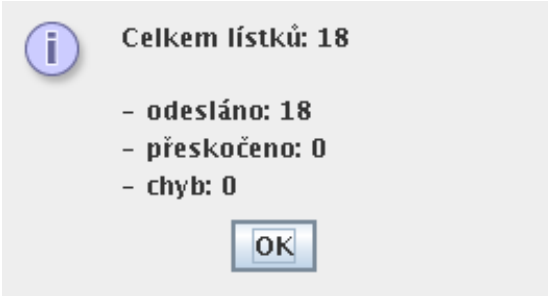
Obrázek 4.4: Záložka se společnými vlastnostmi lístků



Obrázek 4.5: Záložka s přehledem všech lístků



Obrázek 4.6: Záložka umožňující upload lístků na server



Obrázek 4.7: Informace o průběhu všech operací

4.3 Webová aplikace

Detailní popis webové aplikace se nachází na webu, a to v sekci „Nápověda“. Zde uvedeme pouze stručné shrnutí funkcí a obsažených stránek.

4.3.1 Procházení lístků

Srdcem webové aplikace je procházení lístků, a to v katalogu, ve schránce a ve výsledcích vyhledávání. Procházení katalogu a schránky je podobné – je k dispozici nějaký pevně daný seznam lístků (v případě katalogu je to zvolená skupina, v případě schránky je to její obsah) a uživatel jím prochází po hierarchicky uspořádaných intervalech. Procházení výsledku vyhledávání se liší tím, že se všechny výsledky nenačítají najednou a neexistuje hierarchie intervalů. Vždy je viditelná pouze nejnižší úroveň, a to jednotlivé lístky.

Tento způsob procházení je podobný přirozenému listování ve fyzické kartotéce. Pokud si otevřeme šuplík s pěti sty lístky a máme za úkol najít lístek č. 134, neprocházíme lístky po jednom. Víme, že je šuplík seřazený a proto se stačí dostat za první stovku, za třetí desítku a nakonec o čtyři lístky dál.

Procházení po intervalech je podobné. Nejprve zvolíme nejvyšší interval, například tisícovky. Potom volíme stovky, desítky, až nakonec vidíme jednotlivé lístky.

Každý interval lístků je reprezentován zvoleným počtem reprezentantů – čili lístky, které jsou ze seznamu lístků vybrány po rovnoměrných skocích a zobrazeny uživateli. Dále budeme předpokládat, že požadujeme $S = 10$ reprezentantů na interval (tento počet může uživatel ve webové aplikaci měnit na 5, 10, 20, 50, 100). Nižší počet reprezentantů urychluje načítání stránky, ale zároveň zvyšuje počet možných intervalů.

Nyní trochu formálněji. Máme-li tedy k dispozici například $N = 1258$ lístků, nejvyšší mocnina $M = S^x$, kde $M < N$ je rovna $M = S^x = 10^x = 1000 \rightarrow x = 3, M = 1000$. Nejvyšší intervaly tedy budou mít velikost 1000 a budou to intervaly 1–1000 a 1001–1258. Mezera mezi reprezentanty je 1000, takže mezi každými dvěma z nich se bude nacházet dalších $M - 1 = 1000 - 1 = 999$ lístků.

Uživatel si jeden z těchto intervalů vybere, například interval 1–1000. Opět vytvoříme podintervaly s 10 reprezentanty. Vzniknou „nižší“ intervaly 1–100, 101–200, a tak dále. Potom je možné mezi intervaly procházet, například „vlevo“ od 201–300 je 101–200, „vpravo“ je 301–400 a „nahoře“ je 1–1000.

Vlastní způsob reprezentace lístku během procházení si může uživatel měnit. Může si například zobrazit jen vršky lístků, tedy zhruba jejich horní třetinu. Podle ní už se může poměrně dobře v katalogu orientovat a navíc ušetří vertikální prostor na stránce pro další lístky. Zvolit si ale může i kompletní či textové zobrazení.

Katalog

Lístky jsou na webu organizovány následovně: katalog obsahuje několik částí. Každá část obsahuje písmena a pod každým písmenem jsou skupiny. Toto členění je vytvořené tak, aby skupiny obsahovaly nejvýše stovky lístků. V celém katalogu se nachází zhruba jeden a půl milionu lístků.

Uživatel po volbě části, písmena a skupiny prochází jednotlivé lístky klasickým intervalovým průchodem, který byl popsán výše.

Schránka

Schránka je množina uživatelem vybraných lístků, které „sbírá“ při procházení katalogem a výsledků vyhledávání. Tuto množinu může různě organizovat, řadit a měnit. Schránka je uložena v uživatelské relaci (Session), a tak je po jejím zneplatnění ztracena. Uživatel si ji ale může uložit na server anebo na disk.

Schránku je možné exportovat do dvou formátů: ZIP souboru, který obsahuje jednotlivé obrázky a textové soubory s přepisem (a další), nebo ZIP souboru s velkým RTF dokumentem.

Schránku lze procházet klasickým intervalovým průchodem, který byl popsán výše.

Vyhledávání

Fulltextové vyhledávání začíná v okamžiku, kdy uživatel vyplní vyhledávací formulář a odešle jej. Požadavek je zpracován vyhledávacím servletem CouchDB-Lucene, který provede vlastní vyhledávání. Výsledky jsou poté odeslány zpět webové aplikaci, která je zpracuje a zobrazí, přičemž ve výsledcích zvýrazní „nejlepší“ fragmenty, kde došlo k nálezu.

Vyhledávání se zobrazuje jen na úrovni jednotlivých lístků. To znamená, že se vždy zobrazí jen S lístků a není možné přejít o úroveň výš, pouze vlevo a vpravo.

Je tomu tak z výkonnostních důvodů. Kdyby se celý výsledek nahrál do paměti, mohl by se procházet stejným způsobem jako schránka. Pokud by však bylo výsledků mnoho, například několik set tisíc, omezovaly by množství volné paměti dostupné dalším uživatelům. Také by se značně snížila rychlost vyhledávání, protože by se musely přenést klíče všech nalezených dokumentů. Navíc by se uživatel ve výsledku ani nevyznal.

Nemožnost procházet výsledky vyhledávání hierarchicky není velká škoda, protože výsledky nejsou seřazeny jako v katalogu, ale zpravidla podle počtu shod s vyhledávacím dotazem.

Výkon vyhledávání dramaticky snižují tzv. *divoké karty*, tedy hvězdičky a otazníky. Nejdéle trvá dotaz obsahující výrazy s hvězdičkami z obou stran. Pokud se takových výrazů sejde mnoho, trvá vyhledávání až desítky sekund. Časový limit pro vyhledávání je ale shora omezen (viz konfigurace CouchDB-Lucene a CouchDB výše).

Výsledky vyhledávání je možné přidávat a odebírat ze schránky.

4.3.2 Nápověda

Na webu jsou k dispozici všechny nápovědné texty, které byly vytvořeny na straně ÚČL. Nápovědu lze rozdělit na dvě části: nápověda ke členění katalogu a nápověda k jednotlivým funkcionalitám. Nápovědu lze editovat jako ostatní texty stránek ve správě. Pro jednoduchost zde uvedeme několik nejčastěji používaných XHTML tagů a zápisů.

```
<h1>nadpis</h1> <h2>nadpis</h2> <h3>nadpis</h3>
<p>odstavec</p> <em>kurzíva</em> <strong>tučné</strong>
<a href="url">odkaz</a>
<ul><li>seznam</li></ul> <ol><li>seznam</li></ol>
<table><tr><th>nadpis</th></tr><tr><td>buňka</td></tr></table>
<a href="#sekce">odkaz na sekci</a>
<h3 id="sekce">nadpis sekce</h3>
&amp;quot; = uvozovka
&amp;amp; = amperstand
```

4.3.3 Správa

Správa je chráněná sekce webu, která je přístupná jen administrátorům a částečně i editorům. Skládá se z několika částí, které umožňují spravovat jednotlivé součásti systému.

V sekci „Hlášení“ se nachází seznam hlášení s filtrem. Hlášení je zde možné potvrzovat a přecházet na jednotlivé lístky. Potvrzená hlášení starší než měsíc je možné stahovat ve formátu CSV a mazat, aby byl v databázi větší pořádek.

V sekci „Hromadná editace“ je několik formulářů pro hromadnou změnu lístků ve schránce (přidat hodnotu, odebrat hodnotu, změnit skupinu, odkaz, poznámku, vytvořit nový lístek). Po dokončení hromadné operace jsou ovlivněné lístky rozděleny na dvě skupiny: lístky, u kterých byla změna úspěšně provedena; a lístky, u kterých změna z různých důvodů neproběhla. Změny jsou provedeny pouze v paměti, je tedy možné vzít neuložené změny zpět nebo naopak vybrané změny potvrdit a uložit natrvalo do databáze.

V sekci „Uživatelé“ se nachází seznam uživatelů s možností upravit jejich limity a role. Kompletní údaje všech uživatelů (samozřejmě bez hesel) je možné stáhnout ve formátu CSV. Pod tabulkou uživatelů jsou zobrazeny statistiky rolí a institucí, které mají uživatelé nastavené v profilu.

V sekci „Texty“ je možné upravovat jednotlivé stránky na webu. Existují dva druhy textů: texty krátké a texty dlouhé. Texty krátké neobsahují žádné formátovací značky, texty dlouhé se zapisují ve formátu XHTML 1.0 Strict. Při editaci doporučujeme „odkoukat“, jak se co zapisuje a hodně napodobovat, není to složité.

V sekci „Analýzy“ se nachází nejrůznější přehledy. První část se skládá ze statistik (nejvíce přepisů, počet obrázků), druhá část obsahuje informativní údaje o možných chybách v datech (chybné lístky, skupiny bez prvního lístku, skupiny s nesouvislým číslováním) a přehledy (seznam lístků, u nichž je název skupiny odlišný od názvu skupiny pro řazení, bez citlivosti na velikost písmen).

V sekci „Rejstřík“ je roleta s atomickými atributy. Tyto atributy mohou obsahovat hodnoty a je užitečné vědět, jaké různé hodnoty to jsou. K tomu slouží formulář, ve kterém si administrátor vybere atribut a systém mu zobrazí číselník všech různých hodnot tohoto atributu v databázi a počet lístků, které tuto hodnotu obsahují.

V sekci „Nástroje“ jsou věci spíše technického charakteru. V první řadě je to seznam CSV logů s možností jejich stažení a možnosti spuštění jednorázových nástrojů pro import dat (tzv. „udělátka“). Následuje informace o posledním a dalším plánovaným spuštění systémové údržby. Pod tím je ještě několik odkazů pro správu systému, které není nutné za normálních okolností aktivovat. Popíšeme si zde, co dělají:

Znovu načíst a seřadit skupiny: Seznam skupin existuje v jedné instanci na celém serveru a všichni návštěvníci jej využívají společně. Protože se struktura katalogu tak

často nemění, není nutné ji znovu a znovu při každém požadavku naplňovat. Naplnění je navíc časově náročné. Proto byl vypracován mechanismus, který tuto cache automaticky obnoví jen jednou za den (viz kapitoly o konfiguraci). Kliknutím na tento odkaz ale dojde k okamžitému obnovení této cache, tedy k novému načtení a seřazení všech skupin. Tato operace trvá několik minut.

Znovu načíst definici atributů a indexů: Načte soubor s definicemi položkového rozpisu a znovu vytvoří databázové pohledy a rejstříky. Poté obnovi seznam a vlastnosti vyhledávacích indexů.

Znovu načíst definici indexů: Načte soubor s definicemi dodatečných vlastností vyhledávacích indexů a obnoví jejich seznam.

Vyčistit indexy: Ručně vynutí čištění vyhledávacích indexů, které z nich odstraní nadbytečné informace o již neexistujících dokumentech.

Optimalizovat indexy: Ručně vynutí optimalizaci vyhledávacích indexů. Podrobnosti o této proceduře naleznete v dokumentaci Lucene.

PING na všechny indexy: Odešle krátký testovací vyhledávací dotaz na všechny indexy. Způsobí, že se případné neaktuální indexy začnou na pozadí přepočítávat.

PING na všechny pohledy: Odešle krátký testovací dotaz na všechny databázové pohledy. Způsobí, že se případné neaktuální pohledy začnou ihned přepočítávat. Může trvat až několik hodin (podle toho, kolik pohledů se v průběhu bude aktualizovat).

Co se seznamu CSV logů týče, bude neustále narůstat. Aby nebyla stránka příliš rychle a zbytečně zahlcena, doporučujeme staré CSV logy pravidelně mazat (přesné umístění složky je zobrazeno nad seznamem logů). Aby je správce systému nemusel mazat ručně, doporučujeme vytvořit automatickou úlohu pro službu CRON. Uvedený skript jednou měsíčně promaže CSV logy starší než 60 dní.

```
cd /tmp
echo -n "find CESTA_CSV -mtime +60 -exec rm {} \;" >> rm_old_logs.sh
mv rm_old_logs.sh /etc/cron.monthly/rm_old_logs.sh
```

Kapitola 5

Budoucí rozvoj

Zde je několik nápadů pro budoucí rozvoj projektu:

- aktualizovat na Wicket 1.5;
- vzít vyhledávání pod vlastní režii (využít přímo jádro Lucene) a tím se zbavit závislosti na CouchDB-Lucene;
- index vyhledávacího engine přesunout na SSD;
- obrázky lístků ukládat do adresářové struktury jako prosté soubory, např. A/Čapek/1/1.png.