Multimediale Informationssysteme und Datenbanken

Auf dieser Seite finden sich Anleitungen, Hilfestellungen und Code-Beispiele für die Projektarbeiten im Kurs „Multimediale Informationssysteme und Datenbanken“.

Projektsprechstunden

Tipps zu Database Servers

couchdb

Vgl. auch Foliensätze in GRIPS:
https://elearning.uni-regensburg.de/mod/resource/view.php?id=207437 und
https://elearning.uni-regensburg.de/mod/resource/view.php?id=216590 sowie das dortige TestCouch-Paket: https://elearning.uni-regensburg.de/mod/resource/view.php?id=216571

Lokale Installation

Die Installation eines couchdb-Servers erfolgt in der Regel per Installationsprogramm oder Paketsystem und ohne Probleme. Auf Mac OS kann entweder auf die empfohlenen homebrew-Methode zurückgegriffen werden oder ein, veraltets, Paket des CouchDBX-Projektes installiert werden (Bswp.: http://dl.couchone.com/dl/384fe8cac77f981551a6632c020259a3/CouchDBX-1.0.2.0.zip).

Java-Anbindung

Für die clientseitige Anbindung eines Java-Programms an einen couchdb-Server können unterschiedliche Klassenbibliotheken verwendet werden. Für den leichten Einstieg sind lightcouch und couchdb4j zu empfehlen. Beide Bibliotheken verlangen den Import einiger Klassen aus dem Apache-Commons-Projekt.

Code Snippets für Projektarbeiten

Java & couchdb mit couchdb4j

Installation

Um couchdb4j in einem eigenen Programm zu nutzen müssen die notwendigen Dateien heruntergeladen und in das Projekt integriert werden. Die Klassen müssen im Build-Path zugänglich sein, also zum Beispiel in einem package importiert werden oder als isoliertes Projekt mit entsprechenden Abhängigkeiten angelegt werden. Für die Arbeit mit Eclipse kann etwa das von github heruntergeladene zip-Archiv entpackt werden und der Ordner com aus dem src/java-Verzeichnis des entpackten Archivs in den eigenen src-Ordner kopiert werden. In Eclipse sollte dann

unter src ein Paket `com/fourspaces/couchdb` angelegt sein.

Abhängigkeiten

Die Abhängigkeiten können durch Import der nötigen *.java- oder *.jar-Dateien in den Klassenpfad oder die Entwicklungsumgebung gelöst werden. In Eclipse müssen beispielsweise

- ~~commons-httpclient-3.1.jar~~ Mittlerweile sollte/muss Apache Http-Components genutzt werden, da die ursprüngliche Apache Bibliothek nicht mehr weiter entwickelt wird.
- commons-beanutils.jar
- commons-codec-1.3.jar
- commons-collections.jar
- commons-lang.jar
- commons-logging-1.1.jar
- json-lib-2.0-jdk15.jar
- ezmorph-1.0.3.jar

in den Projekteinstellungen zu den Libraries hinzugefügt werden. Die nötigen Bibliotheken können hier gesammelt heruntergeladen werden. (Die Lizenzbestimmungen der einzelnen Pakete sind beigefügt und zu beachten)

Server-Connect

Die Verbindung zum laufenden Server wird über ein `Session`-Objekt aufgebaut, das die URL sowie den Port (Standardmäßig kommuniziert couchdb über 5984) übergeben bekommt:

```
Session couchsession = new Session("localhost", 5984);
```

Datenbank-Operationen

Anschließend kann die Verbindung zu einer bestimmten Datenbank aufgebaut werden. Die Operationen auf der Datenbank laufen über ein `Database`-Objekt. Über implementierte Methoden können Anfragen gestellt werden. Über die Methode `adhoc(String js-view)` können direkt Views aus einem JS-String abgerufen werden. Das Ergebnis eines Views wird in einem `ViewResult`-Objekt geliefert, in dem per `getResults()` auf die gefundenen Dokumente zugegriffen werden kann. Hierbei ist es unter Umständen erforderlich per `getDocument(int document-id)` das komplette Dokument aus der Datenbank zu holen.

```
Database couchdb = couchsession.getDatabase("databasename"):
ViewResults results = couchdb.getAllDocuments();
//oder View results = couchdb.adhoc("function (doc) {emit(null, doc);}")
for (Document d : results.getResults()) {
  //...
  Document full = couchdb.getDocument(d.getId());
  //...
}
```

In die Datenbank eingetragen werden können Objekte vom Typ `Document` über die Methode `save(Document doc)`. Den Dokumenten können beliebiege Key-Value-Paare als Datenfelder übergeben werden.

```
Document doc = new Document();
doc.put(KEY_LENGTH "160");
doc.put(KEY_LANG "DE");
//...
couchdb.saveDocument(doc);
```

Java & PDF

Libraries

- Apache PDFBox: Apache Library für die Bearbeitung von PDF-Dokumenten in JAVA. Neben der Volltext-Extraktion ist auch die Suche nach gespeicherten Metadaten interessant. HowTo's finden sich auf der Website.

Metadaten-Extraktion mit PDFBox: Basisdaten

Anlegen des `PDDocument` Object und laden der Dokumentinformationen

```
File pdfFile;
// init pdfFile
PDDocument pdfDocument = PDDocument.load(pdfFile);
PDDocumentInformation pdfInfo = pdfDocument.getDocumentInformation();
```

Aus dem ''PDDocumentInformation'' Object können vorhanden Metainformationen per Getter-Methoden gelesen werden. Unbedingt zu beachten ist hierbei das abfangen von Fehlern (null-Pointer, ...)

```
String pdfTitle = pdfInfo.getTitle();
int pdfPageCount = pdfInfo.getNumberOfPages();
```

Metadaten-Extraktion mit PDFBox: XMP-Daten

Weiter Metadaten können über die Klasse ''PDMetadata'' gewonnen werden. Dabei wird auf Beschreibungen zurückgegriffen, die XMP-strukturiert in den PDF-Dokumenten vorliegen. Neben dem größeren Umfang der Daten ist die XMP-Codierung vorzuziehen, da hier der gesamte Meta-Inhalt geladen und anschließend analysiert werden kann, ohne, wie mit `PDDocumentInformation`, gezielt über wenige Getter-Methoden die Daten auslesen zu müssen.

Das `PDDocument` wird wie gehabt angelegt

```
PDDocument pdfDocument = PDDocument.load(pdfFile);
```

Anschließend kann über PDDocumentCatalog und PDMetadaten ein Stream mit den XML-/XMP-Daten erzeugt werden

```
PDDocumentCatalog catalog = pdfDocument.getDocumentCatalog();
PDMetadata metadata = catalog.getMetadata();
InputStream xmlInputStream = metadata.createInputStream();
```

Dieser Stream kann, beispielsweise, so ausgelesen werden

```
public static String convertStreamToString(InputStream is) {
  try {
    return new java.util.Scanner(is).useDelimiter("\\A").next();
  } catch (java.util.NoSuchElementException e) {
    return "";
  }
}
```

JDBC

Social Network APIs

Twitter-Crawling

Es existieren zahlreiche Möglichkeiten, große Mengen an Tweets abzurufen und zu speichern. In der Regel wird das offizielle API genutzt, wobei direkt oder über entsprechende Bibliotheken mit dem Interface kommuniziert wird. Über dieses kann kontinuierlich auf die öffentlich publizierten Tweets zugegriffen werden. Über `statuses/sample` die public streams wird ein ausreichend großer Ausschnitt aus den Twitter-Echtzeitdaten zugänglich gemacht (1%, ~30Tweets/second). Theoretisch ist ein Abgreifen **aller** Tweets möglich; Zugriff auf `statuses/firehose` zu erhalten ist aber praktisch unmöglich und nur wenigen Firmen erlaubt. Für die meisten Dienste, die über das API angesteuert werden, ist eine Authentifizierung nötig: Twitter Developer Account, registrierte Applikation und deren = Access-Tokens.

Anlegen einer Applikation im Twitter Developer Account

Wenn Sie sich mit einem gültigen Twitter Account angemeldet haben, können Sie unter https://dev.twitter.com/apps neue Applikationen registrieren und Einstellungen und Access Token von bestehenden Anwendungen ändern bzw. einsehen.

Grundsätzliches zum Crawlen von Tweets

Twitter bietet großartige Möglichkeiten schnell und einfach an große Menge strukturierter, von Nutzern generierter Inhalte zu gelangen. Zu beachten sind bei der automatischen Verarbeitung von Tweets aber immer auch die Developer Rules of the Road in denen Twitter festlegt, wer zu welchem Zweck und in welchem Umfang das API nutzen darf, eine kurze Zusammenfassung stellt Twitter hier zur Verfügung. Generell sollte es vermieden werden, Tweets zu crawlen, zu speichern und dann unabhängig vom eigentlichen Twitter-Service (kommerziell) zu veröffentlichen. Über das Online-Interface der Twitter Developer Page besteht die Möglichkeit, die meisten Funktionen des API (des Twitter REST PI) zu nutzen, in dem über das `OAuth tool` ein String generiert wird, der über cURL

genutzt werden kann. Auch damit ist crawlen möglich, für komplexere Anwendungen und die direkte Verarbeitung der gecrawlten Tweets empfiehlt sich die Nutzung einer der Programm-Bibliotheken, die Twitter für verschiedenen Programmier- und Skriptsprachen zugänglich machen. Für Java existiert beispielsweise twitter4j.

Crawlen mit Java & twitter4j Eine rudimentäre Implementierung von twitter4j ist sehr einfach und basiert im wesentlichen auf drei Schritten. Zuerst werden die nötigen Tokens in eine Konfiguration gespeichert, dann wird mit dieser Konfiguration ein `TwitterStream`-Objekt erstellt und schließlich beispielsweise ein `StatusListener` implementiert, der auf alle erfassten Tweets reagiert.

Access/Consumer Token und Keys

```
private static final String CONSUMER_KEY = "1337";
private static final String CONSUMER_SECRET = "1337";
private static final String ACCESS_TOKEN = "1337";
private static final String ACCESS_TOKEN_SECRET= "1337";
```

Speichern der Konfiguration Im `ConfigurationBuilder` werden die speziellen Token und Keys gespeichert um die Application gegenüber des Twitter API eindeutig zu identifizieren. Nur mit gültigen Token ist eine Validierung der Anfragen am Twitter-Server möglich.

```
private StatusListener listener;
private TwitterStream twitterStream;
private ConfigurationBuilder config;

private void setAuth() {
    config = new ConfigurationBuilder();
    config.setDebugEnabled(true);
    config.setOAuthConsumerKey(CONSUMER_KEY);
    config.setOAuthConsumerSecret(CONSUMER_SECRET);
    config.setOAuthAccessToken(ACCESS_TOKEN);
    config.setOAuthAccessTokenSecret(ACCESS_TOKEN_SECRET);
}
```

Implementierung eines möglichen `StatusListener` Ein möglicher `StatusListener` implementiert das Basis-Interface `twitter4j.StatusListener`. Bei den zu implementierenden Methoden handelt es sich um Call-Backs, die zur Laufzeit automatisch aufgerufen werden, wenn beispielsweise ein neuer Tweet empfangen wird (`onStatus(Status status)`). Das `Status`-Objekt kapselt den empfangenen Tweet. Über entsprechende Getter-Methoden können alle relevanten Informationen ausgelesen werden und dann beispielsweise in einer Datenbank gespeichert werden.

```
public class StatusListener implements twitter4j.StatusListener{

    @Override
    public void onException(Exception arg0) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDeletionNotice(StatusDeletionNotice arg0) {
```

```
        // TODO Auto-generated method stub

    }

    @Override
    public void onScrubGeo(long arg0, long arg1) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onStatus(Status status) {
        String tweetText = status.getText());
    }

    @Override
    public void onTrackLimitationNotice(int arg0) {
        // TODO Auto-generated method stub

    }

}
```

Erzeugen des Listeners sowie eines TwitterStream-Objektes und zuweisen der Konfiguration und Verknüpfung von Listener und Stream

```
listener = new StatusListener();
twitterStream = new TwitterStreamFactory(config.build()).getInstance();
twitterStream.addListener(listener);
```