Web-basierte Anwendungen 2: Verteilte Systeme

Sommersemester 2013

Projektdokumentation Phase 2 - DJ System

Dozent:
Prof. Dr. Fischer
Betreuer:
Benjamin Krumnow, Julian Rahe, Volker Schaefer

Aaron Reiher - 11082376
Sascha Rothkopf - 1081813

Inhaltsverzeichnis

1. Projektbeschreibung

1.1. Einleitung

In der zweiten Phase des Workshops des Moduls Webbasierte Anwendungen 2: Verteilte Systeme, geht es um die Konzeption, Implementation und Client-Entwicklung eines Systems welches eine synchrone Kommunikation mit Hilfe eines REST Servers aber auch asynchroner Datenaustausch mit Hilfe von XMPP ermöglicht.
Teil dieser Dokumentation sind Konzeptidee der benötigten XML Schematas, die Implementierung des RESTful Webservices und XMPP Client und der Entwurf von grafischen Clients mit denen das System zu bedienen ist.

1.2. Ideenfindung

Zuerst stellte sich die Frage welches Problem aus der Realität in dem System abgebildet werden soll. Dafür war es wichtig sich alltägliche Kommunikationsabläufe vor Augen zu führen und Probleme zu isolieren die durch ein solches System verbessert werden können. Das System soll eine sinnvolle Ergänzung für den Kommunikationsablauf sein und deren Teilnehmer bestmöglich unterstützen.
Als geeignet erschien die Kommunikation zwischen einem DJ beziehungsweise einer Band und deren Zuhörer in einem Club, Konzert oder Event. Für Zuhörer ist es oft schwierig sich Lieder zu wünschen oder den Verlauf bisher gespielter Lieder einzusehen. Der DJ hat meist nicht die Möglichkeit sich alle Wünsche zu merken. Im später Verlauf des Workshop kam zusätzlich die Idee auf den DJ zu bewerten und kommentieren zu können, die der DJ einsehen kann.

1.3. Der DJ

Der DJ überträgt diverse Meta-Daten des Songs, welcher zum aktuellen Zeitpunkt abgespielt wird. Dies beinhaltet Titel, Interpret, Album, Laufzeit und ähnliche Daten, sowie das Albumcover. Zusätzlich zu den Daten des aktuellen Songs werden auch die Informationen für den nächsten Song angezeigt. Sowohl der aktuelle als auch der nächste Song werden mit den bereits gespielten Songs in einer History abgespeichert werden, sodass die Nutzer auch im nachhinein die Liederliste einsehen können.

1.4. Die Zuhörer

Das Publikum oder der Zuhörer hat die Möglichkeit den DJ zu abonnieren, damit er immer mit den aktuellen Songinfos versorgt ist. Jeder Zuhörer kann mit dem System Musikwünsche äußern, welcher der DJ abrufen kann. Zu diesem Zweck kann der Zuhörer in einer Liederdatenbank des DJs nach vorhandenen Liedern suchen. Je öfter ein gewisses Lied gewünscht wird, desto höher steigt es in der Priorität. Zudem kann er den DJ bewerten und Kritik in Form von Rezensionen äußern. Falls man sich in einem Club oder auf einem Festival befindet, dann kann man mehrere DJs abonnieren.

2. Konzeption

Da im Rahmen des Workshops sowohl synchrone und asynchrone genutzt werden soll, musste abgewogen werden welche Übertragungsart für welche Funktionen genutzt wird.
Zeitkritische Events, wie das wechseln des aktuellen Liedes und somit auch das Ändern der History sollten asynchron realisiert werden. In ersten Überlegungen sollten auch die Wünsche asynchron übertragen werden, so dass der DJ unmittelbar informiert wird. Im späteren Verlauf erschien dies aber als äußerst unklug, da bei vielen Zuhörer schnell auch sehr viele Wünsche entstehen können und so nicht nur das System möglicherweise überlastet wird, sondern auch der DJ, sollte er im schlimmsten Fall jede Sekunde eine Benachrichtigung über einen neuen Wunsch erhalten.
Es war deshalb besser den DJ entscheiden zu lassen wann er die Wünsche abfragt und diese deshalb synchron zu realisieren. Ebenso sollten die Lieder Liste, die Kommentaren synchron realisiert werden. Ebenfalls sollte die History zusätzlich synchron realisiert werden, so dass der DJ bei Bedarf prüfen kann welche Lieder er gespielt hat und die History nicht im Speicher gehalten werden muss. Wird ein neues Lied gespielt und die History dadurch aktualisiert, fragt der DJ die History synchron ab, erweitert sie mit dem aktuellen und nächsten Lied und “publisht” diese, so dass die Zuhörer asynchron informiert werden.

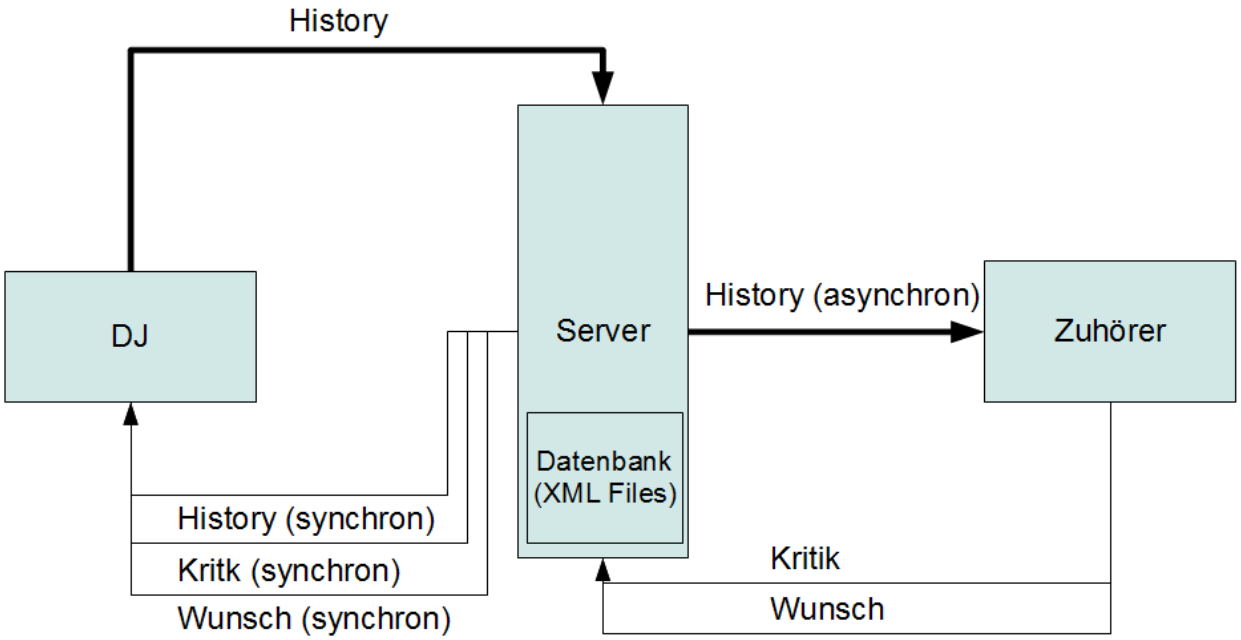


Abbildung 1: Kommunikation DJ - Zuhörer

3. Projektbezogene XML Schemata

Der erste Meilenstein des Workshops, war es ein oder XML Schema / Schemata für das von der Gruppe erdachte Projekt zu gestalten. Da Elemente im Gegensatz zu Attributen später leichter zu ändern sind und keine Struktur abbilden können, wurden ausschließlich Elemente benutzt. Nach reiflicher Überlegung legten man sich auf fünf, später vier, separate xsd-Dateien fest:

3.1. comments.xsd

Die comments Schema-Datei ist für die Kommentare und Kritik zuständig, welche ein Zuhörer/Gast dem DJ hinterlassen kann. Das root-Element von comments ist das comments-Element, welches eine beliebige Anzahl von comment Elementen enthalten kann. Das comment Element beinhaltet all die anderen vorhandenen Elemente.
Das erste referenzierte Element ist „id“, welches genau einmal vorkommen darf. Festgelegt wird dies durch maxOccours=“1“, womit die maximalen Vorkommnisse auf eins gesetzt werden und minOccurs=“1“, wodurch die Mindestanzahl auf eins gesetzt wird.

```
<xs:element ref="id" maxOccurs="1" minOccurs="1"></xs:element>
```

Die ID ist von dem Datentyp “int” und darf nur größer oder gleich 0 sein:

```
<xs:restriction base="xs:int">
<xs:minInclusive value="0"></xs:minInclusive>
 </xs:restriction>
```

Des weiteren ist jede ID einzigartig. Dafür, dass dies gewährleistet ist sorgt ein “unique” im Schema:

```
<xs:unique name="unique-id">
           <xs:selector xpath="comment/id" />
           <xs:field xpath="." />
</xs:unique>
```

Das zweite Element des Sequenz ist der „author“ oder Urheber des Kommentars. Diese Angabe ist freiwillig, da man den Kommentar auch anonym senden kann. Daher darf ein Kommentar maximal einen Autor haben:

```
<xs:element ref="author" maxOccurs="1" minOccurs="0"></xs:element>
```

Es ist vom Datentyp “String”:

```
<xs:element name="author" type="xs:string"></xs:element>
```

Das dritte Element, welches im comment-Element enthalten ist, ist der Zeitstempel „time“. Der Zeitstempel muss bei jedem Kommentar genau ein mal gegeben sein.

```
<xs:element ref="time" maxOccurs="1" minOccurs="1"></xs:element>
```

Ihm ist der Datentyp “dateTime”. Durch den Datentyp sind keine weiteren Unterteilungen in Tag, Monat, Jahr und so weiter notwendig. Er lässt sich später in Java leicht zu einem String formatieren:

```
<xs:element name="time" type="xs:dateTime"></xs:element>
```

Das vierte Element ist das „rating“. Wie auch schon der Autor ist die Bewertung des DJs optional. Da eine Bewertung in dem System nur von 0 bis 10 gehen kann, hat “rating” den Typ “int” mit Einschränkung von 0 bis 10:

```
<xs:restriction base="xs:int">
      <xs:minInclusive value="0"></xs:minInclusive>
      <xs:maxInclusive value="10"></xs:maxInclusive>
</xs:restriction>
```

Das fünfte und letzte Element ist „content“. Ebenfalls ein Pflicht Element und ein simpler “String”, da der Kommentar nicht mehr ist als eine Aneinanderreihung von Characters.

3.2. song_meta.xsd

Die song_meta Schema-Datei war ursprünglich dafür vorgesehen sämtliche relevante Informationen für einen Song zu beinhalten. Das root-Element ist „song“, welches alle anderen, in dem Schema vorhandenen, Elemente verinnerlicht.
Das erste Element ist die “id”. Es sorgt für eine eindeutige Identifikation, wie in “comments” und wird deshalb nicht noch mal hier erwähnt.
Die nächsten drei referenzierten Elemente sind „artist“, „title“ und „length“, welche für den Künstler, den Songtitel und die Länge des Songs stehen. Sie sind Pflichtangaben und es muss genau einer von jedem vorhanden sein. „artist“ und „title“ sind simple “Strings”. “length” bildet die Dauert des Liedes in Sekunden ab.
Die nächsten vier Elemente, „album“, „album_cover“, „album_artist“ und „number_in_album“, beziehen sich alle auf ein potentielles Album. Da ein Song aber nicht unbedingt ein Album haben muss, wurden die “minOccurs” auf 0 und somit als optional gesetzt. Um den Verweis auf eine Bildquelle gut abbilden zu können erhielt album_cover den Datentyp “anyURI”. Das Element “number_in_album” ist wie der Name schon sagt eine Nummer und deshalb vom Datentyp “int” und muss größer oder gleich 1 sein.
Das letzte Element „genre“ ist das Genre des Songs. Dieses Element ist wie Künstler, Titel und Länge ein Pflichteintrag. Ein Lied kann nur ein Genre haben.

Im Verlauf der Workshops fiel auf, dass die Metadaten auch in die Lieder Liste mit einfließen können. So spart man eine XML Datei, ein XML Schema und ein Service des REST Servers. Funktionen die Metadaten über einen Song brauchen benutzen so einfach die Lieder Liste.

3.3. song_list.xsd

Die Schema-Datei „song_list“ sorgt dafür, dass alle vorhandenen Songs in einer Liste gesammelt werden können. Das root-Element ist „songs“, welches eine beliebige Anzahl an „song“ beinhalten kann (aber mindestens einen). „song“ beinhaltet die gleiche Element wie die bereits erwähnte „song_meta“.

3.4. song_history.xsd

Die Schema-Datei „song_history“ ist für den Lieder Verlauf zuständig. Das root-Element ist „history“, welches ein “Nowandnext” und beliebig viele Elemente vom Typ „song“ beinhalten kann. Um Redundanzen zu vermeiden enthält “song” nur die ID des Liedes und die Zeit an der es gespielt wurde (“time_played_at”). Braucht man die Metadaten des Liedes kann anhand der ID in der Lieder Liste nachgegguckt werden.
Das Element “Nowandnext” entstand nach der Überlegung die History leichter zu verwalten. Es enthält genau ein “Now” und genau ein “Next” Element. Diese enthalten wiederum genau ein “song” Element. Das “Nowandnext” Element wurde verwendet um die History leichter zu ändern. Alternativ hätte man jedem Song zwei Attribute geben können die aussagen ob sie der jetziger oder nächste Song sind. Dies würde aber erhebliche Auswirkung auf die Performance haben, da man die komplette History durchsuchen müsste bis ein “song” das gesuchte Attribute hat. In Java lässt sich so das “Nowandnext” Element einfacher adressieren und auch bearbeiten. Ändert der DJ das jetzige und das nächste Lied wird das alte “now” Element als “song” in die History geschrieben und die neuen Lieder in das “Nownext” Element. Das alte “next” Element wird nicht automatisch als neues “now” gesetzt, da es sein könnte, dass der DJ doch ein anderes Lied jetzt spielen möchte als vorher geplant.
Es wurde auf eine ID verzichtet weil die History nur eine Aneinanderreihung von Einträgen ist und dies keine eindeutige Identifikation erfordert.
Da die Datentypen der Elemente bereits vorher behandelt wurden, werden sie hier nicht explizit nochmal aufgeführt.

3.5. wishes.xsd

Die „wishes“ Schema-Datei ist für die Wunschfunktionalität da. Durch diese können sich die Zuhörer/Gäste vom DJ verschiedene Lieder wünschen, welcher sie einer Songdatenbank entnehmen können. Das root-Element ist „wishes“, welches beliebig viele „wish“-Elemente enthalten kann. Das Element „wish“ beinhaltet die restlichen Elemente.
„wish“ enthält drei Elemente, „id“, „song_id“ und „count“, welche alle drei genau einmal vorhanden sein müssen.
Das “id” Element ist wie bei “comments” und “song_list” einzigartig. Das dies bereits in “comments” behandelt wurde, wird hier nicht weiter darauf eingegangen.

„count“ ist wie die vorherigen Elemente Datentyp “int”, jedoch muss es größer oder gleich 1 sein. Count ist der Zähler, welcher für jeden Wunsch für ein bestimmtes Lied die Priorität inkrementiert um dem DJ eine besseren Sortierung nach Beliebtheit zu ermöglichen.

4. Ressourcen und Semantik der HTTP-Operationen

Als nächster Meilenstein stand die theoretische Auseinandersetzung mit REST im Vordergrund. Die Identifizierung von Ressourcen war bereits beim Entwerfen der XML Schemata wichtig.

Bei einer Ressource geht es nicht darum wie Informationen repräsentiert werden, sondern welche Informationen diese enthalten. Objekte der Realität werden beschrieben und stellen einen bestimmten Objekttyp dar. Eine identifizierte Ressource ist eine Schnittstelle zur Realität und sollte daher dem Kontext entsprechend gut durchdacht werden. Der Entwurf der XML Schemata lieferte bereits einen Überblick über vorhandene Ressourcen. Desweiteren gibt es Subressourcen. Diese sind selbst Bestandteil einer Ressource.
Zu den Ressourcen zählen alle Listen: “comments”, “song_list”, “history” und “wishes”. Subressourcen sind untergeordnete Elemente, zum Beispiel ein “comment”, “song” oder “wish”. Da im Rahmen des Workshops nicht nur Path Parameters sondern auch Query Parameters benutzt werden sollen, bot es sich an per Query Paramters die Lieder Liste zu filtern, da ein DJ oft sehr viele Lieder hat und der User nicht manuell suchen will.
Daraus ergaben sich Ressourcen inklusive URI:

Ressource	URI
Liste alle Kommentare	/comments/
Einzelner Kommentar	/comments/{id}
Liste alle Lieder	/songs/
Einzelnes Lied	/songs/{id}
Einzelnes Liede (gefiltert)	/songs?type={artist|title|genre}&text={pattern}
Liste alle Wünsche	/wishes/
Einzelner Wunsch	/wishes/{id}

Nach Identifizierung mussten Operationen bestimmt werden. Dafür stehen vier Methoden GET (Informationen auslesen), POST (Informationen anlegen), PUT (Informationen ändern) und DELETE (Informationen löschen) zur Verfügung.

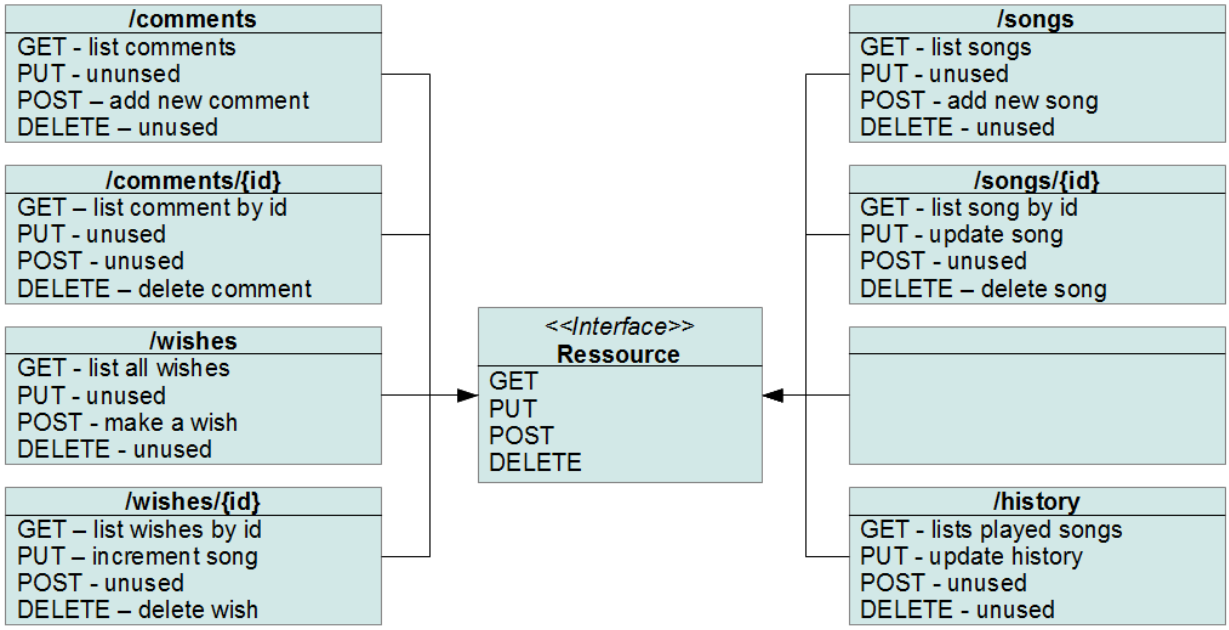


Abbildung 2: Ressourcen und deren HTTP-Operationen

5. RESTful Webservice

Der dritte Meilenstein sah vor einen RESTful Webservice mit Java zur erstellen.
Aus den erstellten XML Schemata wurden automatisiert Java-Klassen erzeugt. Zur Realisierung sollte Grizzly benutzt werden. Zusätzlich wurde zur Entwicklung das Jersey Framework implementiert.

5.1. RESTServer Klasse

Der “RESTServer” war schnell implementiert. Im Rahmen des Workshops beschränkt er sich auf “localhost”. Die start() Methode beinhaltete lediglich:

```
srv = GrizzlyServerFactory.create( url );
```

um den Sever zu starten. Die stop() Methode hält den Server an:

```
srv.stopEndpoint();
```

5.2. Services

Für jede Ressource wurde eine eigene Service Klasse angelegt auf die der RESTServer zugreift um die einzelnen Ressourcen besser verwalten zu können.

Da die Services alle gleich aufgebaut sind wird hier nur auf einen, den “SongService” eingegangen.
Der Pfad der Ressource wird mittels “@” Annotation über der Klasse festgelegt.

```
@Path( "/songs" )
public class SongService
{ … }
```

Eine Methode ohne weitere Pfad Angabe würde so über “http://localhost/songs” erreichbar sein.

5.2.1. GET

Um die Lieder Liste zu erhalten muss man erstens eine “@GET” Annotation hinzufügen, um damit RESTServer weiß, dass er bei einem GET Request diese Methode benutzen soll, und zweitens den Datentyp der Rückgabe per “@Produces” Annotation festlegen.

```
@GET @Produces( "application/xml" )
public Songs songs() throws JAXBException
{ … }
```

Um nun die Lieder Liste zurück zu geben wird ein Unmarshaller verwendet, der aus der benötigten XML Datei liest und die Lieder Liste erstellt. Diese wird anschließend zurück gegeben.

```
ObjectFactory ob = new ObjectFactory();
Songs list = ob.createSongs();
JAXBContext ctx = JAXBContext.newInstance(Songs.class);
Unmarshaller unm = ctx.createUnmarshaller();
list = (Songs) unm.unmarshal(new File(SONG_LIST_XML));

return list;
```

Diese Methode wurde im späteren Verlauf um Query Parameter erweitert. Es bot sich an die Lieder nach Artist, Titel oder Genre zu filtern um eine große und unübersichtliche zu vermeiden.

```
Songs newlist = ob.createSongs();
switch(type.toLowerCase()) {
case "artist":
for (Song s: list.getSong()) {
          if
(s.getArtist().toLowerCase().startsWith(text.toLowerCase())) {
               newlist.getSong().add(s);
          }
}
return newlist;
case "title":
     for (Song s: list.getSong()) {
          if
(s.getTitle().toLowerCase().startsWith(text.toLowerCase())) {
               newlist.getSong().add(s);
          }
     }
     return newlist;
case "genre":
     for (Song s: list.getSong()) {
          if
(s.getGenre().toLowerCase().startsWith(text.toLowerCase())) {
               newlist.getSong().add(s);
          }
     }
     return newlist;
}
return list;
```

Die Liste alle Lieder wird nach den Query Parametern durchsucht. Sollte keins oder ein falsches Angegeben sein, wird die komplette Liste zurück gegeben.

Subressourcen, wie zum Beispiel ein bestimmtes Lied wurden mittels Path Parameter realisiert. Als Alternative standen Query Parameter zu Verfügung aber da diese meist nur zur Verfeinerung eingesetzt und meist optional sind wurden Path Parameter benutzt.
Um das richtige Lied mit der dazugehörigen ID zu finden, wird die Liste komplett durchlaufen bis die richtige ID gefunden wurde ansonsten wird eine leere Liste zurückgegeben.

```
@Path( "/{ID}" )
@GET @Produces( "application/xml" )
public Song songbyid(@PathParam("ID") int id) throws JAXBException
{
	ObjectFactory ob = new ObjectFactory();
	Songs list = ob.createSongs();
	JAXBContext ctx = JAXBContext.newInstance(Songs.class);
	Unmarshaller unm = ctx.createUnmarshaller();
	list = (Songs) unm.unmarshal(new File(SONG_LIST_XML));
	Song newlist = ob.createSong();

	for (Song s: list.getSong()) {
		if (s.getId() == id) {
			newlist = s;
		}
	}
	return newlist;
}
```

5.2.2. DELETE

Ein DELETE Request wird ähnlich behandelt jedoch anstatt der “@GET” Annotation eine “@DELETE” Annotation. Es wird keine “@Produces” Annotation gebraucht da nur gelöscht und nichts zurückgegeben wird. Sonst funktioniert die Methode ähnlich wie die Methode für GET Request außer dass alle Lieder zu “newlist” hinzugefügt werden, nur der mit der übergebenen ID nicht.

```
if (s.getId() != id) {
	newlist.getSong().add(s);
}
```

Danach wird ein Marshaller benutzt um die Änderungen wieder in die XML Datei zu schreiben.

```
Marshaller marshaller = ctx.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
```

```
marshaller.marshal(list, (new File(SONG_LIST_XML)));
```

5.2.3. POST

Die POST Methode funktioniert wie die GET Methode. Sie braucht eine “@POST” Annotation und eine “@Consumes” Annotation damit der RESTServer weiß welchen Dateityp er akzeptieren soll. Um eine fehlerhafte ID zu vermeiden wird die ID im mitgegebenen Song überschrieben und mit der letzten ID + 1 ersetzt.

```
int i = list.getSong().size() - 1;
list.getSong().add(s);
list.getSong().get(i+1).setId(list.getSong().get(i).getId()+1);
```

5.2.4. PUT

Die Methode für PUT Request ist ebenfalls ähnlich aufgebaut jedoch braucht sie eine “@PUT” Annotation. Ist die ID im Path Parameter identisch wird der alte Song mit dem neuen ersetzt.

```
if (s.getId() == id) {
      list.getSong().set(id, song);
}
```

Da die Services für Kommentare, Wünsche und Verlauf annähernd gleich aufgebaut sind wird auf eine weitere Erklärung verzichtet.

6. Konzeption asynchrone Kommunikation

Bei der asynchronen Übertragung für das DJ-System handelt es sich um das Lied, welches momentan am Laufen ist und das Lied, welches als nächstes kommt.

Die Leaf Nodes wurden für das DJ-System als die Djs selbst definiert. Man kann jeden DJ abonnieren und auch wieder abbestellen. Es war geplant, dass man auch Collection Nodes subscriben kann um automatisch alle damit verbundenen Djs zu abonnieren, jedoch wurde diese Funktion von der API noch nicht unterstützt und wurde somit nicht zu einer Priorität.

Der DJ nimmt mit seiner Version des Clients die Rolle des Publishers ein. Die Benutzer haben mit ihrem Client die Möglichkeit bei einen DJ zu abonnieren und sind somit die Subscriber.

Bei der asynchronen Übertragung wird die Liederhistory übertragen in Form der Lieder IDs. Der Client muss dann mit Hilfe der IDs synchron nach den Metadaten fragen. Da dies aber im Anschluss auf eine erfolgte asynchrone Übertragung geschieht ist dazu keine Nutzer Interaktion erforderlich.

Die Erstellung bzw. Einrichtung des XMPP Servers wird hier nicht näher erläutert, da es lediglich eine simple Installation war. Der XMPP Server ist auf dem virtuellen Server

v2201206130288594.yourvserver.net:5222

zu finden. Bisher wurden nur zwei User zu Testzwecken angelegt.

7. XMPP - Client

Sobald sich ein DJ einloggt und damit signalisiert, dass er jetzt die Musik auflegt, wird für ihn eine Leaf Node erstellt. Alternativ kann der DJ ein Event, Stage oder Club auswählen auf dem er ist, dies wurde aber bei der Implementierung außen vor gelassen.
Ein NodeHandler sorgt für die “connect”, “disconnect”, “createNode” und “delteNode” Methoden. Er verbindet sich mit der Server IP und dem Port, loggt anschließen den Benutzer ein und erstellt einen PubSubManager der für das erstellen und löschen der Nodes zuständig ist:

```
con.connect();
con.login(name, pw, res);
mgr = new PubSubManager(con);
```

Eine Node braucht ein “CofigureForm” mit der sie konfiguriert wird. Dort werden Werte gesetzt über die Art der Node (leaf oder collection) und ob sie Nutzdaten überträgt. Danach wird die Node aufgerufen und zurückgegeben:

```
ConfigureForm form = new ConfigureForm(FormType.submit);
form.setAccessModel(AccessModel.open);
form.setDeliverPayloads(true);
form.setPersistentItems(false);
form.setPublishModel(PublishModel.open);
form.setNodeType(NodeType.leaf);
form.setTitle(name);
return (LeafNode) mgr.createNode(name, form);
```

Diese Node wird weiter gegeben um anschließend über sie den Verlauf zu publishen. Es werden Nutzdaten übertragen damit es möglich ist die komplette History zu übertragen, damit der Client diese nicht synchron nochmal abrufen muss.
Um die History zu übertragen wird ein Marshaller benutzt um das History Objekt zu bekommen und es in einen StringWriter zu schreiben. Dies wird in ein SimplePayload gepackt und anschließend in ein PayloadItem um es anschließen zu übertragen.

```
Marshaller marshaller = ctx.createMarshaller();
marshaller.marshal(h, writer);

SimplePayload payload = new SimplePayload(id,"history",
writer.toString());
PayloadItem<SimplePayload> item = new PayloadItem<SimplePayload>(id,
payload);
node.publish(item);
```

In unserem Benutzer-Client werden die Leaf Nodes über die Discovery Items des PubSub Managers erhalten. So wird wird der PubSub Manager durch die Übergabe der Connection erstellt.

```
PubSubManager mgr = new PubSubManager(con);
```

Die Discovery Items werden aus dem PubSub Manager übergeben und in einen Iterator eingespeist.

```
DiscoveryItems discoItems = mgr.discoverNodes(null);
Iterator<Item> it = discoItems.getItems();
```

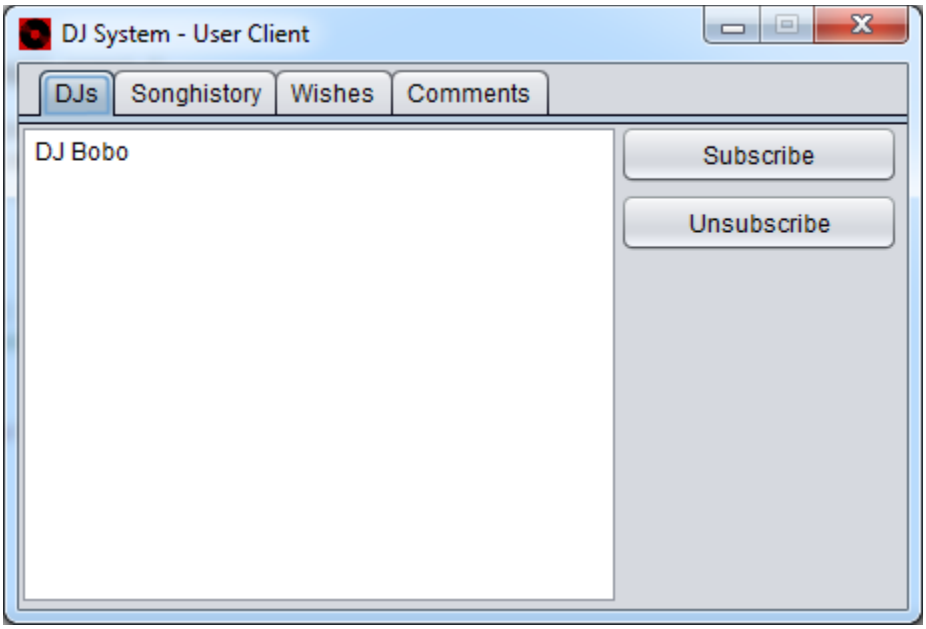
Die Nodes werden schließlich nur noch des einfachen Zugriffs wegen in einen Array gespeichert.

```
Item[] list = new Item[50];
while(it.hasNext()) {
	Integer i = 0;
	Item i1 = it.next();
	list[i++] = i1;
}
```

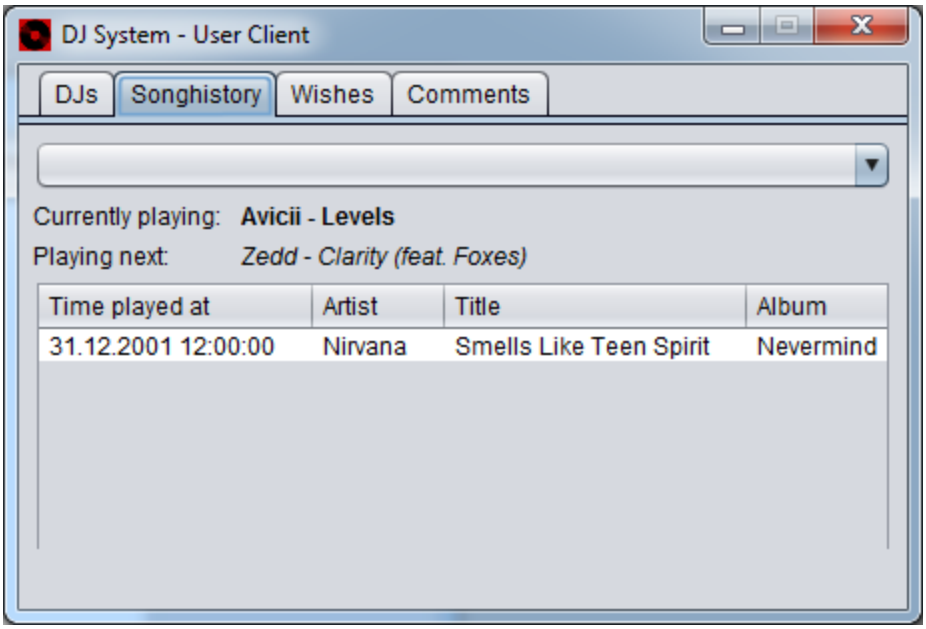
8. Client-Entwicklung

8.1. Der User-Client

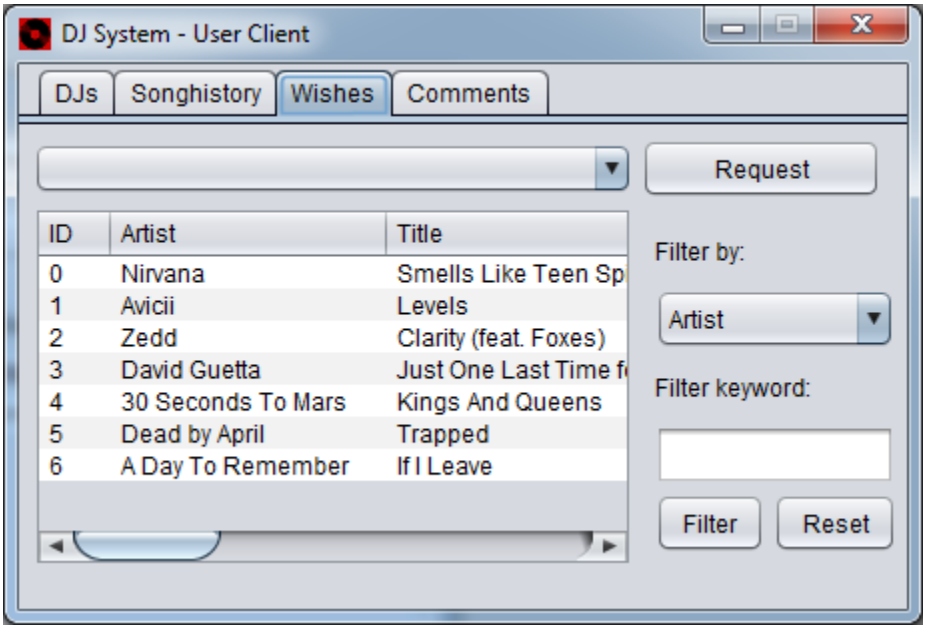
Das Graphical User Interface des User-Clients ist durch ein Tab-System aufgebaut. Dieses sorgt dafür, dass das Interface nicht zu überladen ist. Zudem erlaubt es eine einfache Übersicht und ebenso einfache Navigation durch das Programm. Es werden vier Tabs verwendet: „Djs“, „Songs“, „Wishes“ und „Comments“.



Der „Djs“-Tab hat ein simples Layout mit drei Komponenten. Links ausgerichtet und zwei Drittel des Fensters einnehmend ist eine JList, welche die vorhandenen Djs anzeigt. Links davon, im verbleibenden Drittel, sind zwei JButtons untereinander angeordnet. Der obere ist „Subscribe“ und der untere „Unsubscribe“ und sind jeweils für das abonnieren und abbestellen des momentan selektierten Djs zuständig.
Da wir in den westlichen Kulturen von links nach rechts lesen, so wurde auch die Liste links angeordnet, da man jene zuerst sehen soll. Hat der Benutzer die Liste erkannt und einen DJ ausgesucht, so wird er nun seinen Fokus weiter nach rechts verlagern und dort die Buttons aktiv wahrnehmen. Jetzt kann er sich entscheiden, was für eine Aktion er mit dem markierten DJ ausführen will.

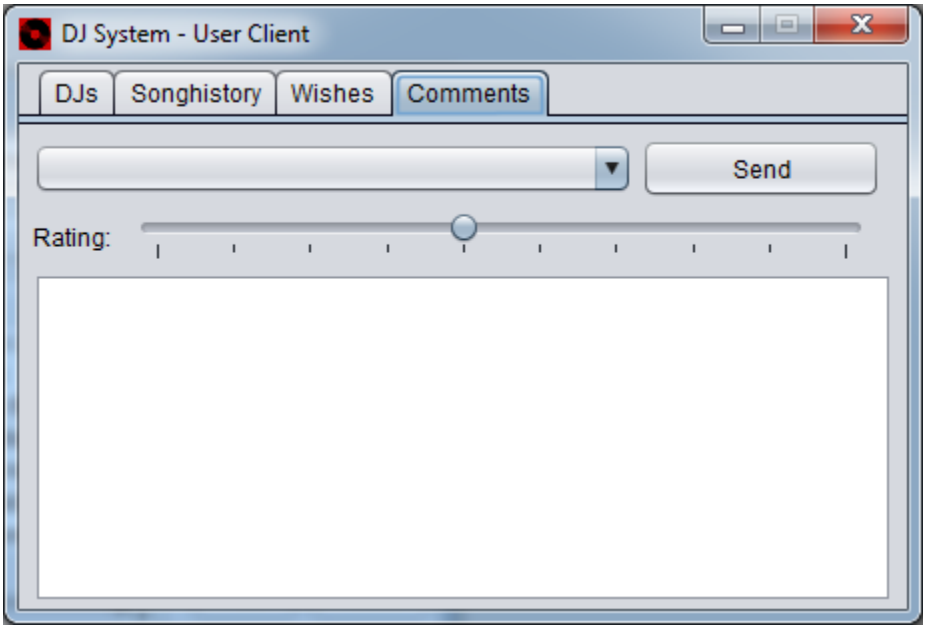


Der „Songs“-Tab ist ein wenig komplexer als der „Djs“-Tab und arbeitet wieder mit drei, Komponenten. Am oberen Rand befindet sich ein Label, welches anzeigt welches Lied momentan abgespielt wird. Darunter befindet sich eine Combobox, welche eine Auswahl zwischen den verschiedenen Djs, die man abonniert hat, erlaubt. Unterhalb der Combobox befindet sich eine Liste, welche alle bereits abgespielten Lieder anzeigt. Die Liste zeigt zudem sämtliche relevanten Metadaten an und lässt sich nach jenen sortieren.



Der „Wishes“-Tab ist ähnlich simpel wie der „Djs“-Tab. Er besteht aus sieben Komponenten. Im oberen Teil des Panels befinden sich eine Combobox zur Auswahl des Djs, welcher den

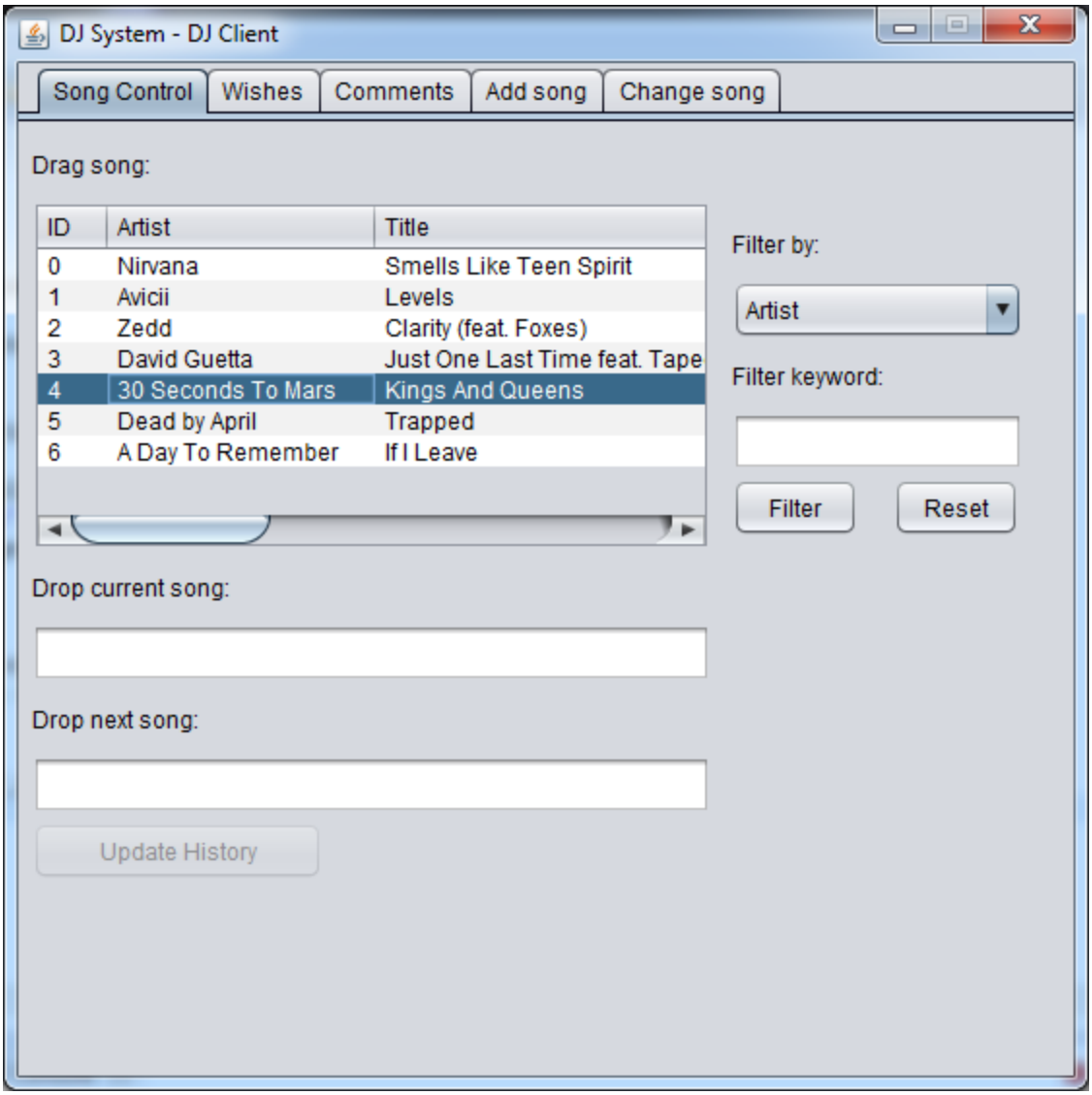
Wunsch erhalten soll, sowie den „Request“ Button um den Wunsch abzusenden. Darunter befindet sich eine Liste, welche die komplette Liederliste anzeigt, samt allen relevanten Metadaten. Rechts davon befindet sich die Kontrolle der Filteroption.



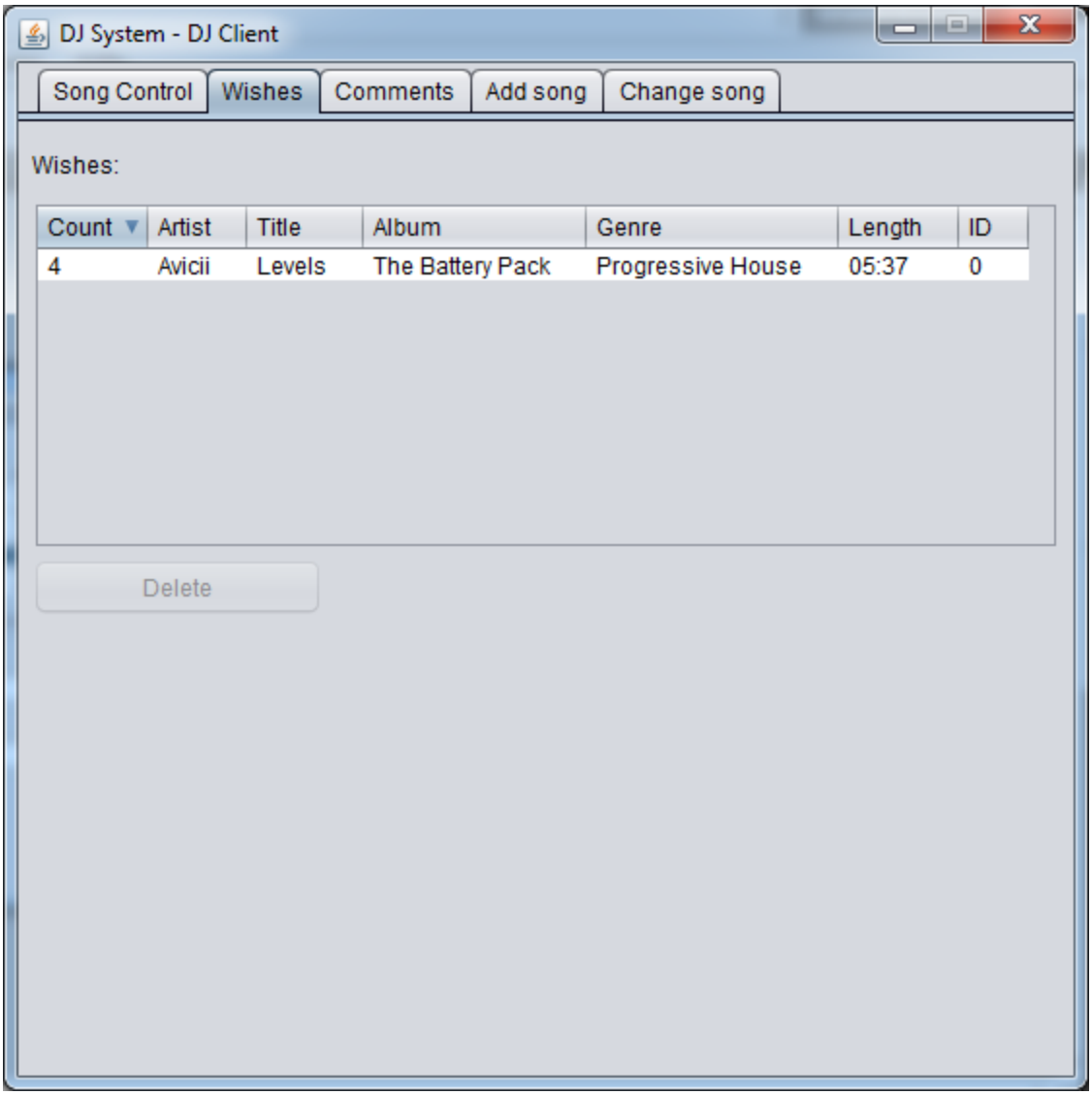
Der „Comments“-Tab besteht aus vier Komponenten, welche untereinander angeordnet sind. Die oberste Reihe beinhaltet eine Combobox, welche zur Auswahl eines abonnierten Djs da ist, sowie einen „Send“ Button, welcher die fertige Nachricht an den DJ sendet. Darunter befindet sich ein Slider, mit welchen man ein Rating abgeben kann, welches von eins bis zehn reicht. Ganz unten befindet sich schließlich eine Textarea in welcher der Kommentar dann geschrieben wird.

8.2. Der DJ-Client

Wie auch schon der User-Client ist der DJ-Client durch ein Tab-System organisiert. Der DJ-Client hat im Gegensatz zum User-Client fünf verschiedene Tabs und ist auch von den Maßen her wesentlich größer, was an der größeren Funktionsvielfalt liegt.

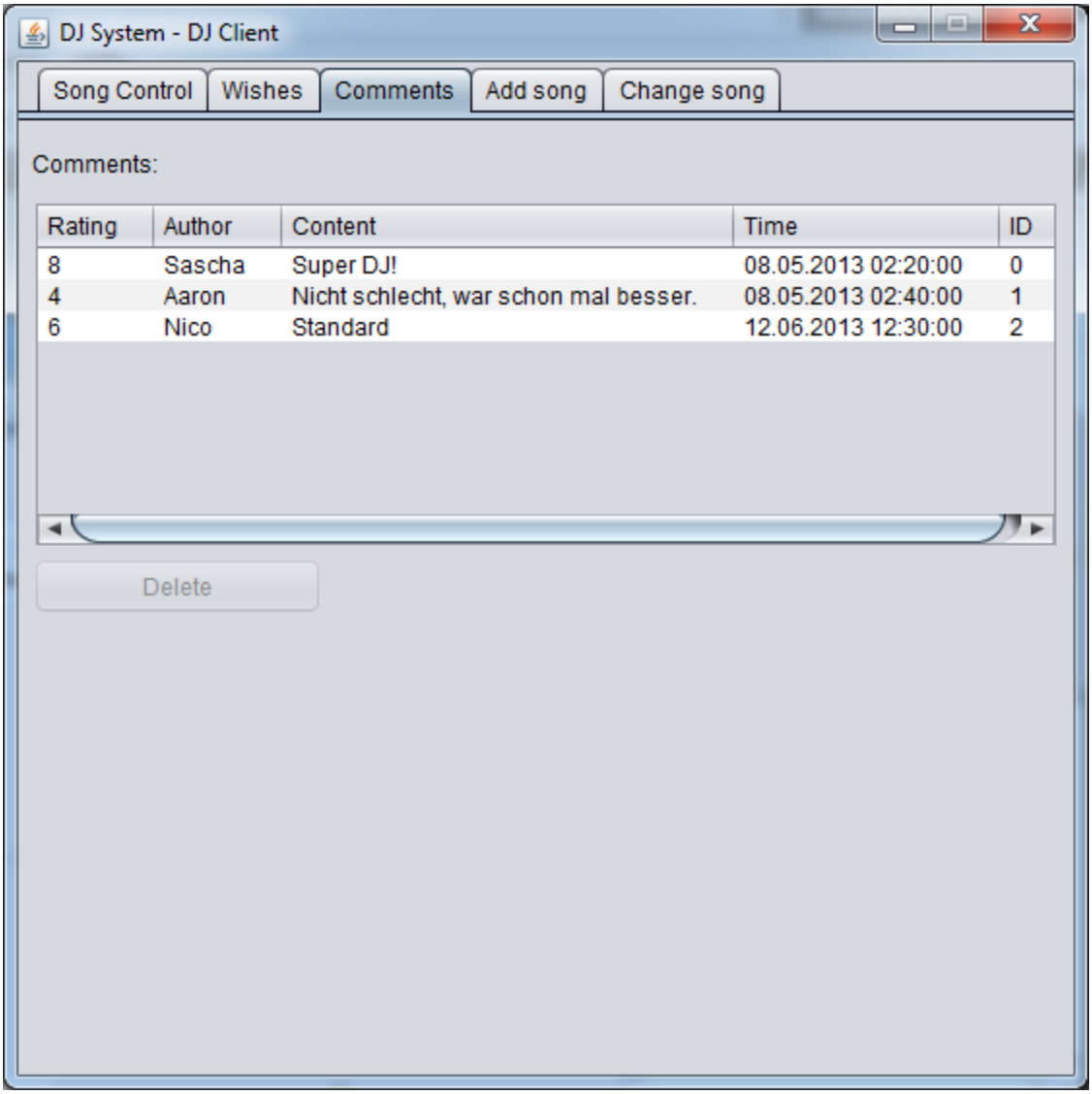


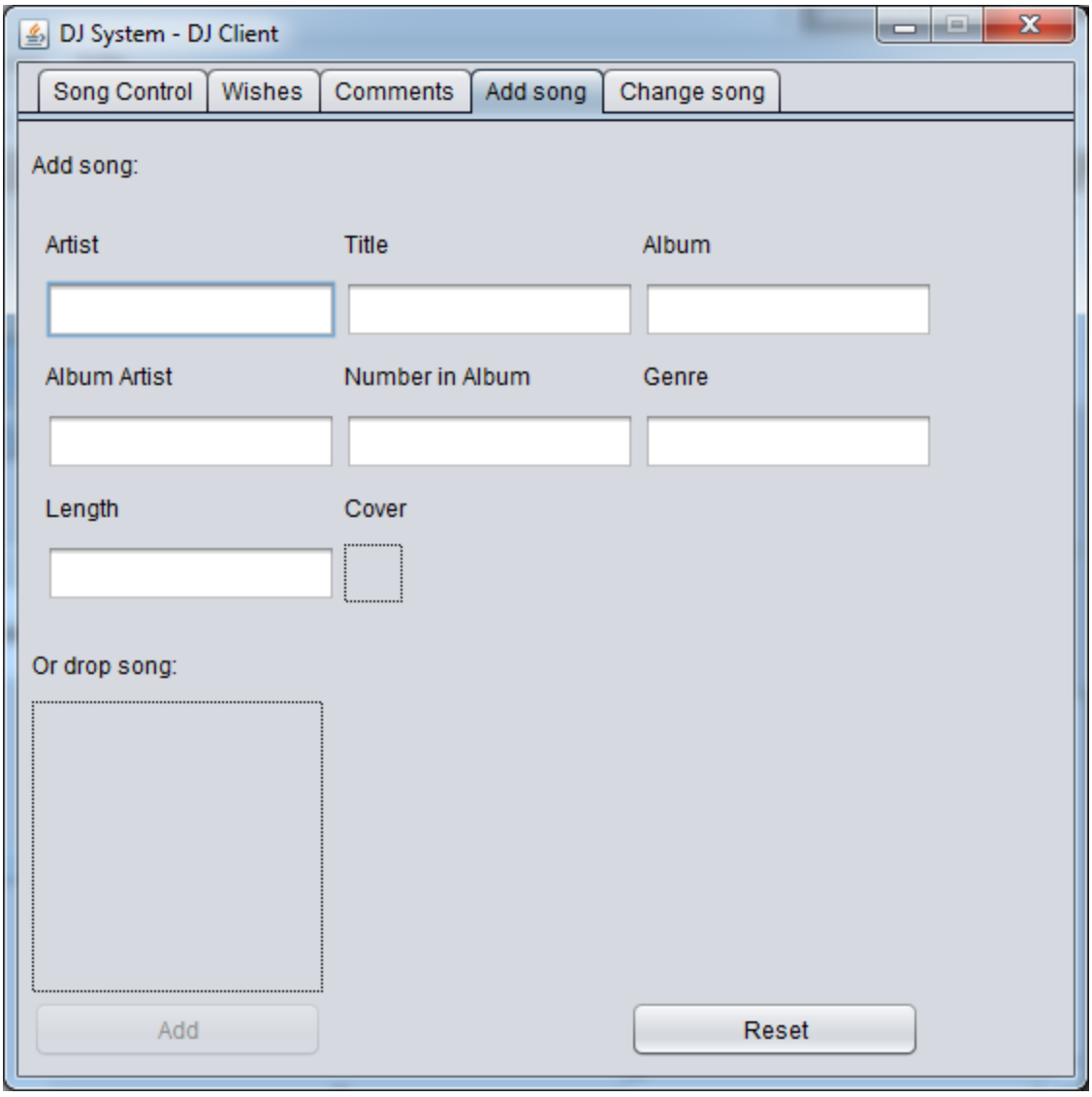
Der erste Tab ist “Song Control”. Dieser Tab ist für das Publishen von den aktuellen Songs zuständig. Am oberen Rand befindet sich eine Tabelle, welche sämtliche Lieder des DJs inklusive Metadaten enthält. Rechts davon befindet sich die Kontrolle für den Filter, welcher bestimmte Elemente aus der Tabelle heraussuchen kann.
Unter der Tabelle befinden sich zwei untereinanderliegende Textfelder, betitelt mit dem Text “Drop current song” und “Drop next song”. In diese kann der DJ aus der Tabelle per Drag and Drop einen Song hineinziehen. Sind die Felder je mit einem Song versehen, so wird unterhalb der Felder ein Button namens “Update History” freigeschaltet. Drückt der DJ auf diesen Button, so wird die History veröffentlicht, und der User-Client wird benachrichtigt.



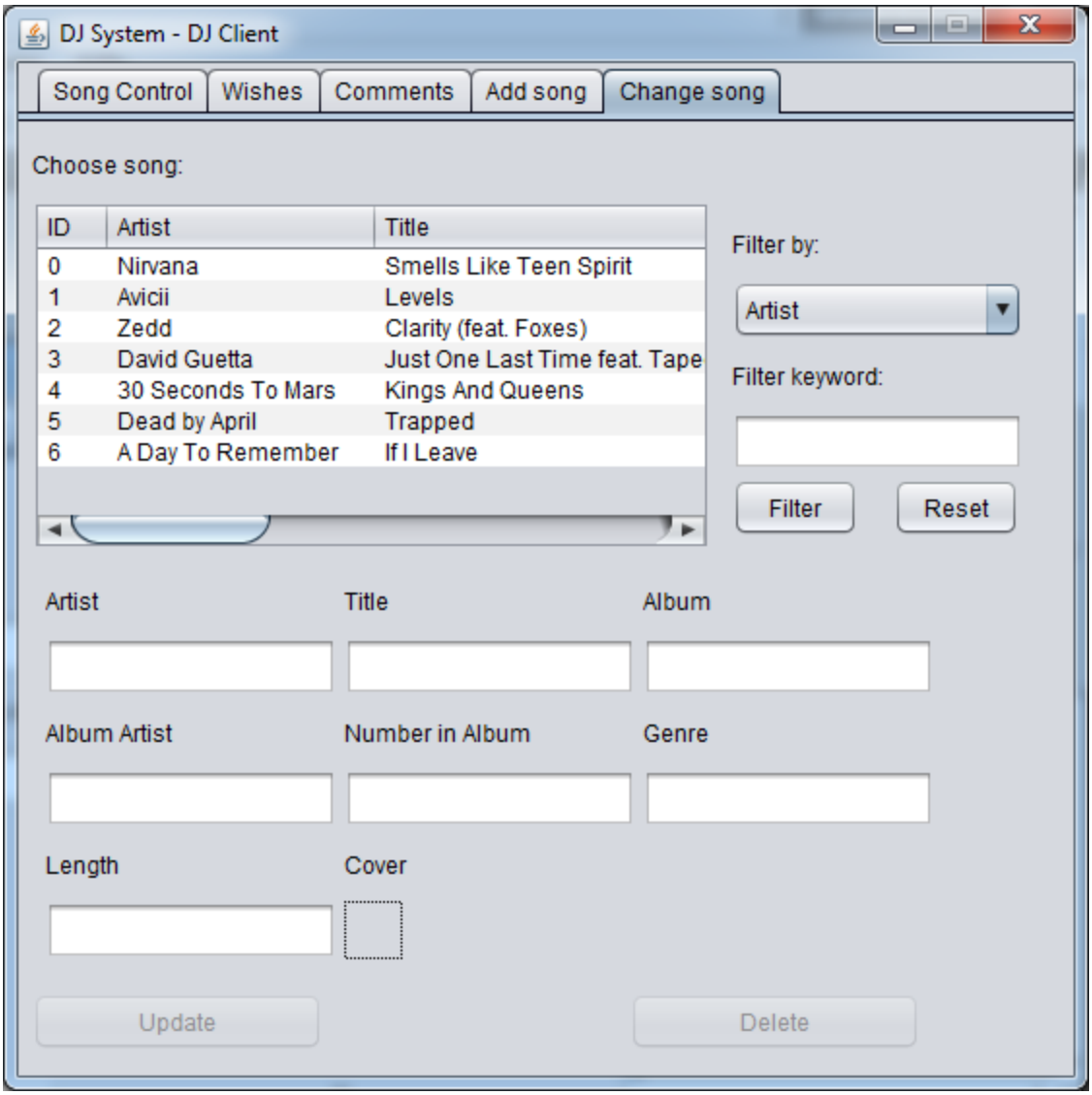
Wie auch der User-Client hat der DJ-Client einen “Wishes”-Tab. Dieser ist hier dazu da die geäußerten Wünsche für den DJ sichtbar zu machen. Die Tabelle, welche die Wünsche enthält ist oben angeordnet. Der DJ hat die Möglichkeit die Tabelle, unter anderem, nach der Anzahl wie oft der Wunsch geäußert wurde zu sortieren, damit er Prioritäten für seine Liederwahl setzen kann. Unter der Tabelle befindet sich ein “Delete”-Button. Mit diesem kann der DJ einen Wunsch löschen, wenn er z.B. ausgeführt oder an dem Abend schon zu oft gespielt wurde. Auch ein “Comments”-Tab ist im DJ-CLient vorhanden und wieder dient es dazu die Kommunikation mit den Zuhörern für den DJ visuell darzustellen. Vom Aufbau ist der “Comments”-Tab mit dem “Wishes”-Tab identisch. Die Tabelle dient diesmal zum Aufzeigen der von den Nutzern eingereichten Kommentaren und Wertungen. Der DJ hat wie bei den anderen Tabellen die Möglichkeit jene zu sortieren. Die Sortierung nach dem Zeitstempel funktioniert nicht ganz korrekt aus uns unbekannten Gründen. Wir haben probiert den Fehler zu korrigieren, es ist uns jedoch nicht gelungen.

Unterhalb der Tabelle befindet sich wieder ein “Delete”-Button. Mit diesem kann der DJ einen Kommentar löschen.





Der vierte Tab ist der “Add Song”-Tab, mit welchem der DJ neue Lieder zu seinem Repartoir hinzufügen kann. Sieben Textfelder stehen dem DJ zur Verfügung, mit welchem er die Metadaten für ein Lied eintragen kann. Sind alle Felder ausgefüllt, so kann der DJ auf den am unteren Rand positionierten “Add”-Button klicken und somit das Lied zu seiner Liste an Liedern hinzufügen. Alternativ kann der DJ auch eine mp3-Datei per Drag and Drop auf auf das quadratische Feld unterhalb der Textfelder ziehen, wodurch die Metadaten des Titels automatisch eingetragen werden. Dies kann die Arbeit des DJs extrem vereinfachen. In der unteren rechten Ecke findet sich noch ein “Reset”-Button, welcher alle Textfelder leert.



Der letzte Tab des DJ-Clients ist der “Change Song”-Tab. Dieser Tab erlaubt es dem DJ die Metadaten eines Songs gezielt zu bearbeiten. Wieder einmal findet sich am oberen Rand des Panels eine Tabelle, in welcher alle Lieder des DJs aufgelistet sind. Rechts davon befindet sich die Steuerung für den Filter, mit welchem sich einfach Lieder finden lassen.
Wählt man einen Song aus der Tabelle aus, so werden die Metadaten in die darunter gelegenen Textfelder übertragen. In diesen kann man die Metadaten dann bearbeiten. Ist man mit den Änderungen zufrieden, drückt man auf den Button “Update”, welcher den Song aktualisiert. Alternativ kann man ein Song auswählen und den “Delete”-Button drücken. In diesem Fall wird der Song gelöscht.

Um die mp3 Tags aus einer Datei auszulesen wurde die OpenSource Bibliothek “mp3agic” (https://github.com/mpatric/mp3agic) benutzt.
Um eine Datei zu öffnen benutzt man “new Mp3File()”:

```
Mp3File mp3 = new Mp3File(file.getAbsolutePath());
```

Danach sollte man prüfen ob die Datei einen Tag der Version 1 oder 2 hat und ihn entsprechend behandeln:

```
if (mp3.hasId3v1Tag()) {
      ...
} else if (mp3.hasId3v2Tag()) {
      …
}
```

Ein Tag kann dann mittels “mp3.getId3v1Tag” beziehungsweise “mp3.getId3v2Tag” addressiert werden. Um einzelne Werte auszulesen werden “get” Methoden auf dem Tag Objekt benutzt:

```
ID3v2 id3v2Tag = mp3.getId3v2Tag();
info_panel.txt_artist.setText(id3v2Tag.getArtist());
info_panel.txt_title.setText(id3v2Tag.getTitle());
```

Für das Layout haben wir den MiG Layout Manager (http://www.miglayout.com/) verwendet. Mit dem Layout Manager lassen sich sowohl feste als auch flexible Layouts recht schnell erstellen.
Das Layout wird als Objekt erstellt und dann einfach einem JPanel zugeordnet:

```
MigLayout layout = new MigLayout();
JPanel panel_list = new JPanel(layout);
```

Dem JPanel kann man dann andere Komponenten hinzufügen und mit den Constraints des Mig Layout Managers versehen.

```
panel_list.add(new JScrollPane(list_DJs), "dock west, width 300!,
height 240!");
panel_list.add(btn_subscribeDJ, "width 140!,height 30!, wrap");
panel_list.add(btn_unsubscribeDJ, "width 140!,height 30!, wrap");
```

So kann man mit width und height die Maße eines Objekts festlegen und mit wrap einen Zeilenumbruch erzeugen. Weitere Parameter find sich in der Dokumentation von MiG.

8.3. Subscriber-Klasse

Die Subscriber-Klasse ist die Klasse, welche für den User-Client die Verbindung zum REST Server herstellt und die Informationen bezüglich der Djs extrahiert.

Wenn ein Subscriber-Objekt erstellt wir und der Konstruktor aufgerufen wird, so wird die Funktion connect() aufgerufen, welche den User mit dem Server verbindet. Dies geschieht über XMPPConnection, von welcher die Funktion connect() aufgerufen wird. Die Funktion des Connection-Objekt stellt die eigentliche Verbindung zum Server her. Durch den darauf folgenden Funktionsaufruf login(), welchem die Parameter Nutzename, Passwort und System mitgegeben werden, findet eine Authentifizierung statt, welche, falls erfolgreich, den Benutzer auf dem Server einloggt.

Von der Connection wird nun ein PubSubManager (Publish Subscribe Manager) erstellt. Einem DiscoverItem-Objekt wird nun mithilfe des PubSubManagers initialisiert durch den Funktionsaufruf discoverNodes(). Diese Node-Items werden nun durch die Funktion getItems() in einen Iterator übertragen. Bei den Nodes handelt es sich um die verschiedenen vorhandenen Djs, welche auf dem Server eingetragen sind. Jener wird nun in einer Schleife durchlaufen und jeder gefundene Knotenpunkt wird in eine Liste zur späteren verwendung übertragen. Zur gleichen Zeit werden die gefundenen Nodes an das Graphical User Interface übertragen.

Die öffentliche Funktion subscribe() ist dazu da um gewisse Knoten, also Djs, zu abonnieren. Der Funktion wird der Name des Knoten mitgegeben, welchen man abonnieren will. Zuerst wird die gewünschte Node aus dem PubSubManager herausgesucht mittels der Funktion getNode(). Danach wird geprüft ob man die gewünschten Node bereits abonniert hat, indem die Node mit denen in der Liste der Subscription verglichen wird. Ist dies nicht der Fall, dann wird der gewählten Node ein Item Even Listener erteilt, damit die Aktionen des Djs überwacht werden können und schließlich wird die Node durch die Funktion subscribe(), welcher die User ID, sowie die Serveradresse übergeben werden, abonniert.

Als Gegenstück der Funktion subscribe() gibt es die Funktion unsubscribe(), welche, wie der Name bereits andeutet, dazu da ist von einer ausgewählten Node das Abonnement zu beenden. Wie auch schon bei subscribe() wird die Node, von welcher man unsubscriben möchte, über ihren Namen ermittelt. Nun wird jene Node in der Liste der abonnements mittels einer Schleife gesucht. Solbald die Node gefunden wurde wird der vorher vergebene Item Event Listener hier nun entfernt. Mittels der Funktion unsubscribe(), welche als Parameter den Nutzername, die Serveradresse und die Node-Id erhält, wird nun das Abonnement beendet. Schließlich wird noch mit der Funktion remove(index) die Node aus der Liste der Subscriptions entfernt.

Die letzte Funktion in der Klasse Subscriber ist die Funktion getSubs(), welche einen String-Array zurückgibt. Der Nutzen der Funktion ist, dass die Subscription-Liste aktualisiert wird und zudem, dass der Name aller Subscriptions für andere Funktionen zurückgegeben wird.

Zuerst wird über den PubSubManager, mithilfe der Funktion getSubscriptions(), die Liste mit allen Subscriptions übergeben und die alte Liste wird, falls vorhanden, überschrieben. Die Namen dieser Abonnements werden draufhin in einer Schleife in einen String-Array eingespeist. Wenn dies erfolgreich ablief wird der String-Array zurückgegeben.

9. Fazit

Im Rahmen des Workshops und aufgrund des nur begrenzten Zeitraums entschied sich die Gruppe dafür den REST Server lokal auf dem PC des DJs laufen zu lassen. Dort werden auch vorerst die XML Dateien gespeichert.
In der Realität würde es so aussehen, dass der DJ mit seinem Laptop ein eigenen Access Point erstellt und die Zuhörer mit ihrem Smartphone oder Laptop sich darauf verbinden. So braucht keiner der Zuhörer einen Internet Zugang sondern ist direkt mit dem DJ verbunden. Dies spart nicht nur Traffic sondern auch Kosten für einen Server.
Aufgrund der Komplexität des Openfire XMPP Servers war es nicht möglich diesen in den DJ Clienten einzubinden, deshalb wird, wenn man asynchrone Live Updates zum Liederverlauf möchte, doch ein Internet Zugang vorausgesetzt.
Möchte man das Projekt außerhalb des Workshops erweitern und mehrere Events, Stages, Clubs und mehrere DJs unterstützen muss REST, XMPP Server und XML Dateien auf einen externen Server ausgelagert werden. Ändern müsste man dafür nur die IP zu denen sich DJ und Client verbinden, also nur 2 Variabeln im Code.
Alles in allem haben wir einen Großteil unserer Ziele erfolgreich umsetzen können und sogar noch ein paar interessante Features während der Entwicklung entdeckt (wie z.B. die MP3 Drag and Drop Bibliothek).