

PhotoBay

WBA 2 – Dokumentation

Gruppe:

Roy Fochtman (11084428)

David Wachs (11081707)

ZIELSETZUNG	3
KOMMUNIKATIONSABLÄUFE UND INTERAKTION	3
KOMMUNIKATIONSARTEN	3
<i>Synchroner Datenaustausch</i>	4
<i>Asynchroner Datenaustausch</i>	4
INFORMATIONSLUSSDIAGRAMM	4
PROJEKTBEZOGENES XML SCHEMA	5
RESSOURCEN UND DIE SEMANTIK DER HTTP-OPERATIONEN FÜR DAS PROJEKT	6
PHOTOGRAPHERS	6
PRESSAGENCIES	7
JOBS	7
PHOTOSSELLS.....	7
CRUD-MATRIX.....	8
RESTFUL WEBSERVICE	8
RESSOURCEN MIT EINDEUTIGER IDENTIFIKATION	8
HYPERMEDIA	8
STANDARDMETHODEN.....	9
REPRÄSENTATIONEN.....	9
WEBSERVICE.....	10
KONZEPTION PUBLISH/SUBSCRIBE	11
XMPP-CLIENT	12
ABGRENZUNGEN	12
PROJEKT-STRUKTUR	13
BENUTZUNG DES CLIENTS	14
LOGIN	14
REGISTRIERUNG.....	14
CLIENT-FENSTER (FOTOGRAF)	15
CLIENT-FENSTER (PRESSEAGENTUR)	15
QUELLEN	16

Zielsetzung

Ziel des Projektes ist die Realisierung einer Kollaborationsanwendung für Fotografen (Photographers) und Presseagenturen (PressAgencies). Primär ist es auf der Seite der Fotografen möglich, Fotos zum Verkauf bzw. zur Versteigerung für Presseagenturen anzubieten.

Auf der Seite der Presseagenturen ist es möglich Aufträge für Fotografen zu erstellen und zu veröffentlichen.

Das Prinzip der Fotoverkäufe (PhotoSells) ähnelt dem der Verkaufsplattform Ebay. Das bedeutet, dass nach einer bestimmten Zeit die Presseagentur mit dem höchsten Gebot (Bids) das Foto ersteigert.

Bei den Aufträgen (Jobs) können sich Fotografen mit einem kurzen Text und, wenn vorhanden, einigen Beispielfotos bewerben (JobApplications). Die Presseagenturen können dann zwischen den Bewerbern entscheiden und einem oder mehreren Bewerbern den Auftrag erteilen. Anschließend wird die Bezahlung des Auftrages ausgehandelt.

Aufträge haben auch eine Zeitliche Begrenzung. D.h. dass sie bis zu einer bestimmten „Deadline“ abgeschlossen sein müssen.

Sowohl für Pressagenturen als auch für Fotografen ist es möglich die Profile andere Benutzer zu besuchen. Z.B. kann eine Presseagentur sich das Profil eines Fotografen anschauen, der sich für einen Auftrag beworben hat, um festzustellen, ob dieser den Anforderungen genügt.

Fotografen sollen nicht nur Fotos zum Verkauf anbieten sondern auch Fotoalben in ihrem Profil präsentieren. Das gleiche gilt auch für Presseagenturen. So sollen beide Seiten in der Lage sein festzustellen, welche Art von Bildern z.B. eine Presseagentur erwartet oder welche Qualität die Fotos eines Fotografen haben.

Außerdem können Fotografen und Presseagenturen sich gegenseitig bewerten. Dabei können Fotografen aber nur Presseagenturen bewerten, für welche sie Aufträge abgeschlossen haben und Presseagenturen können nur die Fotografen bewerten, welche für sie Aufträge durchgeführt haben. Pressagenturen und Fotografen können darüber hinaus direkt über das versenden von Nachrichten miteinander kommunizieren.

Außerdem ist es möglich, dass eine Presseagentur einem Fotografen „folgt“. Das bedeutet, wenn der Fotograf ein neues Foto zur Versteigerung anbietet oder ein neues Foto in einem Fotoalbum veröffentlicht, wird die Presseagentur benachrichtigt. Das gilt umgekehrt auch für Fotografen. Diese werden dann darüber informiert, wenn eine Presseagentur einen neuen Auftrag erstellt hat.

Außerdem folgt z.B. eine Presseagentur automatisch einem Fotoverkauf, bei dem sie ein Gebot abgegeben hat. Wird das eigene Gebot überboten oder wird etwas an dem Fotoverkauf geändert wird die Presseagentur benachrichtigt.

Das gleiche gilt für Fotografen, welche sich für einen Auftrag einer Presseagentur beworben haben.

Aus zeitlichen Gründen konnten nicht alle Punkte der Zielsetzung umgesetzt werden. Im Kapitel „Abgrenzungen“ wird dies näher beschrieben.

Kommunikationsabläufe und Interaktion

Kommunikationsarten

Es gibt zwei Arten von Kommunikation zwischen Server und Client: synchron und asynchron.

In Bezug auf die Anwendung ist synchrone Kommunikation z.B. der Registrierungsvorgang, das Erstellen eines Auftrags/Fotoverkaufs oder das Abrufen von Profildaten. Also immer da, wo der Client eine Anfrage zum Server sendet und so lange wartet bis der Server geantwortet hat. D.h. während der Wartezeit ist die Anwendung blockiert. Meist ist die Wartezeit aber so kurz, dass dies nicht weiter auffällt.

Asynchrone Kommunikation wird immer dann verwendet, wenn der Client ohne Anfrage an den Server von diesem Daten erhält. D.h. die Anwendung wird in dieser Zeit nicht blockiert. Das Empfangen der Daten erfolgt sozusagen im Hintergrund.

Dies ist z.B. der Fall, wenn ein Fotograf einer Presseagentur „folgt“ und Diese einen neuen Auftrag erstellt hat. Dann erhält der Fotograf, ohne vorher eine Anfrage zum Server gesendet zu haben, eine Nachricht.

Umgekehrt gilt dies auch für Presseagenturen, die einem Fotografen folgen.

Diese Vorgänge synchron umzusetzen, wäre nicht effizient, da dann z.B. der Client immer in einem bestimmten Zeitintervall eine Anfrage an den Server senden müsste um festzustellen, ob es im Fall der Pressagentur neue Fotoverkäufe eines Fotografen gibt oder im Fall des Fotografen neue Aufträge einer Presseagentur.

In nachfolgender Liste wird genauer dargestellt, welche Daten wie ausgetauscht werden:

Synchroner Datenaustausch

- Erstmalige Registrierung im System: Zur Registrierung werden persönliche Daten benötigt, wie Vor- und Nachname, Geburtsdatum, Geschlecht, Adresse, Rolle (Fotograf oder Presse), E-Mail, Handynummer bzw. Festnetznummer, Profilbild, Passwort, Schwerpunkt, Registrierungszeitpunkt.
- Anmeldung: E-Mail-Adresse, Passwort.
- Anpassung der Profildaten, Upload von Daten: Personenbeschreibung, Bilder, ...
- Anfordern von Daten: (eigene) Profildaten, Suche (z.B. Fotos, Jobangebote usw.)
- Bewerbung für Aufträge von Presseagenturen: Bewerbungstext
- Beurteilung vom Fotografen: Presse bewertet mit einer Skala und mit Kommentar.
- Upload von Bild/Bild zum Verkauf anbieten: Bild, Titel, Beschreibung, Lizenz, Tags, Zeitpunkt.
- Gebote für Bilder: Preis
- Nachricht schicken
- Auftrag erstellen (für Fotograf): Titel, Beschreibung, (Lohn), Abgabetermin, Ort, Dringlichkeit.

Asynchroner Datenaustausch

- Neue interessante Aufträge von Presseagenturen an Fotografen senden
- Meldung an Presseagenturen, wenn neue Fotos in ihrem Fachgebiet/Interessengebiet vorliegen: Titel, Rubrik, Vorschau
- Nachricht/Gebot von Presseagentur an Fotograf
- Nachricht/Bewerbung von Fotograf an Presseagentur

Informationsflussdiagramm

Folgende Grafik stellt die oben beschriebenen Kommunikationsvorgänge dar. Dabei wird auch die Richtung in der die Daten verlaufen dargestellt.

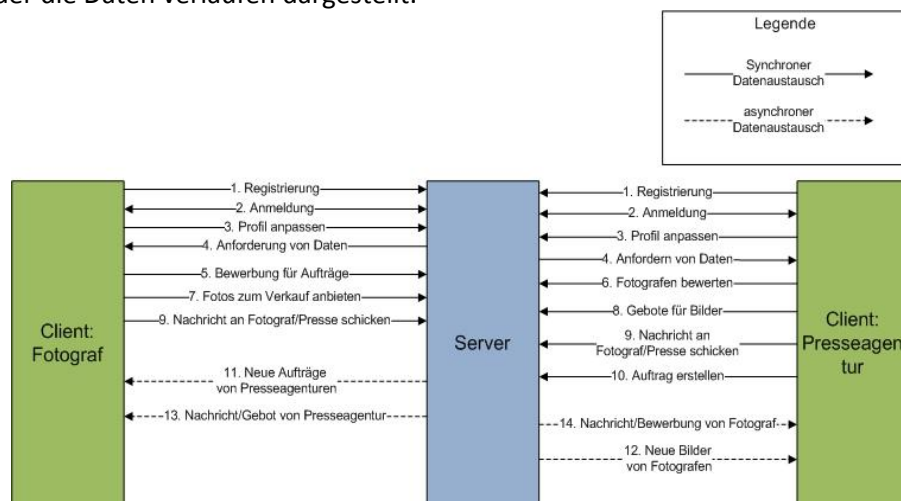


Abbildung 1: Kommunikationsflussdiagramm

Projektbezogenes XML Schema

Die XML-Schemata definieren die Struktur der XML-Dokumente. Da die Anwendung nach dem REST-Paradigma strukturiert werden soll, haben wir uns dafür entschieden, einzelne Schemata für die entsprechenden Ressourcen zu erstellen. Im Abschnitt „Meilenstein 3: REST“ wird das Thema vertieft. Zur Erstellung der XML-Schemata, waren die im Meilenstein 0 herausgefundenen Daten nötig. Die grobe Struktur der XML-Schemata finden Sie in der folgenden Auflistung:

- Jeweils ein XML-Schema für die persönlichen Daten von Fotograf und Presseagentur werden an mehreren Stellen benötigt und können dann in andere Schemata importiert werden
 - Fotograf: Name, Vorname, Geburtsdatum, Geschlecht, Equipment
 - Presse: Agenturname, Hauptstandort, Gründungsjahr
 - Allgemeine Daten: Adresse, E-Mail, Telefonnummer, Profilbild, Beschreibung, Themenbereiche, Webseite, Bewertung, ID
- Registrierung: Rolle(siehe Oben 1.1), Passwort
- Anmeldung: E-Mail, Passwort (Nicht implementiert)
- Profildaten anpassen: siehe 1.1 (Nicht implementiert)
- Auftragsdaten: ID, Titel, Themenbereich, Dringlichkeit, Abgabetermin, Beschreibung, Vergütung
- Bewerbung für Auftrag: Fotograf_ID, Foto(s), Beschreibung, Datum (Metadaten)
- Fotografen Bewerten: Fotograf_ID, Presse_ID, Kommentar, Punkte, Datum (Metadaten) (Nicht implementiert)
- Bild zum Verkauf anbieten: Fotograf_ID, Bild_ID, Preis, Beschreibung
- Gebote für Bild: Presse_ID, Gebot, Bild_ID
- Nachricht schicken/Empfangen: Sender_ID, Empfänger_ID, Betreff, Nachricht (Nicht implementiert)

Nicht alle XML-Schemata wurden, aus zeitlichen Gründen, implementiert. Siehe dazu Kapitel [Abgrenzungen]. Einige Schemata sind im Laufe der Entwicklung hinzugekommen, diese werden im Abschnitt 3 erklärt.

Beispiel photographer.xsd

Als Beispiel analysieren wir zunächst das XML-Schema für die Fotografen. Auf Basis dieses Schemas werden später die eindeutigen Photographer-Ressourcen erstellt. Der Namespace für alle Schemata lautet: `targetNamespace="http://www.example.org/photoBay"`. Alle unsere XML-Elemente- und Attributnamen sind gültig innerhalb des oben genannten Namensraums. PhotoBay ist der Entwicklungsname der Anwendung.

Da mehrere XML-Schemata auf allgemeine persönliche Daten zugreifen (in diesem Fall "PressAgency" und "Photographer"), haben wir uns dafür entschieden das XML-Schema "GeneralPersonalData" zu erstellen, welches natürlich hier inkludiert wird. Das Schema beinhaltet Pflichteingaben (Adresse, Username, Email), und optionale Eingaben (Phone, Description, Website).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.org/photoBay"
3   elementFormDefault="qualified" xmlns:pho="http://www.example.org/photoBay">
4   <xs:include schemaLocation="generalPersonalData.xsd"></xs:include>
```

Der Body des Schemas enthält das Root-Element „photographer“. Ein Fotograf hat die Pflichteingaben „firstname“, „lastname“, „birthdate“, „sex“ und „generalPersonalData“. Von besonderer Bedeutung sind die Attribute „ref“, „ID“, „photosRef“ und „photoSellsRef“.

Der Attribut „ref“ beinhaltet die URI mit der Adresse, wo die Photographer-Ressource persistent gespeichert ist, das heißt es ist eine Referenz zu sich selbst. ID ist die Identifikationsnummer der Ressource, welche einzigartig ist. Die Referenz zu den vom Fotografen hochgeladenen Fotos befindet sich in „photosRef“. Da ein Fotograf Fotos zum Verkauf anbieten kann, muss ihre Referenz innerhalb der Fotograf-Ressource gespeichert werden, und zwar in „photoSellsRef“.

Durch die persistente Speicherung der Referenzen zu den Fotos und Fotos-Verkäufe wird die Architektur des Hypermedia-Prinzips unterstützt. Siehe dazu Kapitel 3 (REST).

```

5<xs:element name="photographer">
6  <xs:complexType>
7    <xs:all maxOccurs="1" minOccurs="0">
8      <xs:element name="firstname" type="xs:string" minOccurs="1"/></xs:element>
9      <xs:element name="lastname" type="xs:string" minOccurs="1"/></xs:element>
10     <xs:element name="birthdate" type="xs:date" minOccurs="1"/></xs:element>
11     <xs:element name="sex" type="pho:sexEnum" minOccurs="1"/></xs:element>
12     <xs:element name="equipment" type="xs:string"/></xs:element>
13     <xs:element name="generalPersonalData" type="pho:generalPersonalDataType" minOccurs="1"/>
14   </xs:all>
15   <xs:attribute name="ref" type="xs:anyURI" use="required"/>
16   <xs:attribute name="ID" type="xs:int" use="required"/>
17   <xs:attribute name="photosRef" type="xs:anyURI" use="required"/>
18   <xs:attribute name="photoSellsRef" type="xs:anyURI" use="required"/>
19   <xs:attribute name="registrationDate" type="xs:date" use="required"/>
20 </xs:complexType>
21 </xs:element>
22<xs:simpleType name="sexEnum">
23  <xs:restriction base="xs:string">
24    <xs:enumeration value="m"/></xs:enumeration>
25    <xs:enumeration value="w"/></xs:enumeration>
26  </xs:restriction>
27 </xs:simpleType>
28 </xs:schema>

```

Ressourcen und die Semantik der HTTP-Operationen für das Projekt

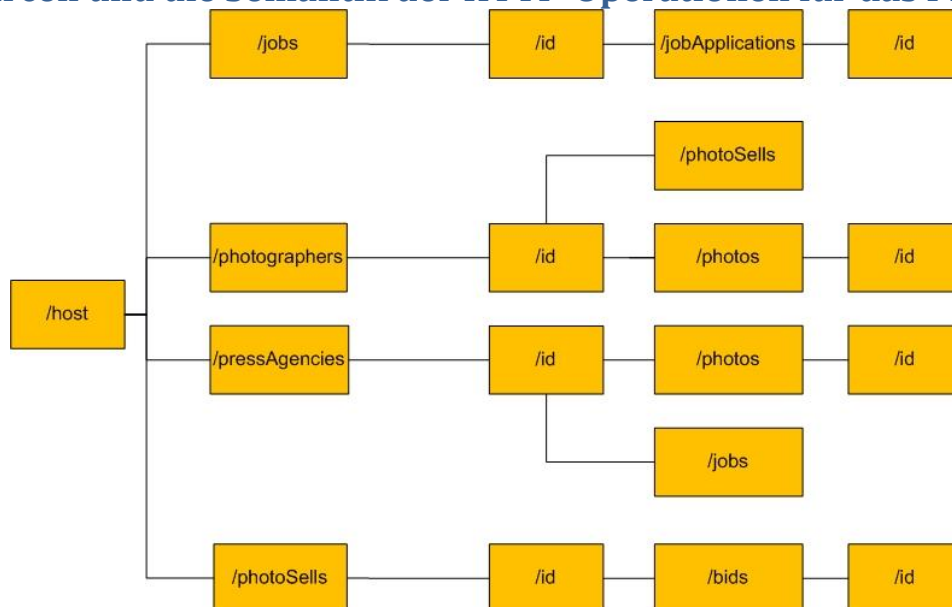


Abbildung 2: Ressourcen-Struktur

Aus der Zielsetzung für das Projekt konnten sieben Hauptressourcen abgeleitet werden. Diese lauten Aufträge (*Jobs*), Fotografen (*Photographers*), Presseagenturen (*PressAgencies*), Fotoverkäufe (*PhotoSells*), Bewerbungen (*JobApplications*), Fotos (*Photos*) und Gebote (*Bids*). Dazu kommen noch die jeweiligen Listen.

Aus diesen Ressourcen wurde oben gezeigte Struktur entwickelt die im nachfolgenden näher beschrieben wird. Einige Funktionen aus der Zielsetzung, wie z.B. das Bewertungssystem oder das Nachrichtensystem wurden hier bereits bewusst ignoriert, da das Projekt sonst zu aufwendig geworden wäre.

Photographers

Fotografen können neu angelegt, gelöscht, geändert oder abgerufen werden. Außerdem ist es möglich eine Liste aller Fotografen auszulesen. Dazu müssen auf die URI *./photographers* die Methoden POST und GET ausgeführt werden. POST um einen neuen Fotografen anzulegen und GET um eine Liste mit allen angemeldeten Fotografen zu erhalten.

Um einen bestimmten Fotografen zu ändern, zu löschen oder abzurufen müssen auf die URI *./photographers/{id}* die Methoden PUT, DELETE oder GET ausgeführt werden.

Da jeder Fotograf eigene Bilder speichern kann existiert innerhalb seines Verzeichnisses ein Verzeichnis für die Fotos `./photographers/{id}/photos/{id}`. Dort wird jedes einzelne Bild mit einer eigenen ID und eigenen Daten gespeichert. Bei einem GET auf die URI `./photographers/{id}/photos/` wird eine Liste mit allen Bildern des Fotografen zurückgesendet. Zum Anlegen von neuen Fotoressourcen muss ein POST auf diese URI angewendet werden.

Zum Abrufen, Löschen oder Bearbeiten von Fotos müssen die Methoden GET, DELETE oder PUT auf die URI der jeweiligen Fotoressource `./photographers/{id}/photos/{id}` angewendet werden.

Außerdem kann ein Fotograf Fotoverkäufe erstellen. Die jeweilige PhotoSell-Ressource wird nicht direkt unterhalb des Fotografen-Verzeichnisses gespeichert. Um aber später die Zuordnung von PhotoSells zu den jeweiligen Fotografen schnell durchführen zu können wird eine Liste aller PhotoSells des jeweiligen Fotografen unterhalb seines Verzeichnisses gespeichert und kann mit einem GET auf die URI `./photographers/{id}/PhotoSells` abgerufen werden.

PressAgencies

Die Struktur und Regeln für die http-Methoden sind analog zu denen der Fotografen. Der einzige Unterschied ist, dass Presseagenturen nicht PhotoSell-Ressourcen sondern Job-Ressourcen erstellen.

Jobs

Job-Ressourcen werden von Pressagenturen erstellt und können von Fotografen angezeigt werden. Zu finden sind die Job-Ressourcen unter der URI `./jobs`. Mit einem GET auf diese URI wird eine Liste aller Jobs (von allen Presseagenturen) zurückgesendet. Mit einem POST wird eine neue Job-Ressource angelegt. Jobs können bearbeitet, gelöscht und abgerufen werden. Dies wird mit den Methoden PUT, DELETE und GET auf die jeweilige Job-Ressource `./jobs/{id}` umgesetzt.

Zu jedem Job kann es Bewerbungen von Fotografen geben, welche innerhalb des jeweiligen Job-Verzeichnisses gespeichert werden (`./jobs/{id}/jobApplications`). Mit einem GET auf diese URI wird wie auch schon bei den anderen Ressourcen eine Liste aller Bewerbungen zurückgesendet.

Mit einem POST wird eine neue Bewerbungs-Ressource angelegt.

Auf die Ressourcen selber kann nur ein GET ausgeführt werden. Die Bewerbungen können nicht gelöscht oder geändert werden, da in der Realität eine Bewerbung bei einem Unternehmen, wenn sie einmal verschickt wurde, auch nicht mehr geändert oder gelöscht werden kann.

PhotoSells

Fotoverkaufs-Ressourcen (PhotoSells) werden von Fotografen erstellt und können von Presseagenturen angezeigt werden. Sie werden unter der URI `./photoSells` gespeichert. Auch hier wird bei einem GET auf die URI eine Liste mit allen Fotoverkäufen von allen Fotografen zurückgesendet. Mit einem POST werden neue Ressourcen angelegt.

Fotoverkäufe können bearbeitet, gelöscht oder abgerufen werden. Dies wird mit den Methoden PUT, DELETE oder GET auf die jeweilige Ressource `./photoSells/{id}` umgesetzt.

Zu jedem Fotoverkauf kann es Gebote (Bids) von Pressagenturen geben, welche innerhalb des jeweiligen Fotoverkaufs-Verzeichnisses `./photoSells/{id}/Bids` gespeichert werden. Eine Liste aller Gebote zu einem Fotoverkauf kann mit einem GET auf diese URI abgerufen werden.

Gebote können wir Bewerbungen nur erstellt und abgerufen aber nicht geändert oder gelöscht werden. D.h. auf eine Gebots-Ressource kann nur ein GET ausgeführt werden. Für den Fotoverkauf selber zählt dabei aber immer nur das höchste Gebot.

CRUD-Matrix

Aus obiger Beschreibung lässt sich folgende CRUD-Matrix ableiten:

Ressource	GET	PUT	POST	DELETE
./jobs	X		X	
./jobs/{id}	X	X		X
./jobs/{id}/jobApplications	X		X	
./jobs/{id}/jobApplications/{id}	X			
./photographers	X		X	
./photographers/{id}	X	X		X
./photographers/{id}/photos	X		X	
./photographers/{id}/photos/{id}	X	X		X
./photographers/{id}/photoSells	X			
./pressAgencies	X		X	
./pressAgencies/{id}	X	X		X
./pressAgencies/{id}/photos	X		X	
./pressAgencies/{id}/photos/{id}	X	X		X
./pressAgencies/{id}/Jobs	X			
./photoSells	X		X	
./photoSells/{id}	X	X		X
./photoSells/{id}/Bids	X		X	
./photoSells/{id}/Bids/{id}	X			

RESTful Webservice

Im Kapitel 2 wurde bereits die Ordnerstruktur der Anwendung vorgestellt. Nun wird tiefer ins Thema Ressourcen eingegangen. Wir haben uns während der Entwicklung der Anwendung auf die folgenden 4 REST-Grundprinzipien konzentriert:

- Ressourcen mit eindeutiger Identifikation
- Hypermedia
- Standardmethoden
- Repräsentationen

Ressourcen mit eindeutiger Identifikation

Als Beispiel betrachten wir die URI von einem Fotografen mit der ID 4. Die Ressource hat somit folgende URI: ./photographers/4

Da diese URIs menschlesbar gestaltet sind, sind sie auch relativ leicht zu interpretieren. Um auf die XML-Repräsentation des Fotografen zugreifen zu können, muss man die Datei photographer.xml abrufen, welche unter dieser URI zu finden ist:

./photographers/4/photographer.xml

URIs bilden einen globalen Namensraum. Die Verwendung einer URI als ID stellt sicher, dass die wesentlichen Elemente eindeutig identifiziert sind.

Hypermedia

Dieses Prinzip hat die formale Beschreibung „Hypermedia as the engine of application state“. Wir greifen in unserer Anwendung auf dieses Prinzip zurück, um einerseits Fotos, Fotosverkäufe, Jobs und Bewerbungen auf die Besitzer zu referenzieren. Wie es in der Abbildung [2] zu erkennen ist, sind alle Jobs und alle Fotos-Verkäufe als eindeutige Ressourcen innerhalb der jeweiligen URIs zu finden. Da diese Ressourcen dort zentral gespeichert werden, muss die Referenz als Attribut des Fotografen beziehungsweise der Presseagentur gespeichert werden. Dadurch vermeiden wir Redundanz und Doppelttspeicherung der Ressourcen an verschiedenen Stellen.

So könnte der Job mit der URI `./25/job.xml` aktualisiert werden, ohne dass Änderungen innerhalb der Presseagentur vorgenommen werden müssen, da sich die Referenz nicht ändert (Außer wenn der Job gelöscht wurde).

Wir veranschaulichen dies anhand der XML-Datei einer Presseagentur.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <pressAgency xmlns="http://www.example.org/photoBay"
3
4     ref="./pressAgencies/1"
5     ID="1"
6     photosRef="./pressAgencies/1/photos"
7     jobsRef="./pressAgencies/1/jobs">
```

Die Presseagentur verfügt über vier Attribute. Über den Attributwert „ref“ lässt sich die URI der Presseagentur ableiten. Sie hat die ID 1, welche natürlich auch in der URI zu finden ist. Innerhalb des Ordners befinden sich die Ordner „photos“ und „jobs“, deren URIs in den Attributen „photosRef“ beziehungsweise „jobsRef“ zu sehen sind.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <jobs xmlns="http://www.example.org/photoBay"
3     ref="./pressAgencies/1/jobs/jobs.xml">
4
5     <jobRef>
6         <jobName>Photos please</jobName>
7         <uri>./jobs/1</uri>
8     </jobRef>
9     <jobRef>
10        <jobName>OneMoreTime</jobName>
11        <uri>./jobs/2</uri>
12    </jobRef>
13 </jobs>
14
```

Hier ist nun die XML-Datei `jobs.xml` dargestellt. Sie referenziert auf die Presseagentur mit der ID „1“ (`./pressAgencies/1`) und enthält gleichzeitig die URI von sich selbst (`./pressAgencies/1/jobs/jobs.xml`). Diese XML-Datei enthält die Liste aller Jobs, die zur Presseagentur mit ID „1“ gehören. Diese werden hier aufgelistet, und zwar jeweils mit dem Jobname und mit der URI.

Standardmethoden

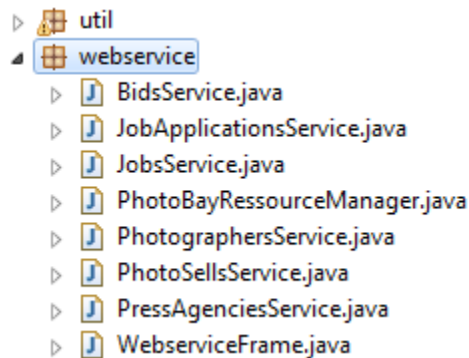
An dieser Stelle werden die Standardmethoden von HTTP PUT, POST, DELETE und GET implementiert. Obwohl HTTP über noch zwei Standardmethoden verfügt (OPTIONS und HEAD), werden diese vernachlässigt, denn die spielen bei der aktuellen Umsetzung der Anwendung keine wesentliche Rolle.

Repräsentationen

Für unsere Anwendung kommen wir mit einem einzigen Repräsentationsformat aus: XML. Dabei haben wir die XML-Repräsentationen recht einfach gehalten: Alle verwenden den gleichen XML-Namespace (`http://www.example.org/photobay`). Dabei können unterschiedliche Wurzelemente verwendet werden (`photographers`, `pressAgencies`, `jobs`, `photoSells`, etc). Bei der Ermittlung von XML-Inhalten, verwenden wir den Mediatyp „`application/xml`“.

Webservice

Im Package *main.java.com.photobay.webservice* befindet sich der Kern unserer REST-Architektur. Folgende Abbildung zeigt den Inhalt des Ordners:



Betrachten wir zunächst die Klassen „PhotographersService.java“ und „PhotoBayRessourceManager.java“.

PhotoBayRessourceManager kümmert sich um die tatsächliche Verwaltung der Ressourcen. Er ist dafür zuständig, Instanzen des JAXB-Marshaller bzw. -Unmarshaller zu erstellen, neue Ressourcen anzulegen, bereits existierende zu aktualisieren oder zurückzugeben und auch zu löschen.

Als konkreter Fall schauen wir uns die GET-Methode eines Fotografen an. Der erste Abruf erfolgt über den Dienst „PhotographersService.java“.

```
16 @Path("/photographers")
17 public class PhotographersService {
18
19     @GET
20     @Path("/{id}")
21     @Produces(MediaType.APPLICATION_XML)
22     public Response getPhotographer(@PathParam("id") int id)
23     {
24         Photographer photographer = PhotoBayRessourceManager.getPhotographer(id);
25
26         if (photographer == null)
27         {
28             return Response.status(Response.Status.NOT_FOUND).build();
29         }
30
31         return Response.ok(photographer).build();
32     }
}
```

Wird die HTTP-Methode GET aufgerufen, so springt das Programm in die Methode `getPhotographer(@PathParam("id") int id)`. Um die Operation ausführen zu können, ist die Übergabe des angeforderten Fotografen nötig. Wir haben uns in diesem Fall für ein `PathParam` entschieden, da keine Filter angewendet werden sollten, sondern nur eine einzige Entität geholt werden muss. Mit `@Produces(MediaType.APPLICATION_XML)` definieren wir den MIME-Type der angeforderten Entität. In „photographer“ speichern wir das durch die Methode `PhotoBayRessourceManager.getPhotographer(id)` geholte `Photographer`-Objekt. Die Methode gibt ein Objekt vom Typ (HTTP) `Response` zurück. Dieses Objekt enthält ein HTTP Status (OK: 200 wenn die Entität existiert, oder NOT_FOUND: 404 wenn nicht), und die Entität des angeforderten Fotografen falls er vorhanden ist, ansonsten Null.

Innerhalb der Klasse `PhotoBayRessourceManager.java` wird nun die tatsächliche Entität geholt. Dafür wird die URI des Fotografen anhand der übergebenen ID aufgebaut, eine neue Instanz des JAXB-Unmarshaller vom Typ `Photographer` erzeugt, und die umgewandelte XML-Repräsentation als Objekt zurückgegeben.

Alle anderen HTTP-Methoden funktionieren analog zum GET, nur die URIs sind unterschiedlich. Bei der Erstellung und

```
40 /**
41  * Get|existing Photographer.
42  *
43  * @param id ID from the Photographer
44  * @return Photographer with ID "id".
45  */
46 public static Photographer getPhotographer(int id)
47 {
48     try
49     {
50         File file = new File("./photographers/" + id + "/photographer.xml");
51         JAXBContext context = JAXBContext.newInstance(Photographer.class);
52         Unmarshaller unmarshaller = context.createUnmarshaller();
53         return (Photographer)unmarshaller.unmarshal(file);
54     }
55     catch (Exception ex)
56     {
57         return null;
58     }
59 }
```

Aktualisierung von Ressourcen ist es aber zu beachten, dass eine neue Instanz des JAXB-Marshallers benötigt wird, da die XML-Datei erstellt wird, und nicht abgelesen.

Kleine Abweichungen treten auf, wenn die PhotoSellList eines bestimmten Benutzers angefordert wird.

```
69 @GET
70 @Path("/query")
71 @Produces(MediaType.APPLICATION_XML)
72 public Response getPhotoSellsList(@QueryParam("owner") String ownerRef)
73 {
74     PhotoSells photoSells = PhotoBayRessourceManager.getPhotoSellsList(ownerRef);
75     if(photoSells == null)
76     {
77         return Response.status(Response.Status.NOT_FOUND).build();
78     }
79     return Response.ok(photoSells).build();
80 }
81
82 }
```

Hier wird `@QueryParam` benutzt, da die angeforderte Liste aus dem Verzeichnis des Besitzers (Owner) geholt werden muss. Die Liste beinhaltet alle Referenzen zu den Foto-Verkäufen des Fotografen.

Konzeption Publish/Subscribe

Für die Asynchrone-Übertragung der Daten greifen wir auf das Publish-Subscribe-Paradigma zurück. Für die Konkretisierung des Paradigmas muss aber zuerst entschieden werden wer Publisher und wer Subscriber ist, welche Topics die Anwendung hat und welche Daten übertragen werden sollen. Folgende Tabelle gibt einen groben Überblick über diese Entscheidungen.

Topic	Publisher	Subscriber	Daten
Neue Jobs	P	F	Auftrags-URI
Neue Fotos	F	P	Bild-URI
Job-Bewerbung	F	P	Bewerbung-URI
Job-Bewerbung Rückmeldung	P	F	Rückmeldung-URI
Gebot	P	F	Gebot-URI
Überboten	Server	P	Gebot-URI

Abkürzungen: P: Presseagentur. F: Fotograf.

Alle Daten, die übertragen werden, sind vom Typ „PayloadMessage“. Ein Payloadmessage enthält eine kurze Nachricht, und die URI mit der Referenz zur Ressource.

Bei der Registrierung als Fotograf oder Presseagentur wird eine neue Topic für den jeweiligen Benutzer angelegt. Es werden aber noch keine Daten an diese Topic gehängt.

Die Presseagentur hängt beim erstellen eines neuen Jobs ein neues Item mit einem PayloadMessage-Object als Payload an ihre eigene Topic. Somit werden alle Subscribers dieser Topic immer darüber benachrichtigt, wenn diese Presseagentur einen neuen Job erstellt hat. Außerdem wird beim erstellen des Jobs auch eine extra Topic für Diesen angelegt, sodass Fotografen, welche sich bewerben die Job-Topic automatisch „subscribe“. Sollte die Presseagentur nun Änderungen an diesem Job vornehmen oder den Job einem Bewerber zuweisen und somit schließen, werden alle Subscribers benachrichtigt.

Der Fotograf hängt beim Hochladen eines neuen Fotos und beim Erstellen eines neuen Foto-Verkaufs ein neues Item mit einem PayloadMessage-Objekt als Payload an ihre eigene Topic. Somit werden alle Subscribers dieser Topic immer darüber benachrichtigt. Auch wenn der Fotograf Änderungen vornehmen soll, werden alle Subscribers benachrichtigt. Die Presseagenturen werden auch benachrichtigt, wenn sie bei einem Foto-Verkauf überboten wurden und wenn die Auktion zu Ende ist.

Aus zeitlichen Gründen konnte nicht alles implementiert werden. Deswegen mussten wir uns nur für ein paar entscheiden. Folgende Funktionalitäten wurden implementiert:

Bei der Registrierung des Fotografen und der Presseagentur wird direkt im Hintergrund eine eigene Topic erstellt. So ist es möglich, dass ein Fotograf eine oder mehrere Presseagenturen folgt und vice versa. Payloads werden an die Nodes angehängt wenn ein Fotograf ein neues Foto hochlädt, und die Presseagentur wird benachrichtigt wenn es neue Foto-Verkäufe gibt.

Die restlich überlegten Funktionalitäten konnten leider nicht implementiert werden.

XMPP-Client

Wir betrachten nun die Erstellung eines PhotoSells und das Anhängen eines PayloadMessages.

Dies geschieht in der PhotographerFrame.java-Datei.

```
photoSell = response.getEntity(PhotoSell.class);
PayloadMessage message = new PayloadMessage();
message.setMessage(photographer.getGeneralPersonalData().getUsername() + "has post a new photo Sell." +
"See link below.");
message.setUri(photoSell.getRef());
cnHandler.assignPayloadToNode(photographer.getGeneralPersonalData().getUsername()+"MainNode",
cnHandler.createNodeConf(PublishModel.publishers, AccessModel.open)
, "payloadMessage", message);
JOptionPane.showMessageDialog(PhotographerFrame.this, "Photo Sell saved!", "Saved!",
JOptionPane.INFORMATION_MESSAGE);
```

Nach dem die neue Ressource „PhotoSell“ über die HTTP-Methode POST angelegt wurde, wird ein neues PayloadMessage-Objekt erzeugt. Zu diesem werden den Username des Fotografen und die URI des PhotoSells hinzugefügt. In einer Instanz des XmppConnectionHandlers wird ein neues Node erstellt, welches das PayloadMessage-Objekt „message“ enthält. Alle Subscribers des Fotografen werden dann benachrichtigt.

Abgrenzungen

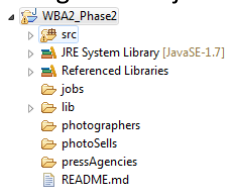
Aus Zeitlichen Gründen haben wir es nicht mehr geschafft alle geplanten Funktionen zu implementieren. Mit dem Client können folgende Funktionen ausgeführt werden:

- Anmeldung und Registrierung (als Fotograf oder als Presseagentur)
- Als Fotograf
 - Eigene Daten abrufen
 - Erstellen eines Fotoverkaufs
 - Abrufen von erstellten Fotoverkäufen
- Als Presseagentur
 - Eigene Daten abrufen
 - Erstellen eines Jobs
 - Abrufen von erstellten Jobs
 - Ändern von Jobs
 - Löschen von Jobs
 - Liste der angemeldeten Fotografen anzeigen
 - Einen Fotografen Subscriben/Unsubscriben

Die Struktur des Eclipse-Projektes und die Benutzung des Clients werden in den folgenden Kapiteln beschrieben.

Projekt-Struktur

Die grobe Projektstruktur sieht wie folgt aus:



Die Ordner *jobs*, *photographers*, *photoSells* und *pressAgencies* müssen vor Ausführung der Anwendung vorhanden sein, da dort die XML-Daten gespeichert werden und die Ordner nicht vom Webservice erstellt werden.

Der eigentliche Code befindet sich innerhalb des src-Ordners und ist auf verschiedene Packages verteilt.

main.java.com.photobay.gui: Dieses Package enthält alle Klassen für die Client-Anwendung

ClientMain.java	Führt die Client Anwendung aus. Muss also zum Ausführen der Anwendung aufgerufen werden.
LoginFrame.java	Stellt das Login-Fenster mit seiner Funktionalität dar.
PhotographerFrame.java	Stellt das Client-Fenster für die Rolle des Fotografen dar.
PressAgencyFrame.java	Stellt das Client-Fenster für die Rolle der Presseagentur dar.
RegisterFrame.java	Stellt das Registrierungs-Fenster dar.
WebserviceConfig.java	Enthält Einstellungen die für den Zugriff auf den Webservice verwendet werden

main.java.com.photobay.util: Enthält einige Hilfsklassen

DeleteFolder.java	Löscht rekursiv Verzeichnisse mit ihren Inhalten
IdGenerator.java	Generiert die IDs für alle Ressourcen
ImageManipulation.java	Wandelt Bilddateien in byte und wieder zurück
ImagePanel.java	JPanel in dem Bilder angezeigt werden können

main.java.com.photobay.webservice: Enthält die Webservice-Klassen

BidsService.java	Enthält die http-Methoden für die Gebot-Ressourcen
JobApplicationsService.java	Enthält die http-Methoden für die Bewerbung-Ressourcen
JobsService.java	Enthält die http-Methoden für die Job-Ressourcen
PhotographersService.java	Enthält die http-Methoden für die Fotograf-Ressourcen
PressAgenciesService.java	Enthält die http-Methoden für die Presseagentur-Ressourcen
PhotoSellsService.java	Enthält die http-Methoden für die Fotoverkauf-Ressourcen
PhotoBayResourceManager.java	Ist für den Zugriff und Manipulationen der XML-Dateien der verschiedenen Ressourcen zuständig. Wird von den einzelnen Service-Dateien aufgerufen
WebserviceFrame.java	Ist für das Starten und Stoppen des Webservices zuständig

main.java.com.photobay.xmppClient: Enthält Klassen für die Kommunikation mit dem Xmpp-Server

XmppConfig.java	Enthält einige Variablen für die Konfiguration der Verbindung zum Server (wird nicht mehr verwendet)
XmppConnectionHandler.java	Ist für die Verwaltung der Verbindung zum Server zuständig. Enthält die Methoden für z.B. das Anmelden, Registrieren von Benutzern und Erstellen und Subscriben von Nodes.

main.java.com.photobay.jaxbfiles: Enthält alle JAXB-Files, welche anhand der XML-Schemata erstellt wurden. Auf die einzelnen Dateien wird hier nicht genauer eingegangen, da sie nur die Struktur der XML-Schemata widerspiegeln.

Benutzung des Clients

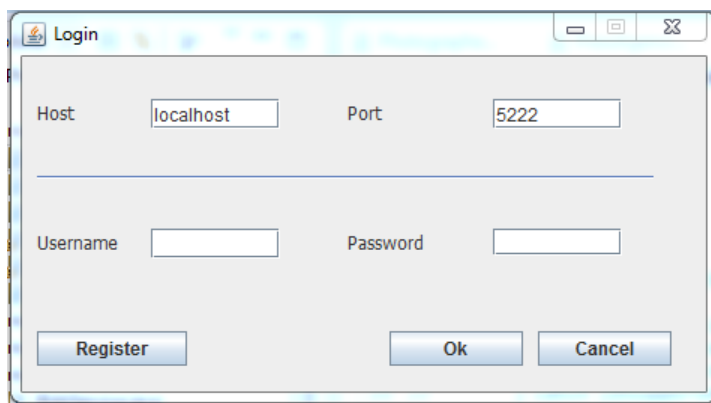
Login

Voraussetzung zum Starten des Clients ist ein eingerichteter OpenFire-Server. Servername und Port können beim Starten der Anwendung angegeben werden und müssen nicht vorher in einer Config-Datei definiert werden.

Außerdem muss der Webservice vorher gestartet werden. Dieser wird standardmäßig unter *http://localhost:4456/* ausgeführt. Der Hostname und der Port können allerdings angepasst werden. Dazu öffnet man zuerst die *WebserviceConfig.java*-Datei im Package *main.java.com.photobay.gui* und passt die Werte der Variablen *WS_HOST* und *WS_PORT* an. Anschließend öffnet man die *WebserviceFrame.java*-Datei im Package *main.java.com.photobay.webservice* und setzt die Variablen *HOST* und *PORT* auf die gleichen Werte.

Dann startet man den Webservice indem man die Datei *WebserviceFrame.java* ausführt. Es erscheint folgendes Fenster indem man auf „Start Webservice“ klickt.

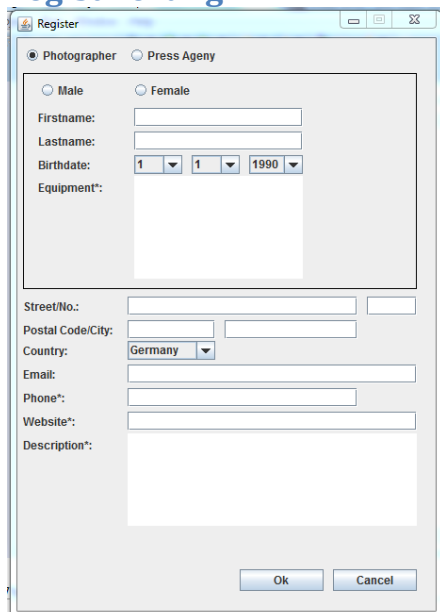
Anschließend kann die Client-Anwendung, mit Ausführung der Datei *ClientMain.java*, gestartet werden. Es erscheint folgendes Fenster:

The screenshot shows a 'Login' dialog box. It has two rows of input fields. The first row has 'Host' with the value 'localhost' and 'Port' with the value '5222'. The second row has 'Username' and 'Password' fields, both empty. At the bottom, there are three buttons: 'Register', 'Ok', and 'Cancel'.

Die Felder Host und Port werden automatisch ausgefüllt, können aber auch angepasst werden. Anschließend müssen die Felder Username und Passwort ausgefüllt werden, egal ob man sich registrieren oder anmelden möchte.

Abbildung 3: Login-Fenster

Registrierung

The screenshot shows a 'Register' dialog box. At the top, there are two radio buttons: 'Photographer' (selected) and 'Press Agency'. Below them are two more radio buttons: 'Male' (selected) and 'Female'. The form contains several input fields: 'Firstname:', 'Lastname:', 'Birthdate:' (with dropdowns for day, month, and year), 'Equipment*:', 'Street/No.:', 'Postal Code/City:', 'Country:' (with a dropdown menu showing 'Germany'), 'Email:', 'Phone*:', 'Website*:', and 'Description*:' (with a large text area). At the bottom, there are 'Ok' and 'Cancel' buttons.

Beim Registrierungs-Fenster kann zwischen Photographer und PressAgency gewählt werden. Je nach Auswahl ändert sich der Inhalt der Box.

Die einzigen Felder, welche nicht ausgefüllt werden müssen, sind die Felder deren Beschreibung einen Stern am Ende haben.

Abbildung 4: Registrierungs-Fenster

Client-Fenster (Fotograf)

In dem Haupt-Tab „My Data“ werden nur die Daten des Fotografen angezeigt.

Abbildung 5: Client-Fenster (Fotograf) - My Data-Tab

Im Tab „My Photo Sells“ können erstellte Fotoverkäufe angesehen oder Neue erstellt werden.

Die erstellte werden in der Liste (links) angezeigt.

Alle anderen Tabs beim Client für den Fotografen wurden deaktiviert, da keine Zeit mehr zur Verfügung stand um diese Funktionalitäten für den Fotografen zum implementieren.

Abbildung 6: "My-Photo-Sell"-Tab im Client-Fenster

Client-Fenster (Presseagentur)

Wie bei Fotografen werden im Haupt-Tab „My Data“ die Daten der Presseagentur angezeigt. Im Tab „My Jobs“ können Jobs erstellt, geändert und gelöscht werden.

Abbildung 8: "My-Jobs"-Tab bei der Presseagentur

Abbildung 7: "My Subscriptions"-Tab

Unter dem Tab „My Subscriptions“ werden neue Meldungen angezeigt, welche von den „subscribed“ Topics an die Presseagentur gesendet werden.

Unter „Search“ werden in der linken Liste alle angemeldeten Fotografen aufgelistet. Die Pressagentur kann dann einen Fotografen auswählen und diesem über den Button „Subscribe“ folgen. Die Elemente innerhalb der mittleren Box sind deaktiviert und besitzen keine Funktionalität, da nicht mehr ausreichend Zeit zur Verfügung stand um diese zu implementieren.

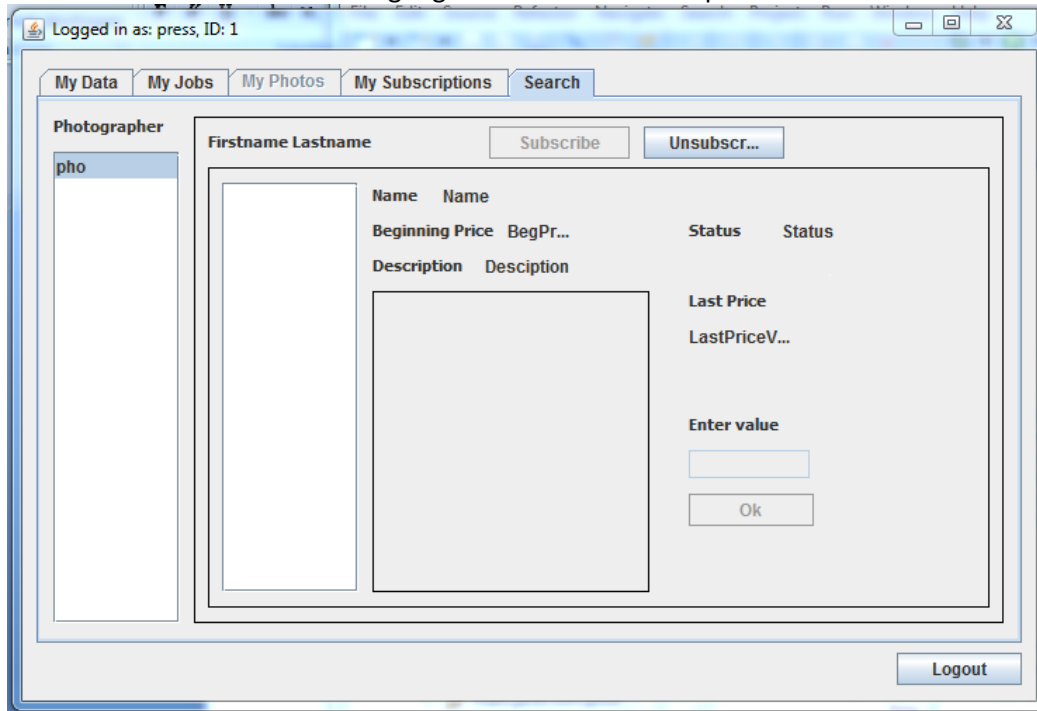


Abbildung 9: Search-Tab

Quellen

<http://de.wikipedia.org/wiki/HTTP-Statuscode>
<http://rest.elkstein.org/2008/02/using-rest-in-java.html>
http://www.se.uni-hannover.de/pages/de:tutorials_restful_guestbook
<http://www.restapitutorial.com/lessons/whatisrest.html>
<http://www.restapitutorial.com/lessons/httpmethods.html>
<http://www.igniterealtime.org/builds/smack/docs/latest/documentation/extensions/pubsub.html>
<http://www.youtube.com/watch?v=AueeTNJsyk>
<http://de.slideshare.net/shaunmsmith/restful-services-with-jaxb-and-jpa>
<http://www.mkyong.com/webservices/jax-rs/jax-rs-queryparam-example/>
<http://www.mysamplecode.com/2011/07/convert-image-to-string-and-string-to.html>
<https://grizzly.java.net/documentation.html>
<http://www.torsten-horn.de/techdocs/jee-rest.htm#JaxRsHelloWorld-Grizzly>
<http://www.igniterealtime.org/builds/smack/docs/latest/javadoc/>