



# **Leuchttisch Designdokument**

2. Mai 2011

Softwaretechnik Projekt 2011

Gruppe 01

Simon Franzen

Lukas Fröhner

Julien Hachenberger

Sven Hesse

Katharina Irrgang

Alexander Knobloch

Lorenz Liebler

Jan Staubach

Tim Sterker

Markus Tacker

Robert Vowe

## Inhaltsverzeichnis

1 Versionen.....	1
2 Umsetzung des Systems.....	2
3 Client.....	3
3.1 Aufbau der GUI.....	3
3.2 Aufbau des Client.....	3
4 Schnittstellen.....	4
5 Server.....	4
6 Exemplarische Architektur-Durchstiche.....	4
7 Anhänge.....	5

## 1 Versionen

Version	Datum	Autor(en)-Kürzel	Änderungen
1.0	03.05.2011	mtack001	Version zur Abgabe
0.6	02.05.2011	mtack001	Einleitung, Strukturierung der Themen
0.5	02.05.2011	sfran001	Sequenzdiagramme Server
0.4	02.05.2011	mtack001	Formatierungsversuche der Wiki-Syntax-Legastheniker gefixt
0.3	02.05.2011	llieb001	Ausformulierung Requests
0.2	02.05.2011	aknob001	Struktur erweitert, Link zu Frontend-Architektur-Beschreibung hinzugefügt
0.1	01.05.2011	shess002	Seite angelegt, grobe Strukturierung, Beschreibung Persistenz und Datenbank

## 2 Umsetzung des Systems

In diesem Dokument werden die Vorgaben zur technischen Umsetzung der Anforderungen aus dem Pflichtenheft definiert.

Das System ist in folgende Komponenten gegliedert:

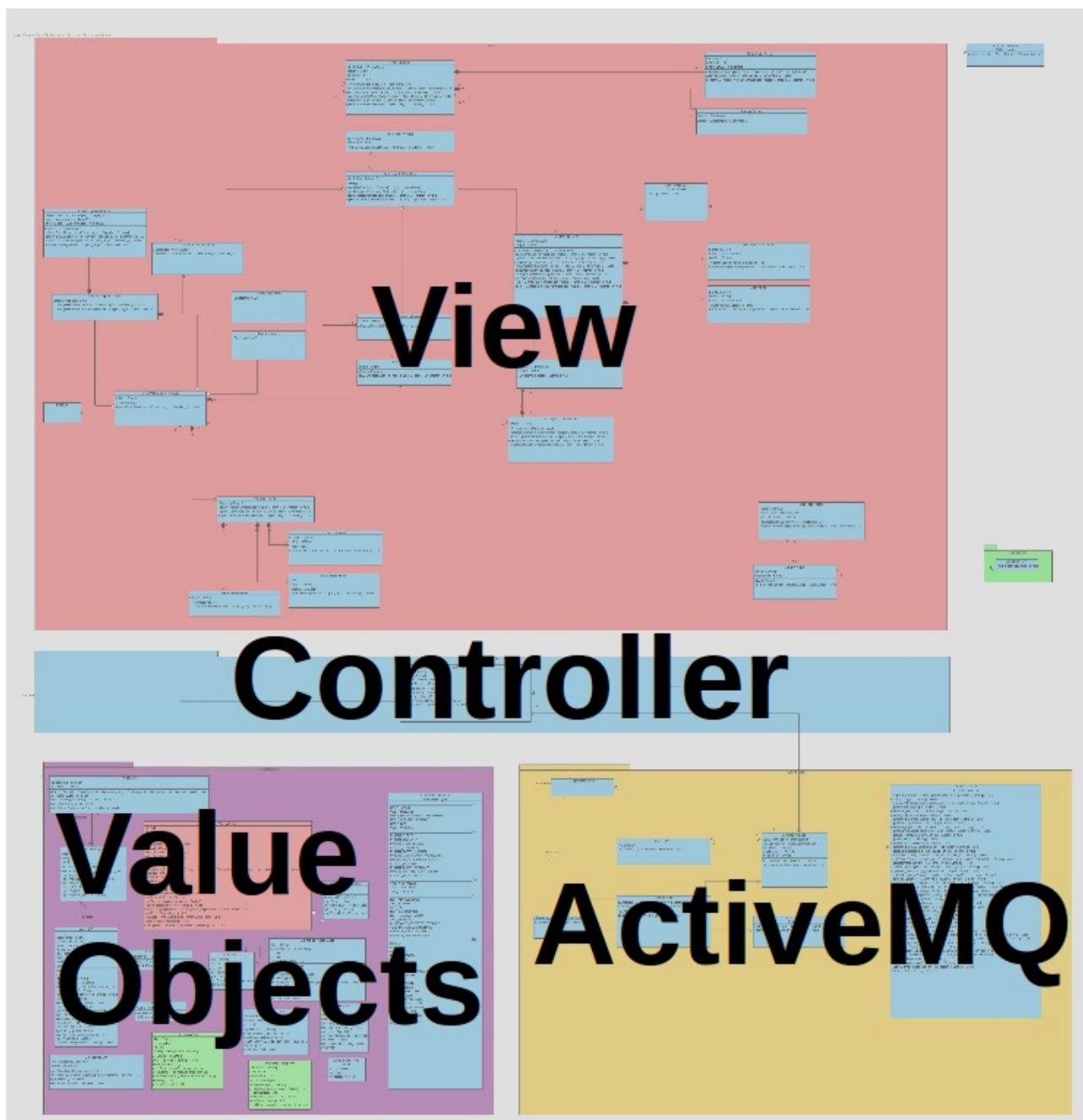
- **Client**  
eine in C# / WPF umgesetzte GUI-Anwendung
- **Schnittstelle**  
teil des Backends, bindet den Client an das Backend an
- **Server**  
ein in Java umgesetzter Server der die Businesslogik des Systems implementiert
- **Persistenz**  
die Businessdaten werden mittels einer Datenbank persistent gespeichert

## 3 Client

### 3.1 Aufbau der GUI

Die Umsetzung des GUI wird durch den **Anhang GUI Scribbles** definiert.

### 3.2 Aufbau des Client



Der Aufbau der Client-Architektur wird im **Anhang Architekturbeschreibung Client**

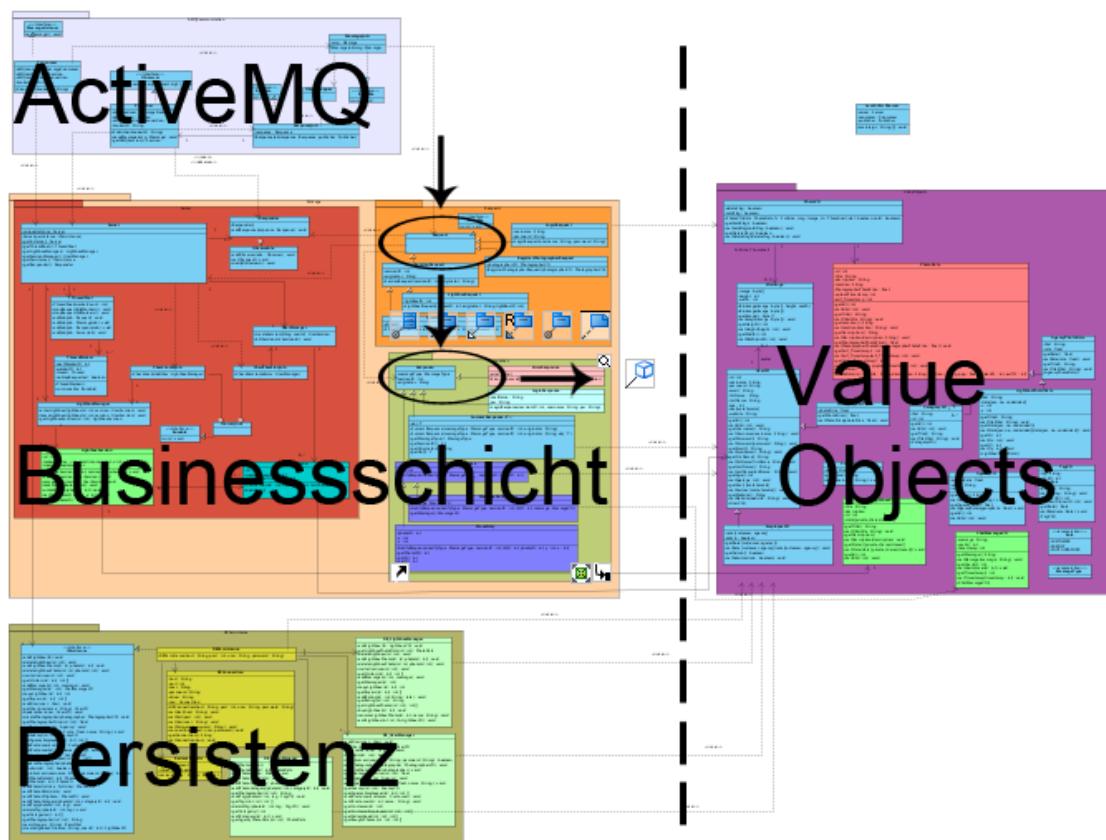
beschrieben.

## 4 Schnittstellen

Die Kommunikation zwischen Client und Server ist im **Anhang Kommunikation Client Server** beschrieben.

Die Methoden, die der Server zur Kommunikation mit dem Client zur Verfügung stellt werden im **Anhang Schnittstellen** erläutert.

## 5 Server



- die Umsetzung der Bussiness- und der Kommunikationsschicht ist in den **Anhängen Architekturbeschreibung Server** und **Request-Übersicht Server** beschrieben
- die Umsetzung der Persistenzschicht ist im **Anhang Persistenz** beschrieben

## 6 Exemplarische Architektur-Durchstiche

Exemplarische Architektur-Durchstiche sind mit Hilfe von Sequenzdiagrammen im **Anhang Exemplarische Architektur-Durchstiche** beschrieben.

## 7 Anhänge

- Schnittstellen
- GUI Scribbles
- Architekturbeschreibung Client
- Persistenz
- Architekturbeschreibung Server
- Request-Übersicht Server
- Kommunikation Client Server
- Exemplarische Architektur-Durchstiche

Dieser Anhang definiert die Umsetzung des GUI anhand von Scribbles.

Die folgenden Scribbles dienen nicht als Gestaltungsrichtlinie bezüglich der Farben, Formen und Größen der GUI-Elemente, sie dienen vielmehr als Gestaltungsrichtlinie bezüglich deren Anordnung.

## Inhaltsverzeichnis

1 Basis-Elemente der GUI.....	2
2 Login-Panel.....	3
3 Fotograf Registrieren-Panel.....	4
4 Initialer Zustand der GUI nach dem Login.....	5
4.1 Agency- / Customer-View.....	5
4.2 Photographer View.....	5
5 Ausgefahrene Panels.....	6
5.1 Agency- / Customer-View.....	6
6 Fotouploader-Panel.....	7
6.1 einzelnes Bild ausgewählt.....	7
6.2 mehrere Bilder ausgewählt.....	7
7 Alle Lightboxen-Panel.....	8
7.1 Agency-View.....	8
7.2 Customer-View.....	8
7.3 Photographer-View.....	9
7.4 Lightbox freigeben Dialog.....	10
8 Adressbuch-Panel.....	11
9 Supersuche-Panel.....	13
10 Lightbox-Panel.....	14
10.1 Agency-View.....	14
10.2 Fullscreen-View.....	14
10.3 Lightbox-Chat.....	14
10.4 Customer-View.....	16
10.5 Photographer-View.....	17
10.6 Metadaten-Panel.....	17
11 Toolbar.....	19
11.1.1 Agency- / Customer-View.....	19
11.2 Benutzerverwaltung.....	19
12 Statistiken.....	22
12.1 Agency-View.....	22
12.2 Customer-View.....	23
12.3 Photographer-View.....	24
13 Dialogboxen.....	25
13.1 Erfolg.....	25
13.2 Erfolgsfall.....	25
13.3 Fehlerfall.....	25

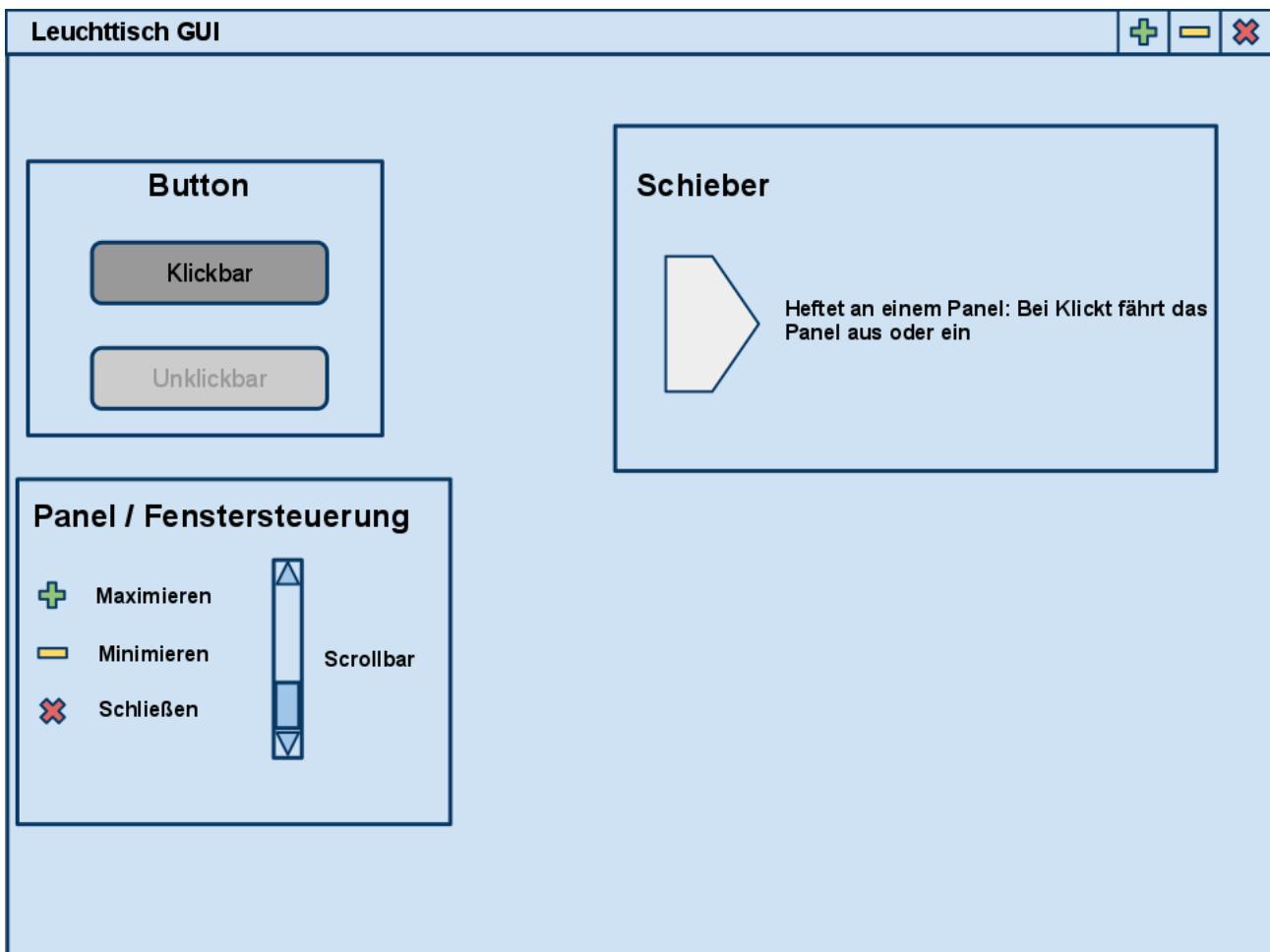


# GUI Scribbles

Anhang zum Leuchttisch Designdokument

# 1 Basis-Elemente der GUI

Vor den Erläuterungen, zunächst ein mal eine Legende über die Elemente die in den Scribbles verwendet werden (für den Fall dass diese nicht sofort zu identifizieren sind):



Dies nur als Vorabinformation bezüglich der Existenz dieser Elemente. Sie werden pro Scribble näher erläutert.

## 2 Login-Panel



Jeder registrierte Benutzer kann sich über das **Login-Panel**, welches beim Start der Anwendung als erstes erscheint, beim System anmelden. Je nach Rolle des Benutzers, bekommt dieser eine eigene View zugewiesen:

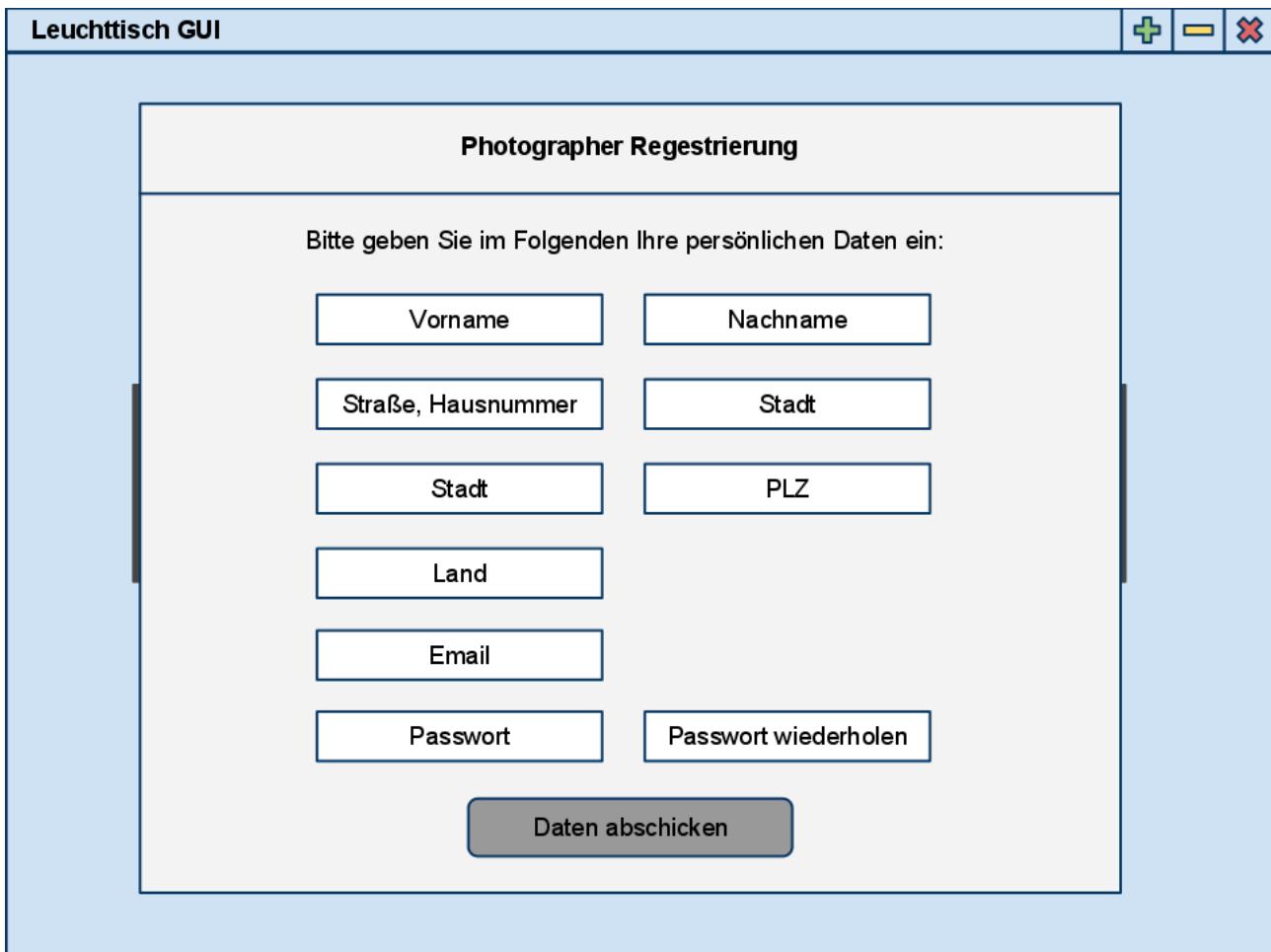
- **Agency-View**
- **Customer-View**
- **Photographer-View**

Da die Unterschiede der verschiedenen Views nicht unerheblich sind, werden diese im folgenden detailliert erklärt.

Nur ein **Photographer** kann sich selbst einen Account anlegen, **Agency-** und **Customeraccounts** werden von uns, der Leuchttisch AG erstellt (bzw. von **Agency-Admins**).

### 3 Fotograf Registrieren-Panel

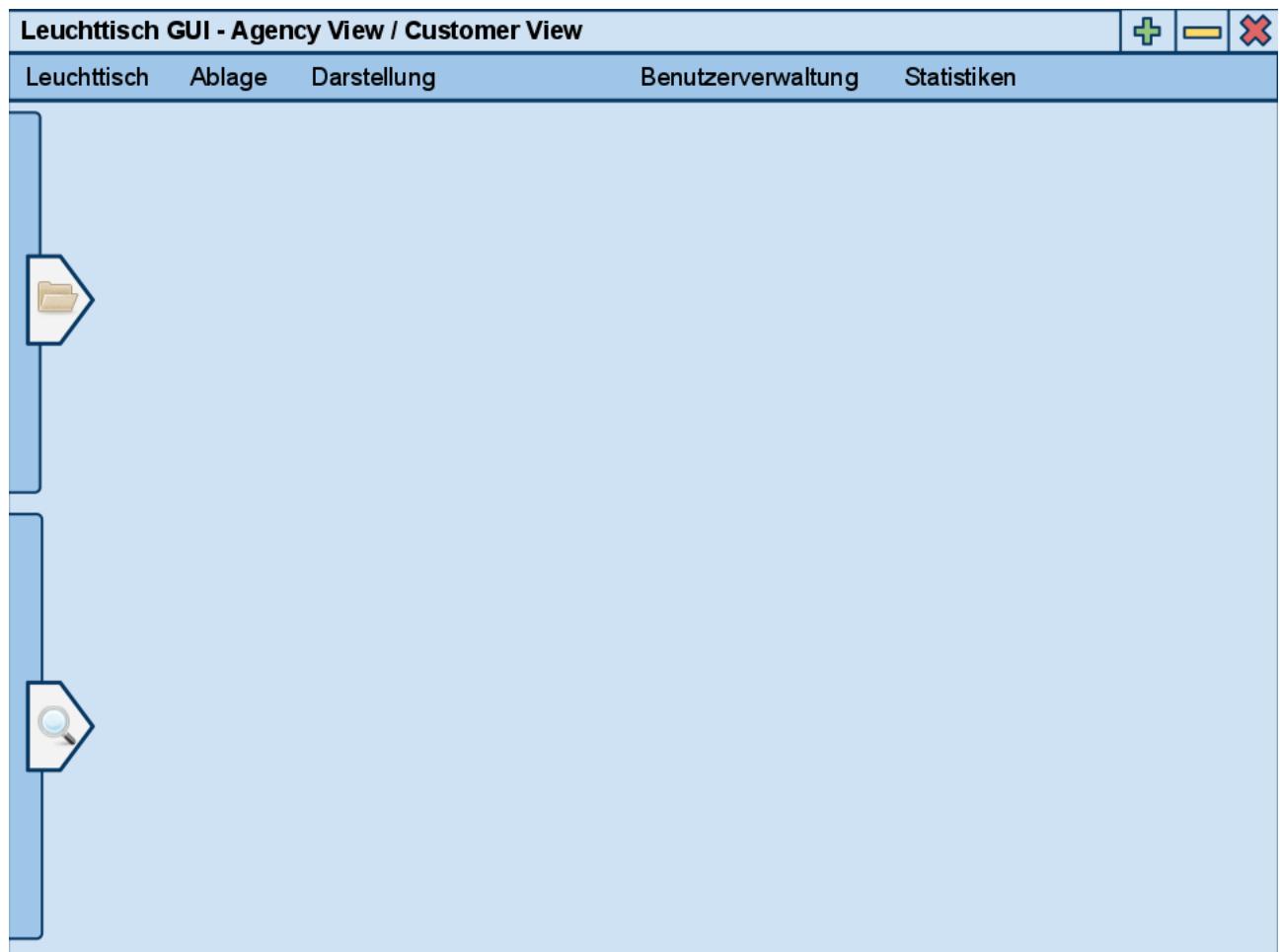
Falls ein Photographer sich registrieren möchte, muss er auf die Schaltfläche **Als Fotograf registrieren?** klicken. Danach erscheint folgender Registrierungs-Dialog:



Daraufhin schaltet die Leuchttisch AG den Photographer frei.

## 4 Initialer Zustand der GUI nach dem Login

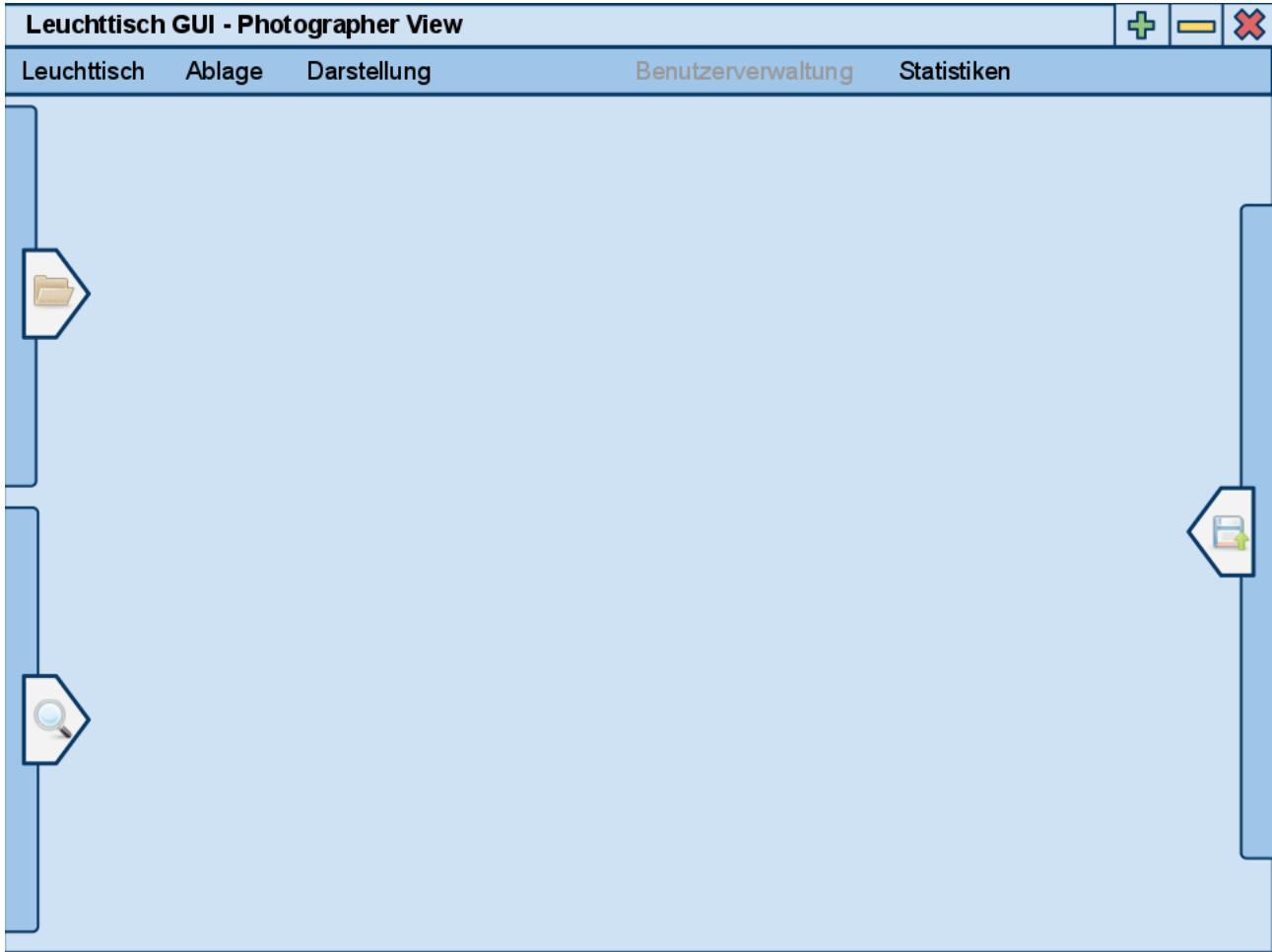
### 4.1 Agency- / Customer-View



Nach einem erfolgreichen Login bekommt der Benutzer die Leuchttisch-GUI zu sehen. Sie baut sich aus folgenden Elementen auf:

- **Toolbar(Oben)**
- **Alle Lightboxen-Panel(Links oben)**
- **Supersuche-Panel(Links unten)**

## 4.2 Photographer View



Zusätzlich zu den bereits bei Agency- / Customer-View erwähnten Panels baut sich die GUI aus folgenden Elementen auf:

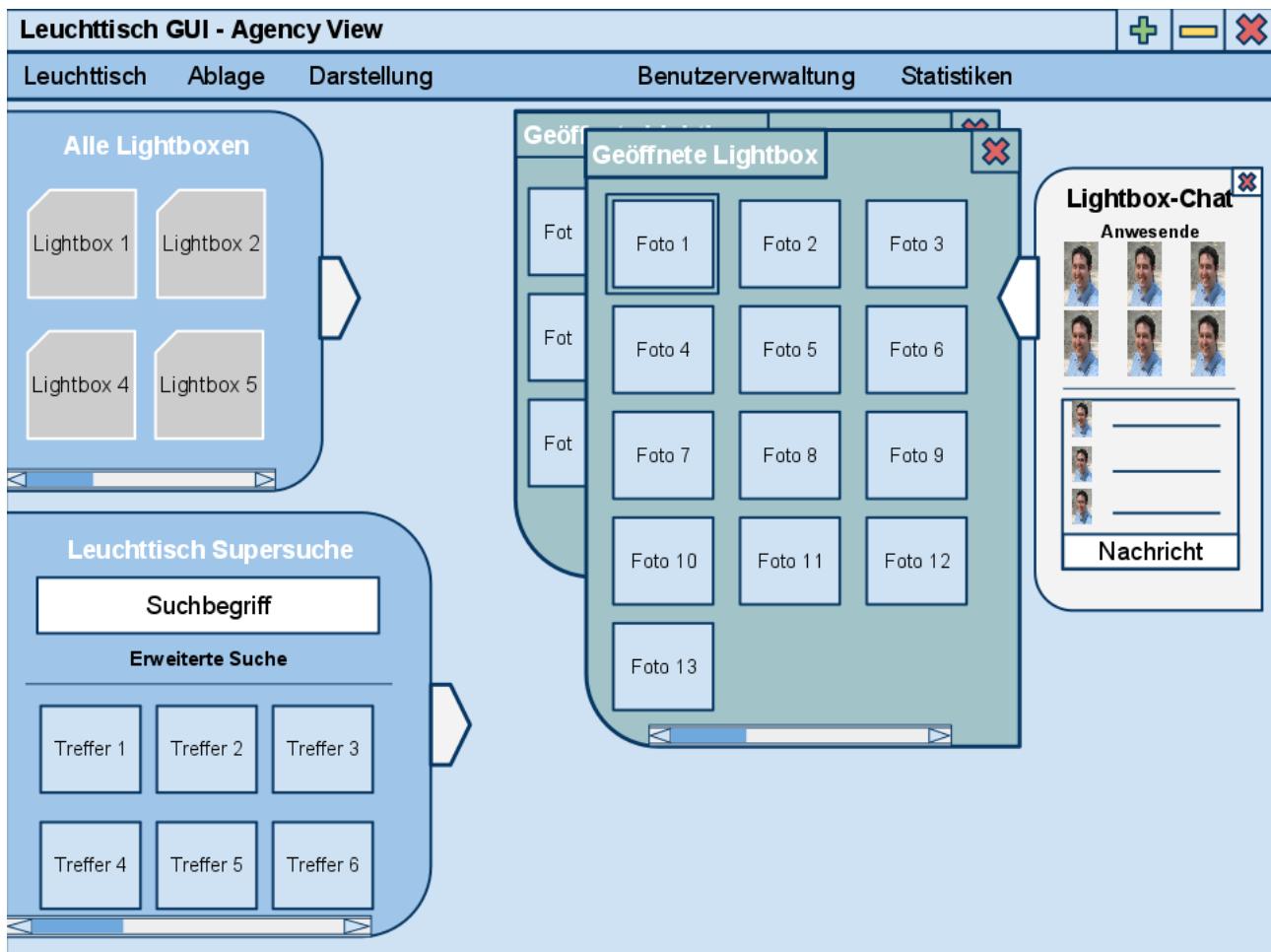
- **Foto-Uploader-Panel(Rechts)**

Im initialen Zustand der GUI für alle Views sind die vorhandenen Panels, sprich alle Lightboxen, Supersuche, Alle-Lightboxen-Panel und Foto-Uploader eingefahren. Über die Schieber kann man jedes Panel für sich einfahren und ausfahren. Auch im eingefahrenen Zustand kann man erkennen, um welches Panel es sich dreht, da sich auf den jeweiligen Schiebern erklärende Symbole befinden.

# 5 Ausgefahrene Panels

Die folgenden Gesamtansichten dienen als Übersicht bzw. Hilfe, wie die einzelnen Panels im Anwendungsfenster anzutragen sind. Darauf folgend werden Einzelansichten für die Panels gezeigt, welche detailliert darstellen, welche Buttons z.B. in ein Panel gehören.

## 5.1 Agency- / Customer-View



Das Panel **Alle Lightboxen** enthält eine Übersicht über Lightboxen welche dem jeweiligen Benutzer zugänglich sind.

Die **Supersuche** der GUI enthält zum einen die Möglichkeit der direkten Stichwortsuche, zum anderen lassen sich noch Filtereinstellungen vornehmen. *Dazu später mehr in der Detailansicht.*

Sobald eine Lightbox aus dem **Alle-Lightboxen-Panel** geöffnet wurde, öffnet sich ein neues Panel, welches den Inhalt der jeweiligen Lightbox darstellt. Das Panel ist frei innerhalb des Programms zu bewegen und anzuordnen.

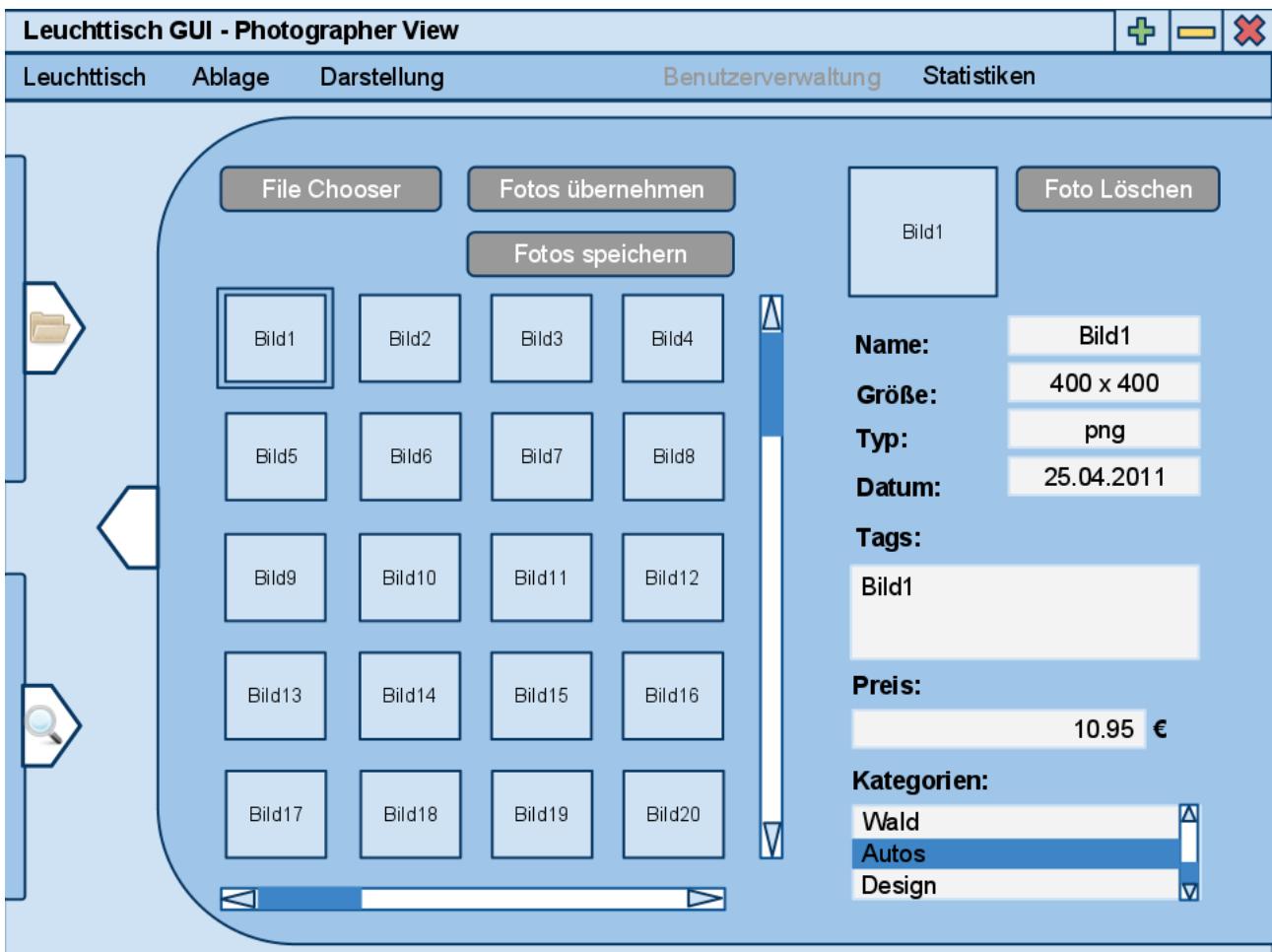
Der **Lightbox-Chat** ist an die entsprechende **Lightbox** angedockt und ein und ausklappbar. Das Panel enthält eine Übersicht über die Teilnehmer der Lightbox und die

Möglichkeit Chatnachrichten zu schreiben und zu lesen.

Das **Metadaten-Panel** hat die Headline Foto 1 im Scribble. Hier werden alle relevanten Informationen, bzw. Metadaten eines selektierten Fotos angezeigt.

# 6 Fotouploader-Panel

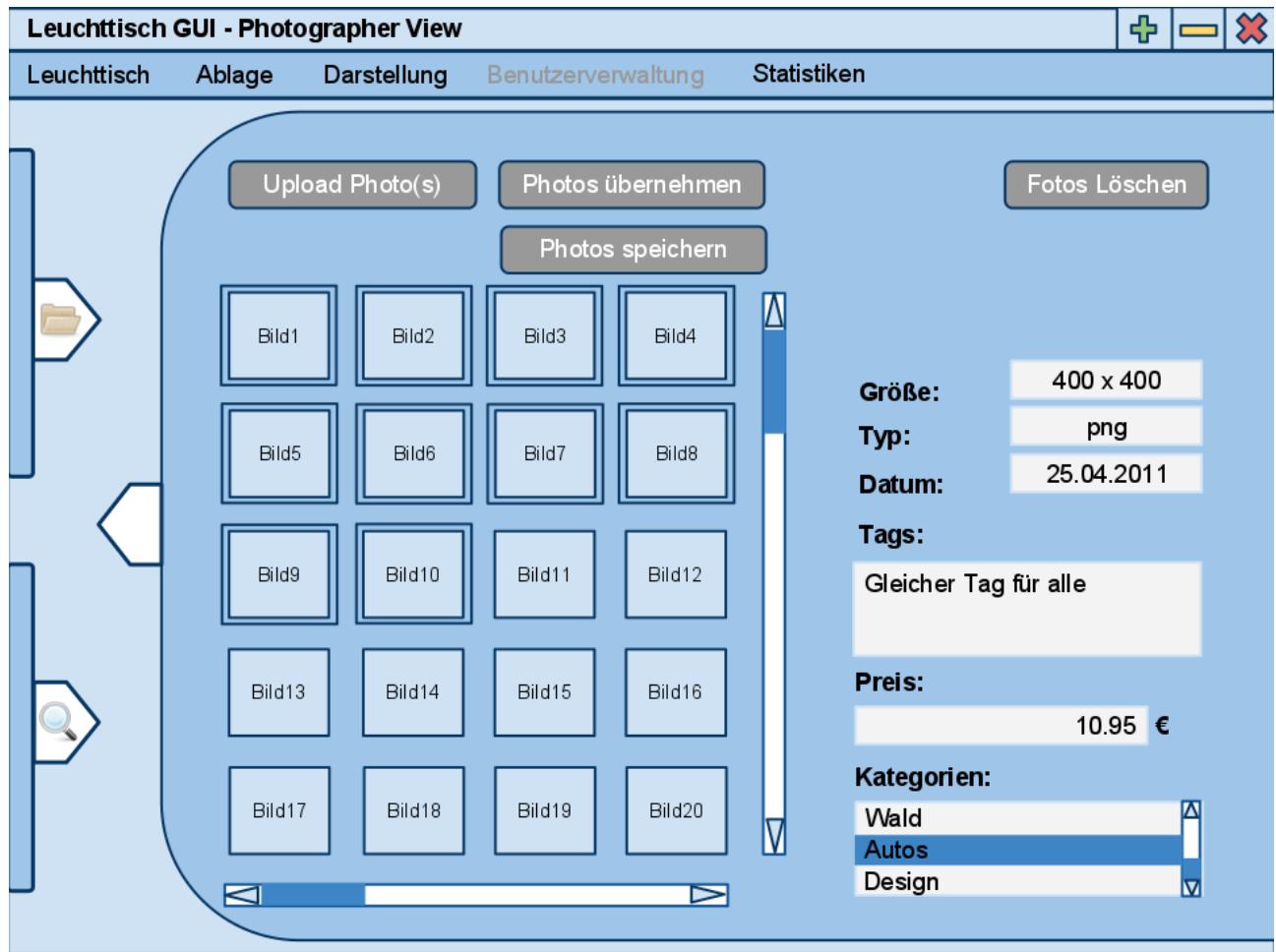
## 6.1 einzelnes Bild ausgewählt



Wie hier zu sehen, hat der Photographer die Möglichkeit über ein **Uploader-Panel** die von ihm gewünschten Fotos über einen **File-Chooser** von seiner Festplatte auszuwählen. Die ausgewählten Fotos werden dem Server geschickt und in dem Panel angezeigt, wo der Photographer dann die Möglichkeit hat, die Fotos zu kategorisieren, den Bildnamen, Größe, Typ, Datum, Tags und einen Preis festzulegen. Wenn dieses geschehen ist, kann der Photographer über den Button **Fotos übernehmen** die eingegebenen Daten ebenfalls an den Server schicken. Weitere Bearbeitungsmöglichkeiten bieten die Buttons:

- **Fotos speichern:** Der Photographer kann über diesen Button seine aktuell bearbeiteten Fotos lokal in einer Lightbox abspeichern, falls er aus irgendeinem Grund die Bearbeitung der Fotos nicht abschliessen konnte und zu einem anderen Zeitpunkt weiter machen möchte.
- **Foto löschen:** Der Photographer hat die Möglichkeit ausgewählte Fotos zu löschen vor dem Upload.

## 6.2 mehrere Bilder ausgewählt



Neben der Möglichkeit ein einzelnes Bild auszuwählen, kann ein Fotograf auch mehrere Bilder auswählen. Einzig die Option der Namensgebung fällt weg, die anderen Daten werden dann gleichermaßen allen ausgewählten Bildern hinzugefügt.

# 7 Alle Lightboxen-Panel

## 7.1 Agency-View



Dieses Panel bietet zunächst die Möglichkeit zu zwei verschiedenen Ansichten (siehe klickbare Registerkarten am oberen Rand des Panels):

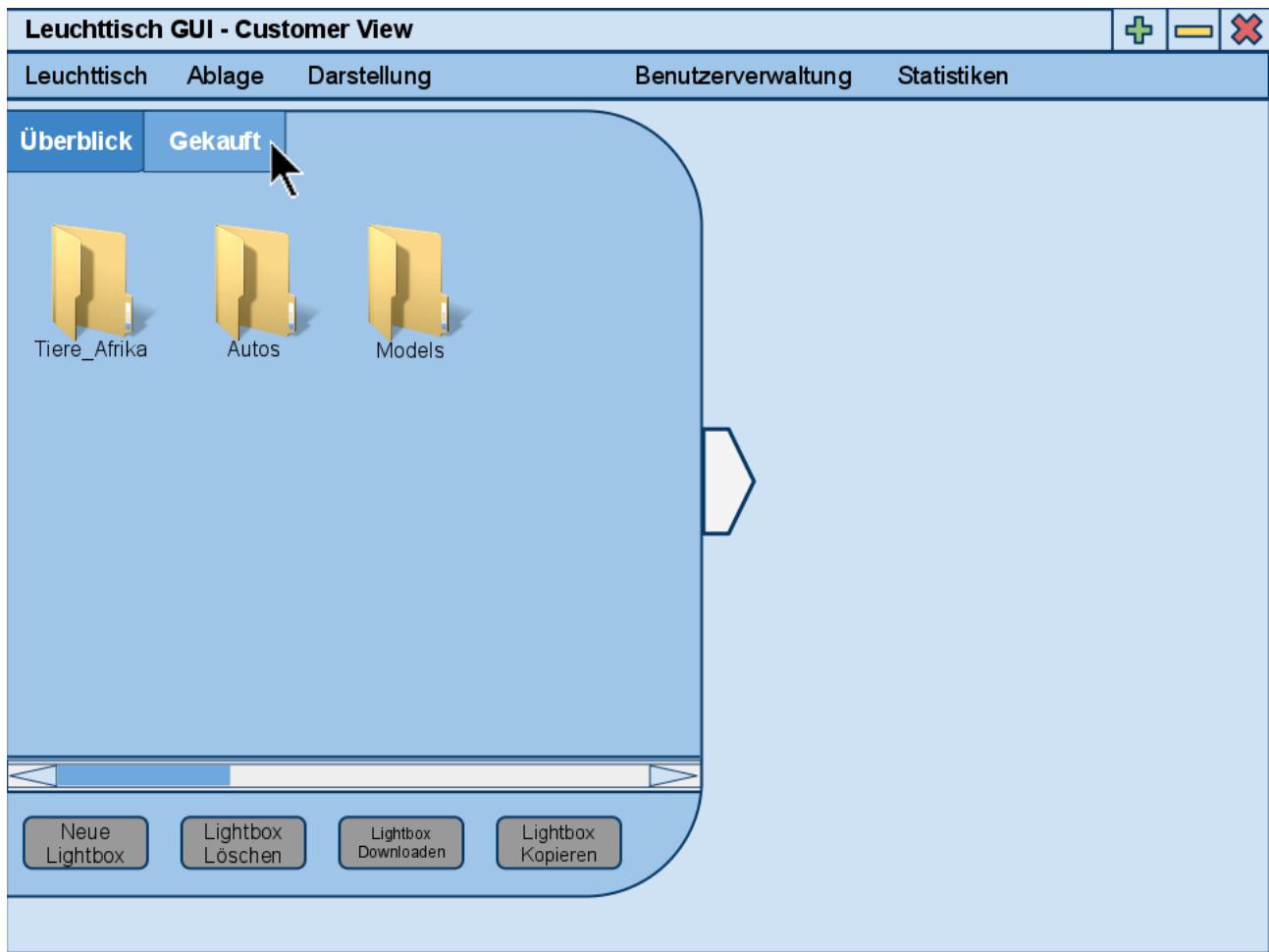
1. **Übersicht** aller Lightboxen die der Benutzer erreichen kann  
Hier werden ALLE Lightboxen angezeigt, die dem eingeloggten Benutzer zugänglich sind.  
Besonderes Augenmerk liegt hierbei auf der ersten Lightbox innerhalb der Liste, dem **FOTO-POOL**. In dieser Lightbox liegen alle von den Photographern hochgeladenen Fotos. Falls ein Agency-Employee nicht über die **Supersuche** gehen möchte, sondern im kompletten Fotobestand stöbern möchte, wäre es über diese Lightbox möglich.
2. **Freigegebene Lightboxen**  
Hier werden nur diejenigen Lightboxen angezeigt, welche den Customern freigegeben wurden. Sprich: Fertiggestellte Lightboxen, welche verkauft werden sollen.

Die Lightboxen lassen sich per Drag 'n Drop beliebig innerhalb des Panels verschieben. Eine Scrollbar befindet sich am unteren Rand des Panels.

Unterhalb der Scrollbar befinden sich diverse Buttons:

- **Neue Lightbox:** Button um eine neue, leere Lightbox zu erstellen.
- **Lightbox löschen:** Button um eine ausgewählte Lightbox zu entfernen.
- **Lightbox freigeben:** Button um eine ausgewählte Lightbox einem Customer freizugeben. Nach Klick öffnet sich ein Freigabe-Dialog, mit welchem der gewünschte Customer gesucht/ausgewählt werden kann. *Mehr dazu im nächsten Scribble.*
- **Lightbox kopiere:** Button um eine ausgewählte Lightbox zu kopieren.

## 7.2 Customer-View



Es gibt zwei Unterschiede zur Agency-View: Das Vorkommen einer Registerkarte: Statt **Freigegebene Lightboxen** gibt es hier **Gekaufte Lightboxen**. Das Vorkommen eines Buttons: Statt **Lightbox Freigeben** gibt es hier **Lightbox Drucken**.

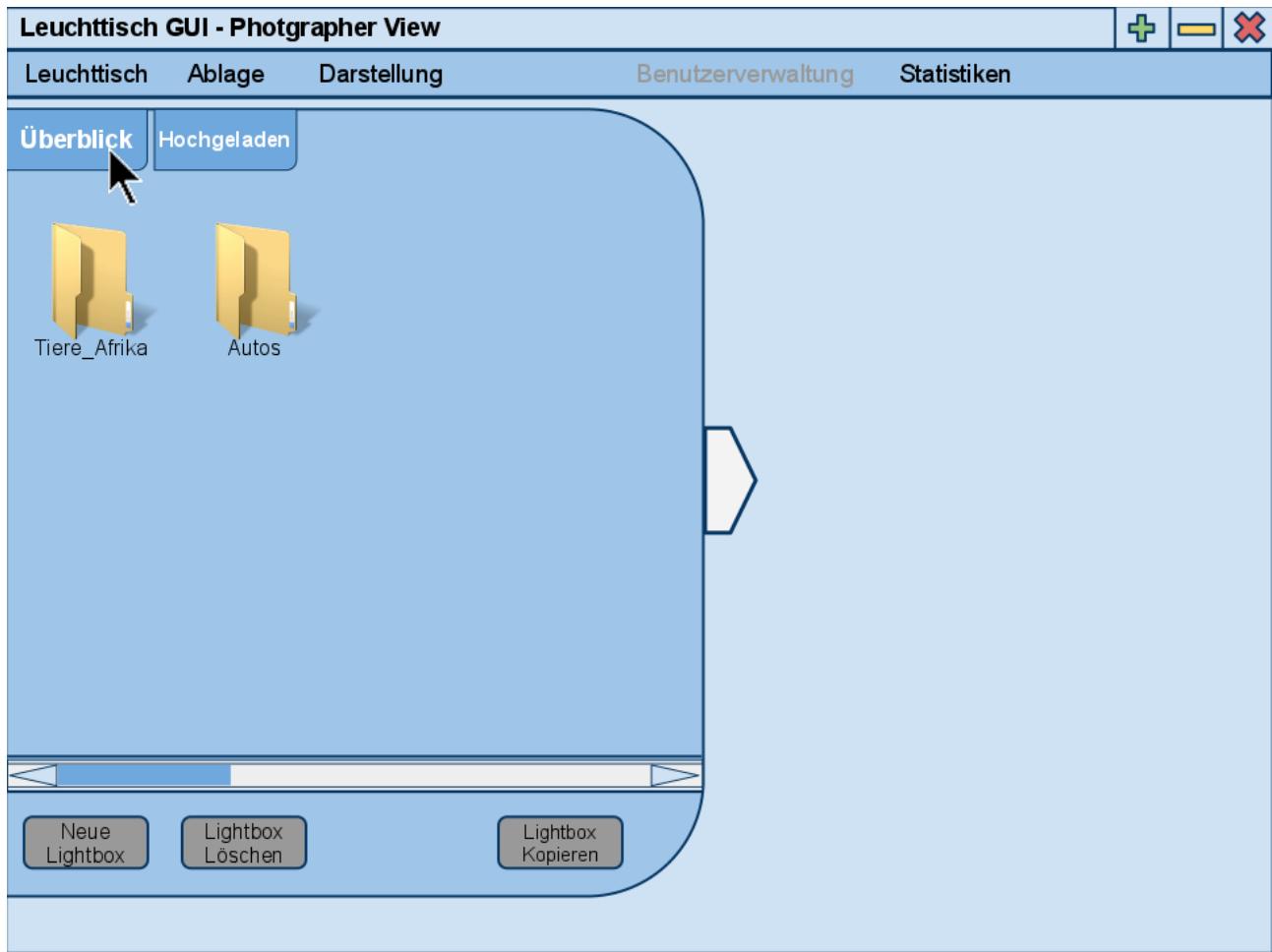
**Gekaufte Lightboxen** Hier werden nur diejenigen Lightboxen angezeigt, welche der Benutzer gekauft hat.

Die Lightboxen lassen sich per Drag 'n Drop beliebig innerhalb des Panels verschieben. Eine Scrollbar befindet sich am unteren Rand des Panels.

Unterhalb der Scrollbar befinden sich diverse Buttons:

- **Neue Lightbox:** Button um eine neue, leere Lightbox zu erstellen.
- **Lightbox löschen:** Button um eine ausgewählte Lightbox zu entfernen.
- **Lightbox drucken:** Button um die Lightbox bzw. die Fotos innerhalb der Lightbox auszudrucken. Die Fotos haben nicht die originale Größe, vielmehr eine Thumbnail-Größe. Sie dienen dazu, einen schnellen Übersichtsdruck der Fotos seinen Mitarbeiter zu zeigen um diese evtl. zu besprechen. Der **Customer** erhält alle Fotos in einer **PDF-Datei**.
- **Lightbox kopieren:** Button um eine ausgewählte Lightbox zu kopieren.

## 7.3 Photographer-View

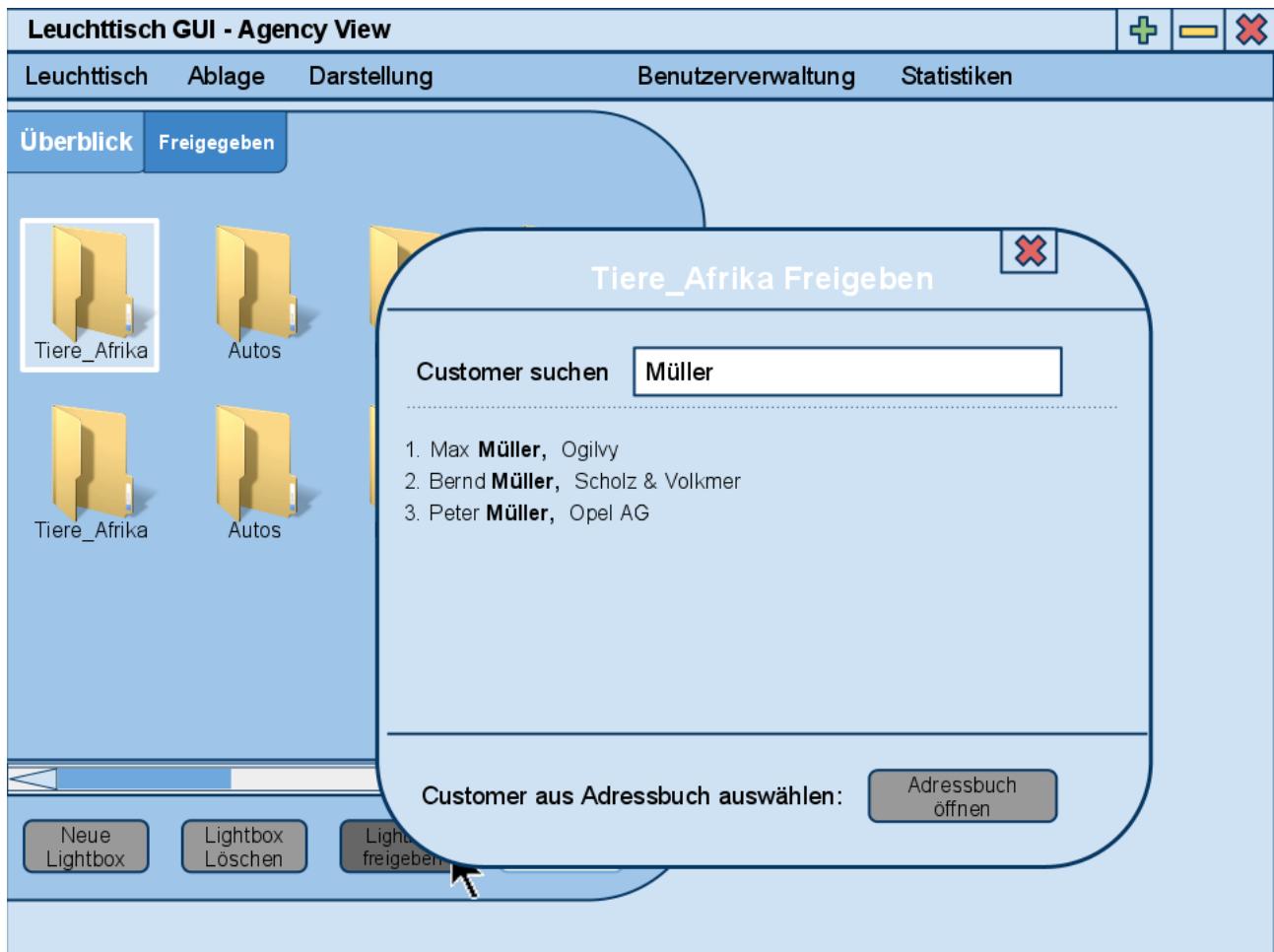


Es gibt zwei Unterschiede zur **Agency- / Customer-View**: Das Vorkommen einer Registerkarte: Es gibt nur eine **Übersicht** über alle Lightboxen. Das Vorkommen eines Buttons: Hier gibt es nur drei statt vier verschiedene Buttons.

- **Neue Lightbox:** Button um eine neue, leere Lightbox zu erstellen.

- **Lightbox löschen:** Button um eine ausgewählte Lightbox zu entfernen.
- **Lightbox kopieren:** Button um eine ausgewählte Lightbox zu kopieren.

## 7.4 Lightbox freigeben Dialog



Wie bereits erwähnt, hat der Benutzer in der **Agency-View** die Möglichkeit einem Customer oder einem Agency-Employee eine bestimmte Lightbox freizugeben, damit dieser die Lightbox samt Inhalt sieht. Ein Customer könnte die Lightbox danach kaufen und ein Agency-Employee könnte an der Lightbox mitarbeiten, sprich Fotos hinzufügen/entfernen. Dieser Dialog bietet dem Benutzer zwei Möglichkeiten:

1. **Direkte Suche nach einem Customer:** Per Stichwort suche kann direkt nach einem Customer gesucht werden. Alle Treffer, egal ob in Vor- Nachname oder Firmenname, unterhalb der Suche angezeigt.
2. **Customer aus Adressbuch auswählen:** Falls der Benutzer den Namen des Customers nicht im Kopf hat, kann er über sein Adressbuch nach diesem suchen. Nach Klick auf den Adressbuchbutton wird das Adressbuch geöffnet. *Mehr zum Adressbuch im nächsten Scribble.*

## 8 Adressbuch-Panel



Sortiert nach den **Gruppen**:

- **Alle Kontakte:** Hier sind **alle** Kontakte, zu denen der Benutzer Zugang hat, aufgelistet.
- **Meine Agency:** Hier sind alle Kontakte aus der eigenen **Agency** aufgelistet.
- **Meine Customer:** Hier sind alle **Customer**, zu denen der Benutzer Zugang hat, aufgelistet.
- **Meine Photographer:** Hier sind alle **Photographer**, zu denen der Benutzer Zugang hat, aufgelistet.

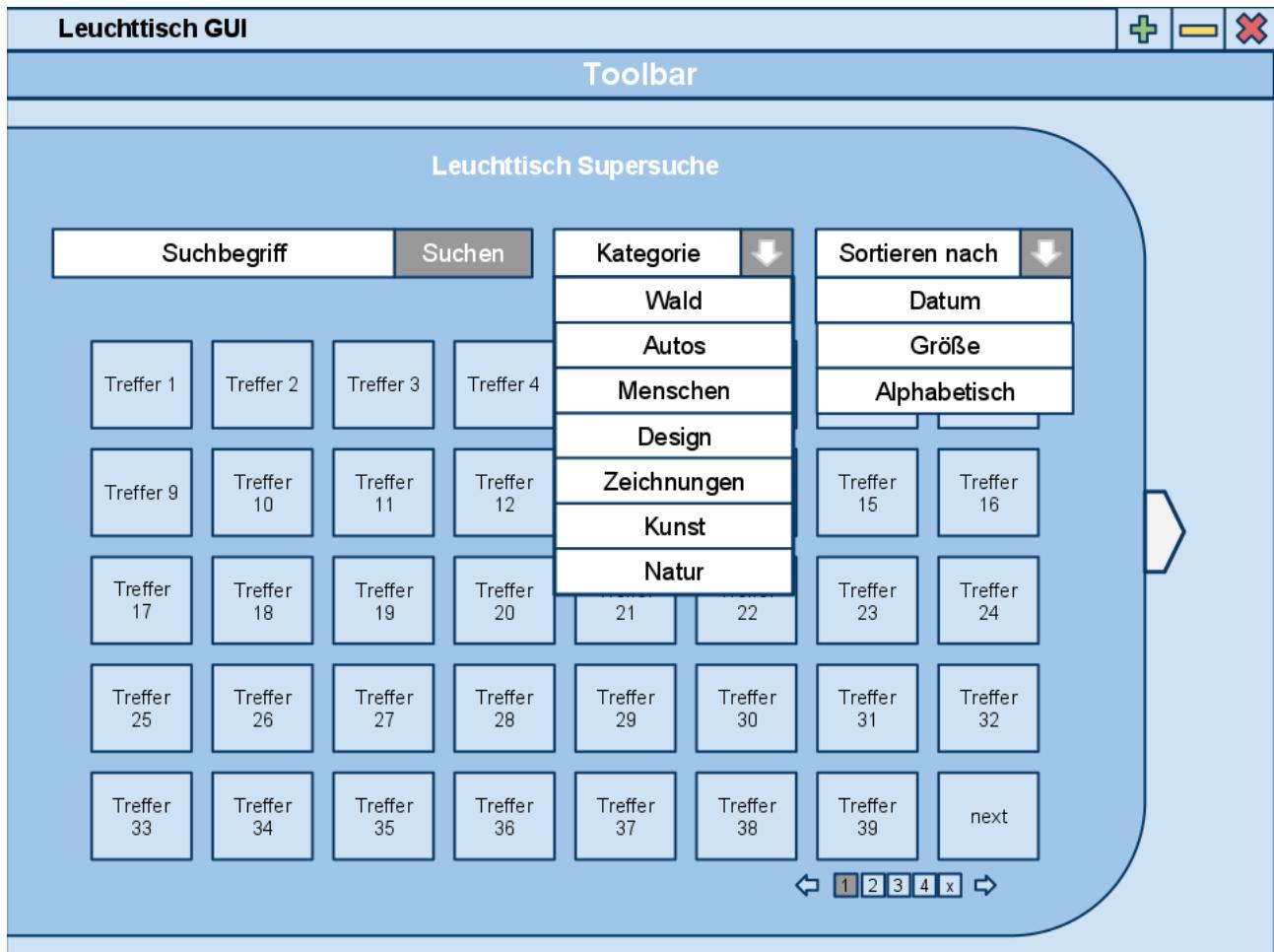
Sobald eine Gruppe selektiert ist, werden unter dem Menüpunkt **Name** alle Kontakte aufgeführt, die der jeweiligen Gruppe angehören. Klickt der Benutzer auf einen Namen, erscheint daneben eine **Detaillansicht** über den ausgewählten Kontakt. Ist ein Kontakt selektiert, kann der Benutzer zwei verschiedene Buttons anklicken:

- **Zu Lightbox hinzufügen:** Die zuvor ausgewählte Lightbox freigeben. Der

Ausgewählte Kontakt hat nun Zugang zur **Lightbox** / zum **Lightbox-Chat**.

- **Aus Lightbox entfernen:** Den Ausgewählten Kontakt den Zugang zur zuvor ausgewählten **Lightbox** / zum **Lightbox-Chat** entziehen.

## 9 Supersuche-Panel



Die Leuchttisch Supersuche ist jeder Benutzerrolle zugänglich.

Der Benutzer kann durch einfache Stichwortsuche suchen. Weiterhin hat der Benutzer die Möglichkeit einen **Kategorien-Filter** einzustellen und sich die Ergebnisse nach den Kriterien **Datum**, **Größe** oder **Alphabetisch** ausgeben zu lassen.

Die Treffer werden dann unterhalb des Suchfeldes angezeigt. Pro Seite wird nur eine bestimmte Anzahl von Treffern angezeigt. Übersteigt das Ergebnis diese Anzahl, dann erscheint am unteren Rand des Panels eine **Seiten-Navigation**, über welche alle Treffer zugänglich sind. Der Benutzer hat die Möglichkeit, per Drag 'n Drop die angezeigten Treffer in eine Lightbox zu ziehen.

# 10 Lightbox-Panel

## 10.1 Agency-View



Der Benutzer kann aus dem **Alle-Lightboxen-Panel**, welches weiter oben gezeigt wurde, per Doppelklick eine ausgewählte **Lightbox** öffnen. Daraufhin erscheint dieses Panel.

Im oberen Bereich des Panels steht zunächst der Name der **Lightbox**. Dann folgt eine Liste mit Thumbnails aller **Fotos** welche in dieser **Lightbox** enthalten sind.

## 10.2 Fullscreen-View

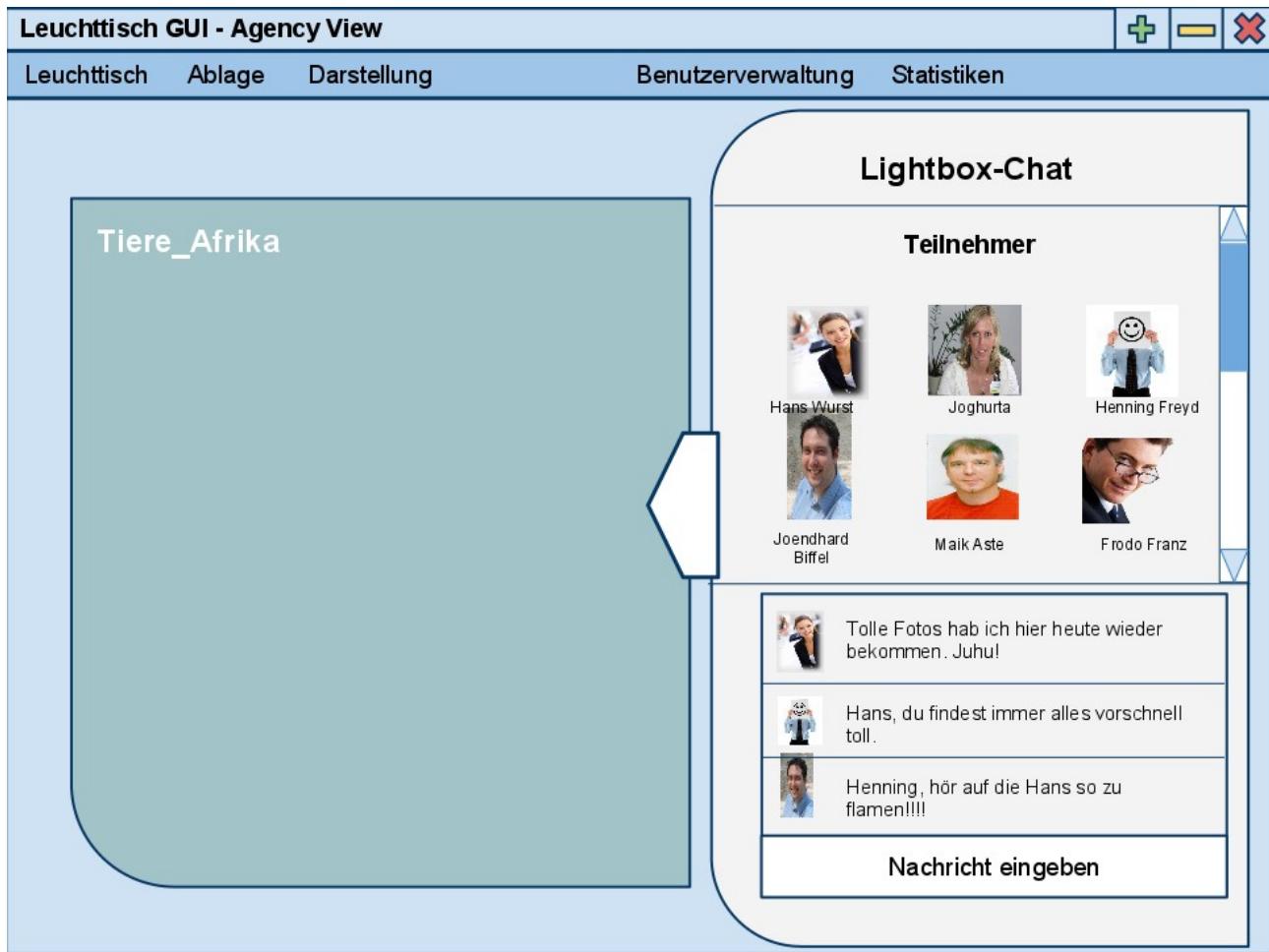
Per Doppelklick auf eines der Fotos erscheint eine **Fullscreen-View**:



In der **Fullscreen-View** kann über die beiden Pfeile, links und rechts neben dem aktuell angezeigten Bild, durch die Lightbox navigiert werden. Die Reihenfolge der Fotos ist die gleich wie innerhalb der Lightbox. Wenn also z.B. in der Lightbox ein Bild rechts neben dem angezeigten Bild liegt, erscheint dieses auch wenn man den Pfeil rechts von dem Bild anwählt.

## 10.3 Lightbox-Chat

Zurück zur Detaillansicht des **Lightbox-Panels**: Am rechten Rand lässt sich ein Schieber finden, welcher durch Klick den **Lightbox-Chat** ausfährt:



Der **Lightbox-Chat** ist aufgeteilt in zwei Bereiche:

1. **Teilnehmer:** Hier werden alle Teilnehmer der **Lightbox** aufgeführt, die die **Lightbox** sehen bzw. bearbeiten können.
2. **Chat:** Hier werden die Chat-Nachrichten angezeigt und der Benutzer hat die Möglichkeit, selber Nachrichten einzugeben und abzuschicken.

Der eigentliche Austausch über die kollaborative Arbeit findet hier statt. So können sich **Agency-Employees** bspw. über den Inhalt der Lightbox beratschlagen, oder es kann auch die Kommunikation zwischen **Agency-Employees** und **Customer** statt finden. Der **Customer** kann den zuständigen **Agency-Employees** klar machen, welche Art von Fotos er braucht und welche nicht.

Der **Lightbox-Chat** dockt an dem jeweiligen **Lightbox-Panel** an.

Im unteren Bereich lassen sich verschiedene Buttons finden:

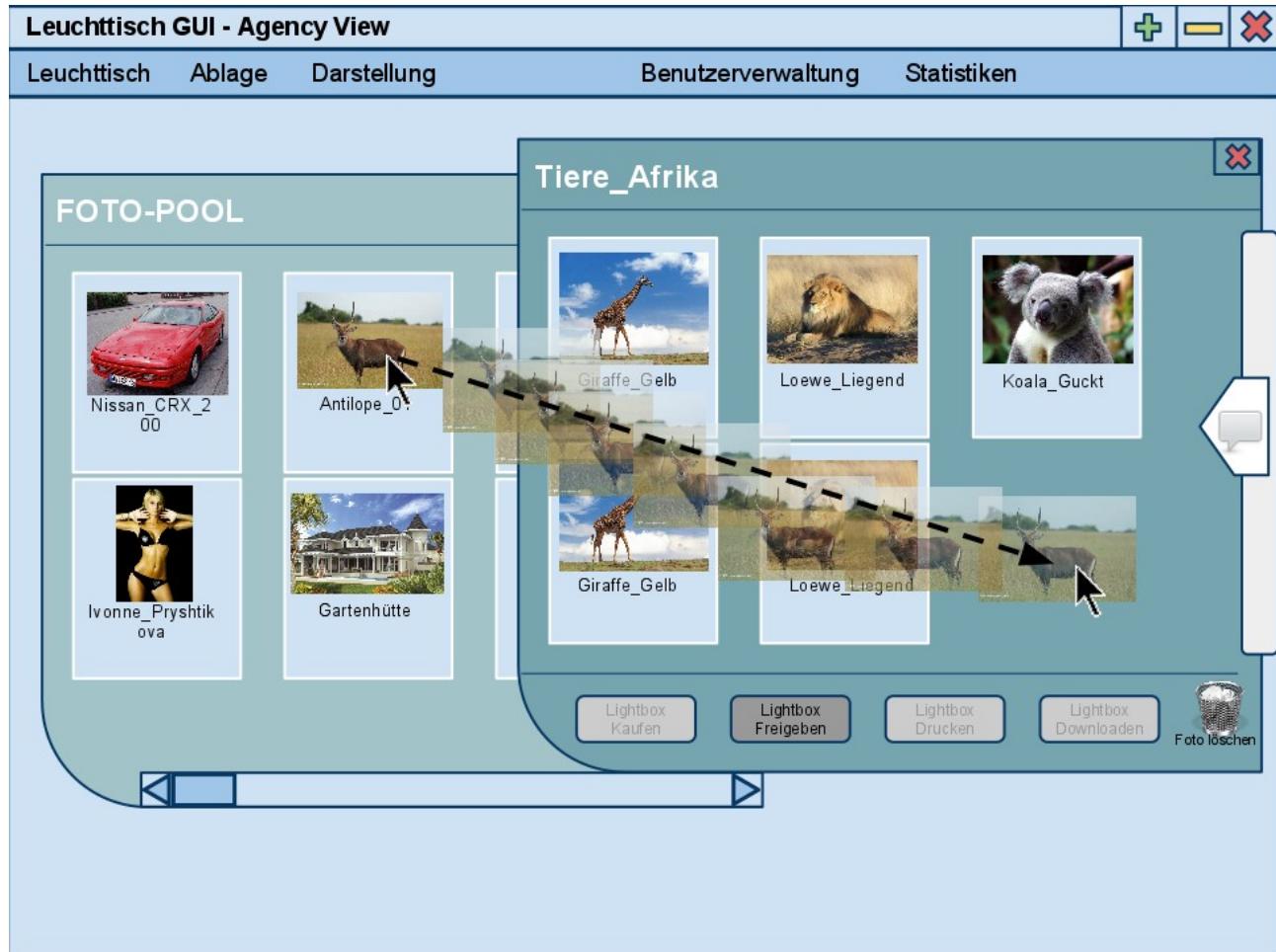
- **Lightbox kaufen:** Button nur für **Customer** klickbar.
- **Foto löschen:** Der Benutzer kann entweder ein Foto per Drag 'n Drop auf das Mülleimer-Icon ziehen, oder ein Foto selektieren und danach das Mülleimer-Icon anklicken. Das Foto ist danach gelöscht bzw. aus der Lightbox entfernt.
- **Lightbox freigeben:** Button um die Lightbox einem **Agency-Employee** oder

einem **Customer** zur Bearbeitung freizugeben. Bei Klick erscheint das Adressbuch, aus welchem ein Kontakt ausgewählt werden kann, welcher die Freigabe erhalten soll.



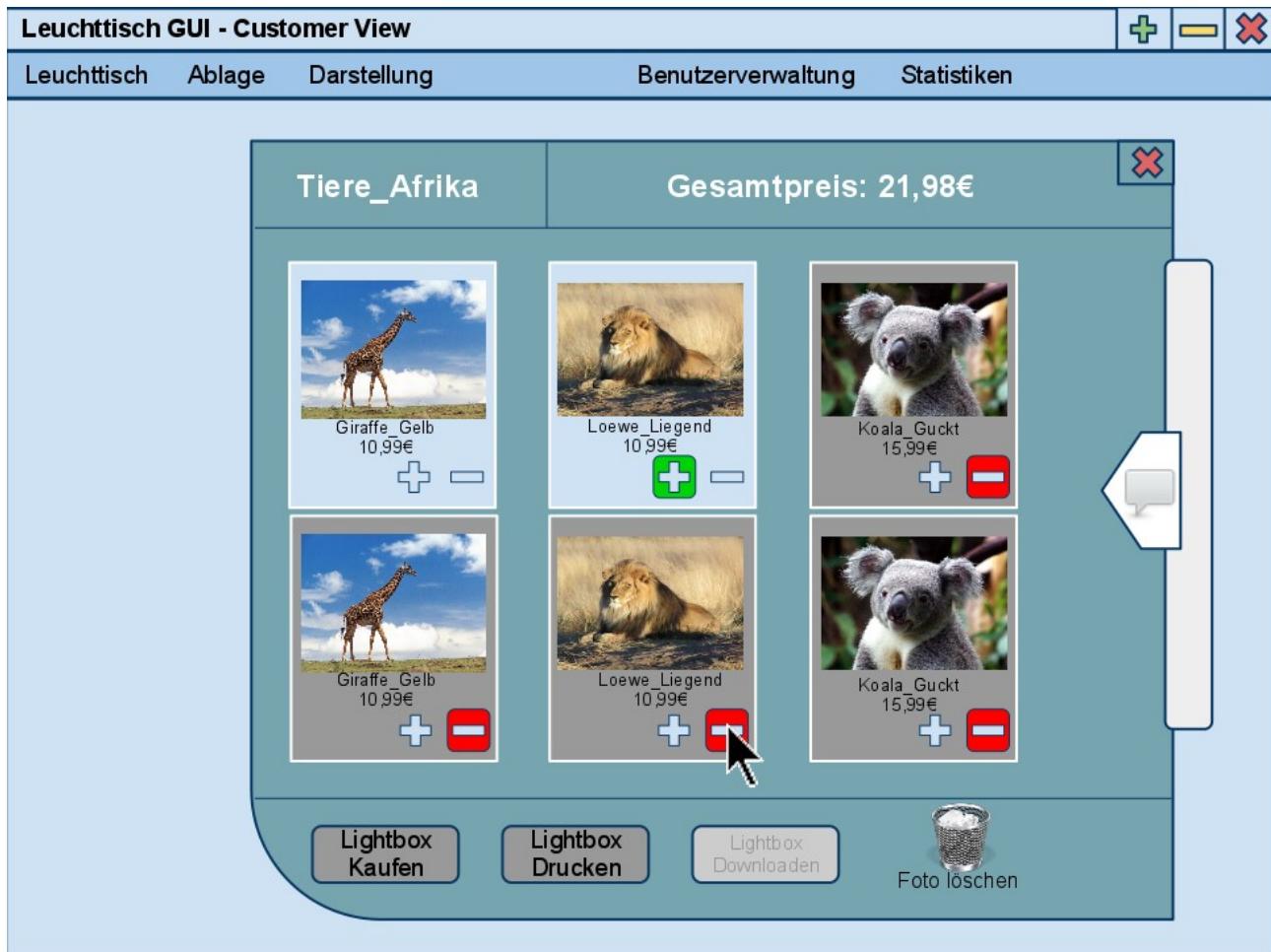
Erläutert wurde dieses Scribble bereits weiter oben.

Nach der Freigabe sieht der hinzugefügte Benutzer die **Lightbox** und ist dem **Lightbox-Chat** hinzugefügt worden. Ein **Agency-Employee** kann danach die **Lightbox** bearbeiten, sprich **Fotos** hinzufügen oder entfernen. Dieses funktioniert indem er sich aus dem **FOTO-POOL** oder der **Supersuche** per **Drag 'n Drop** Bilder holt.:



Ein **Customer** kann die **Lightbox** danach kaufen, bzw. zuerst die einzelnen Fotos **positiv** oder **negativ** bewerten. *Mehr dazu im nächsten Scribble.*

## 10.4 Customer-View



Die **Customer-View** des **Lightbox-Panels** unterscheidet sich direkt in mehreren Punkten von der **Agency-View**:

1. Im oberen Bereich steht nicht nur der Name der **Lightbox**, sondern auch der **Gesamtpreis** der vom **Customer** ausgewählten Fotos.
2. Die Rahmen der Fotos enthalten mehr, als nur den Namen des Fotos. Sie beinhalten den **Preis** des Fotos und ausserdem noch ein **Plus** für **Positive** und ein **Minus** für **Negative Bewertung**.

Was bedeutet **Positive oder Negative Bewertung**?

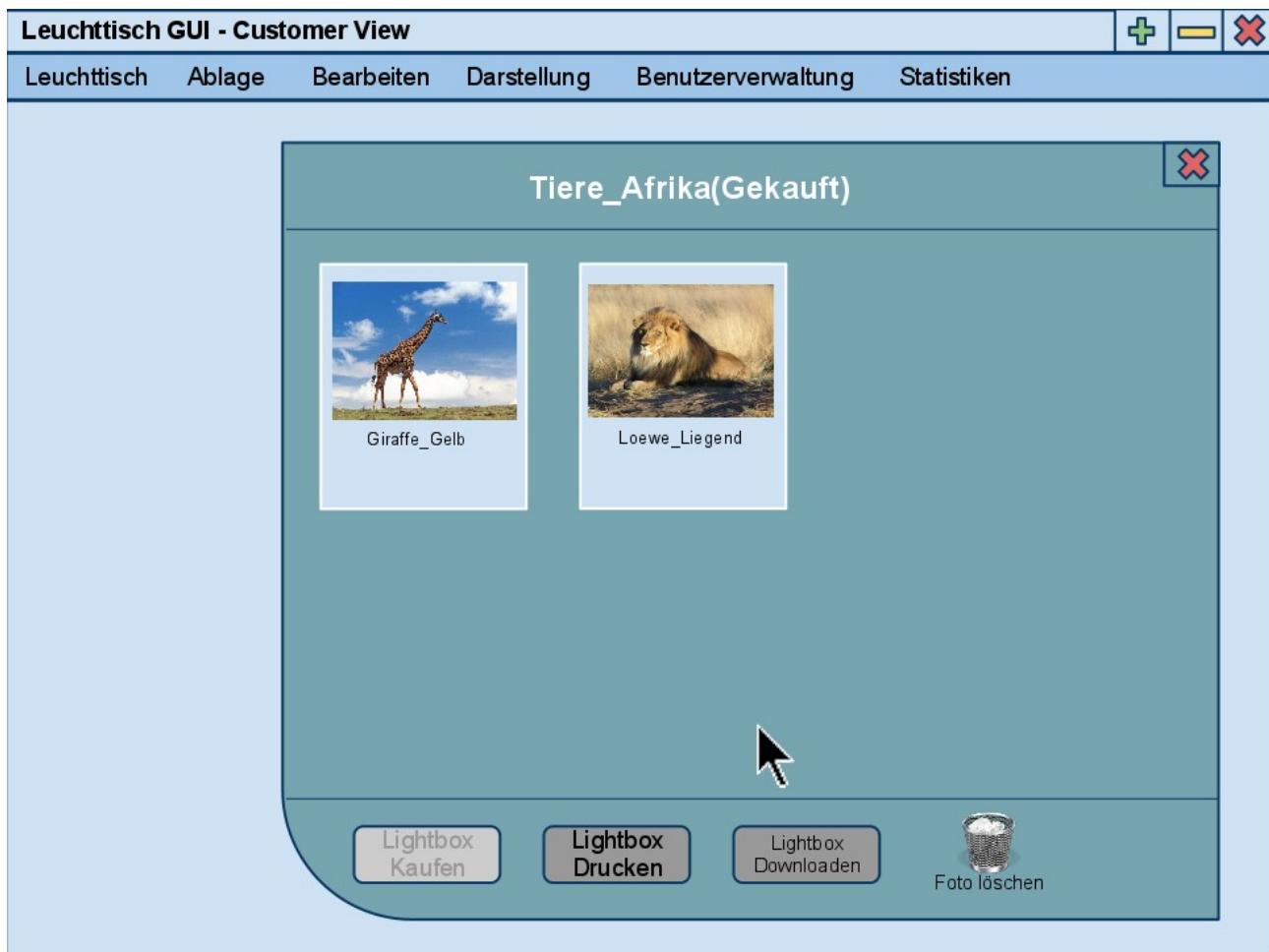
- Klickt ein **Customer** auf das **Plus**, so signalisiert er damit, dass er das Foto kaufen möchte.
- Klickt ein **Customer** auf das **Minus**, so signalisiert er damit, dass er das Foto nicht kaufen möchte. Der Rahmen verfärbt sich.

Sobald der Customer mindestens ein Foto **Positiv** bewertet/markiert hat, erscheint im oberen Bereich der **Gesamtpreis**, welcher sich aus den Preisen aller markierten Fotos zusammensetzt. Die markierten Fotos können nun gekauft werden.

3. Er hat Zugang zu mehreren Buttons:

- **Lightbox Kaufen:** Button um eine Lightbox zu kaufen. Es werden alle **Positiv** markierten Fotos gekauft.
- **Foto löschen:** Der Benutzer kann entweder ein Foto per Drag 'n Drop auf das Mülleimer-Icon ziehen, oder ein Foto selektieren und danach das Mülleimer-Icon anklicken. Das Foto ist danach gelöscht bzw. aus der Lightbox entfernt.
- **Lightbox Drucken:** Button um die Lightbox bzw. die Fotos innerhalb der Lightbox auszudrucken. Die Fotos haben nicht die originale Größe, vielmehr eine Thumbnail-Größe. Sie dienen dazu, einen schnellen Übersichtsdruck der Fotos seinen Mitarbeiter zu zeigen um diese evtl. zu besprechen. Der **Customer** erhält alle Fotos in einer **PDF-Datei**.
- **Lightbox Downloaden:** Sobald eine **Lightbox** erfolgreich gekauft wurde, ist dieser Button klickbar. Die **Lightbox** wird komprimiert heruntergeladen und der **Customer** hat Zugang zu allen gekauften Fotos in Originalgröße.

Nach dem Kauf sieht die Lightbox dann folgendermaßen aus:



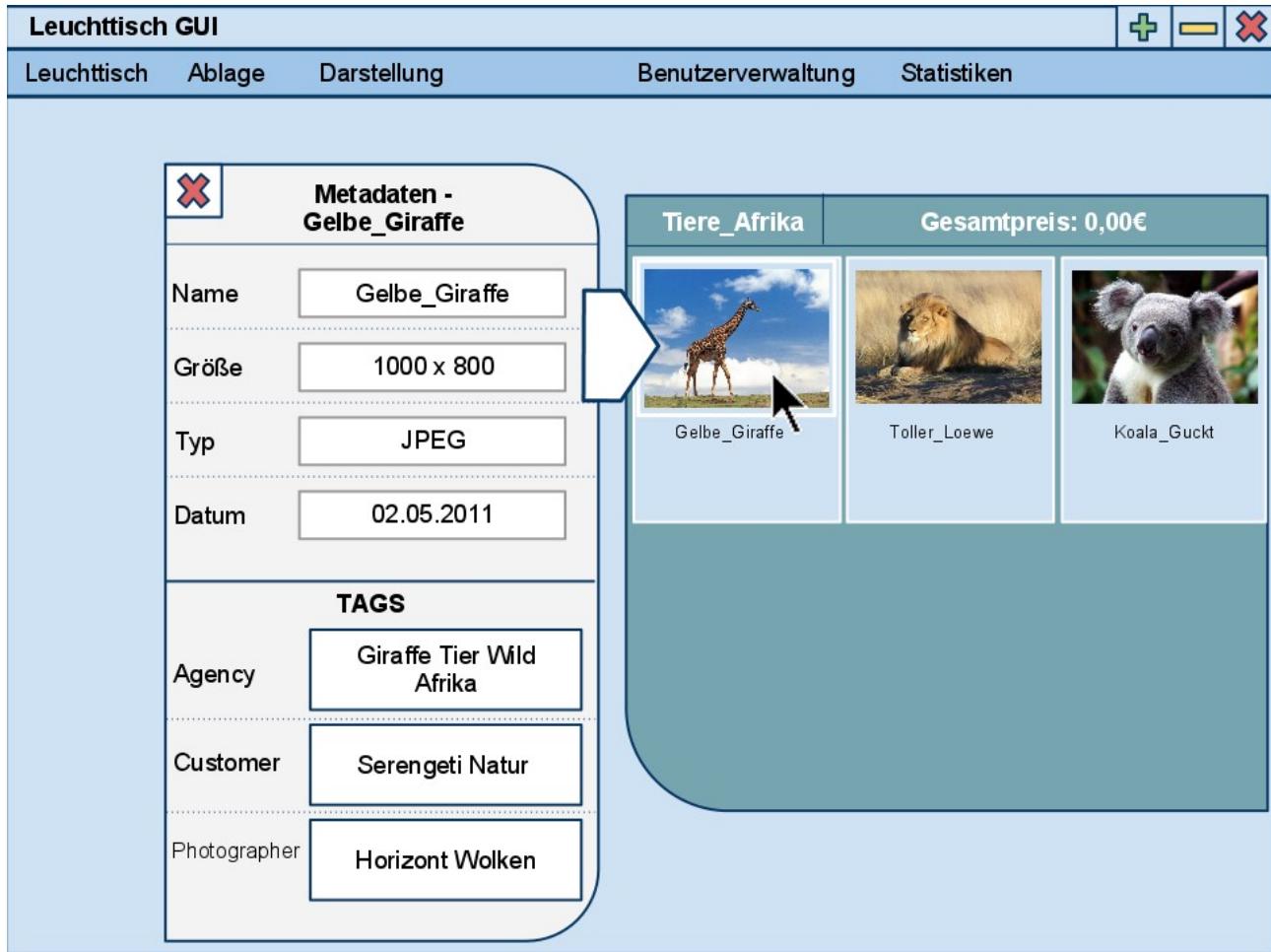
## 10.5 Photographer-View



Der **Photographer** hat eine abgespeckte Version des **Lightbox-Panels**. Er kann zwei Buttons bedienen:

1. **Lightbox Hochladen**: Button um zuvor von der Festplatte in diese **Lightbox** importierte Fotos in den **FOTO-POOL** zu laden.
2. **Foto löschen**: Der Benutzer kann entweder ein Foto per Drag 'n Drop auf das Mülleimer-Icon ziehen, oder ein Foto selektieren und danach das Mülleimer-Icon anklicken. Das Foto ist danach gelöscht bzw. aus der Lightbox entfernt.

## 10.6 Metadaten-Panel

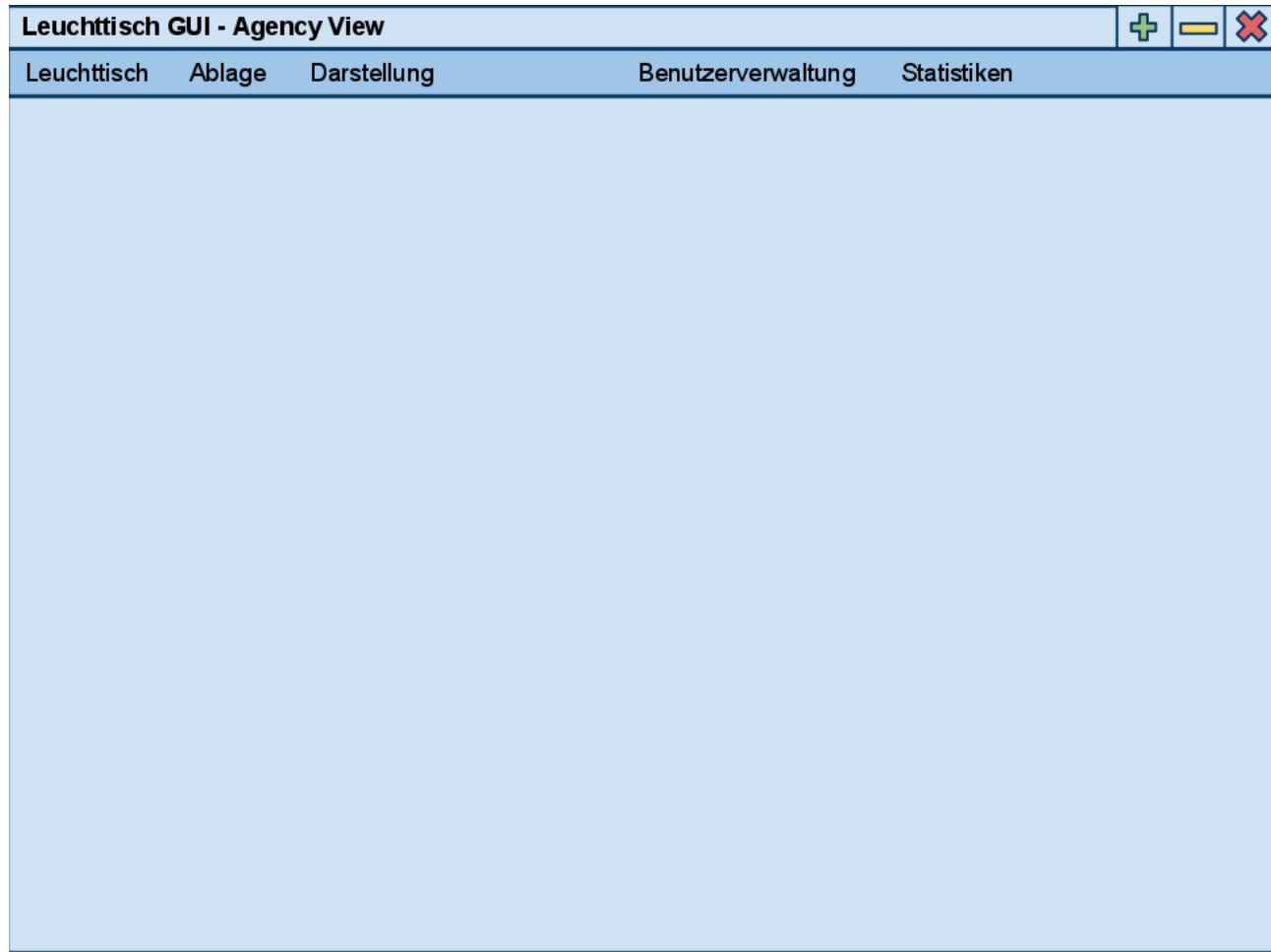


Das **Metadaten-Panel** erscheint bei einmaligem Klick auf das gewünschte Foto. Es ist in zwei Bereiche Unterteilt:

1. **Metadaten:** Hier sind Basisinformationen bezüglich des Fotos zu finden: **Name**, **Größe**, **Typ** und **Datum**.
2. **Tags:** Hier hat jeder Benutzer in dem für ihn bzw. seine Rolle vorgesehenen Feld **Tags** nach Belieben einzutragen. Die Tags dienen der besseren Lokalisierung des Fotos bei einer eventuellen Suchanfrage.

# 11 Toolbar

## 11.1.1 Agency- / Customer-View

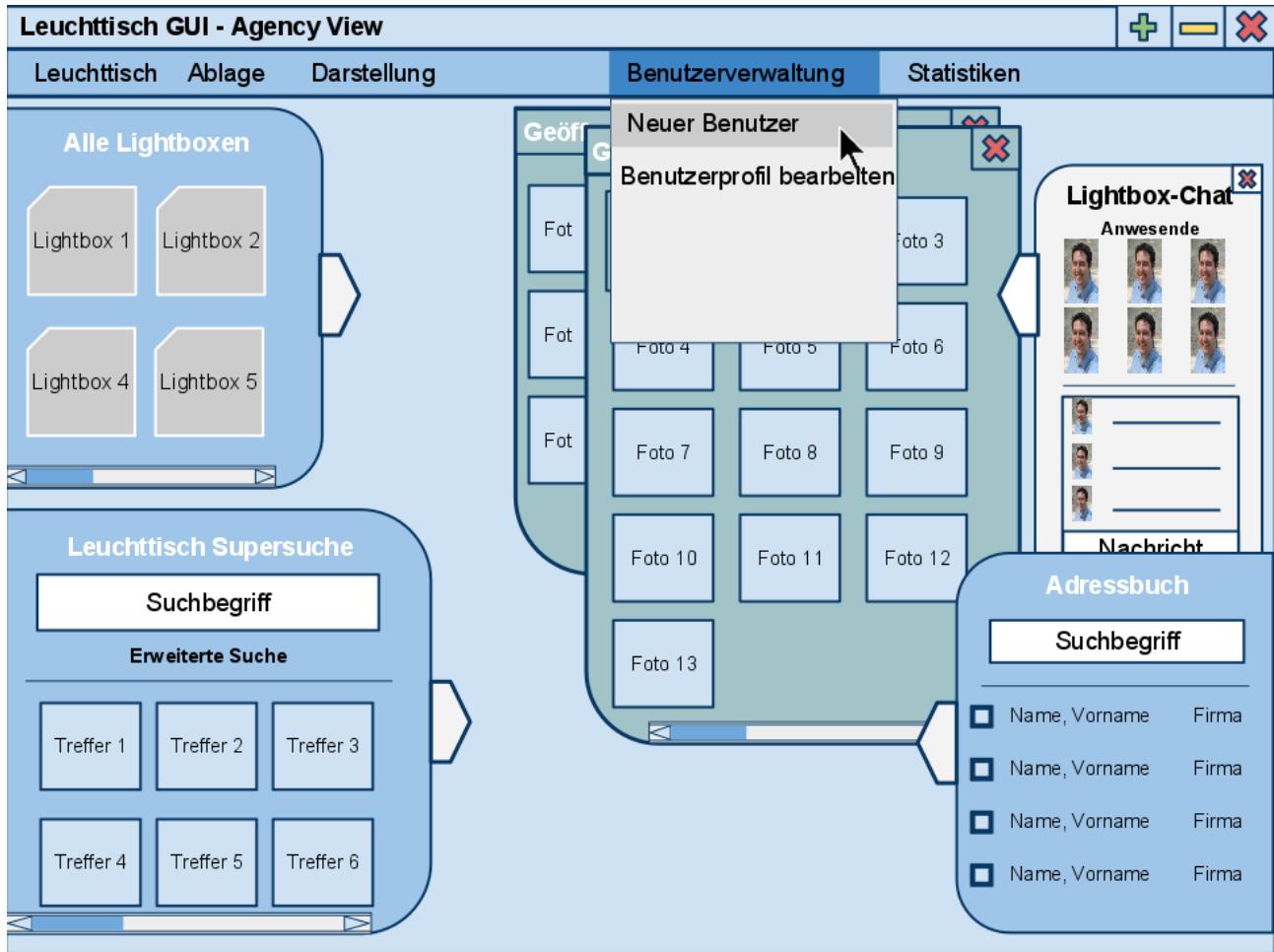


Die Toolbar bietet folgende Menüpunkte:

1. **Leuchttisch** *OPTIONAL*
2. **Ablage** *OPTIONAL*
3. **Darstellung** *OPTIONAL*
4. **Benutzerverwaltung**
5. **Statistiken**

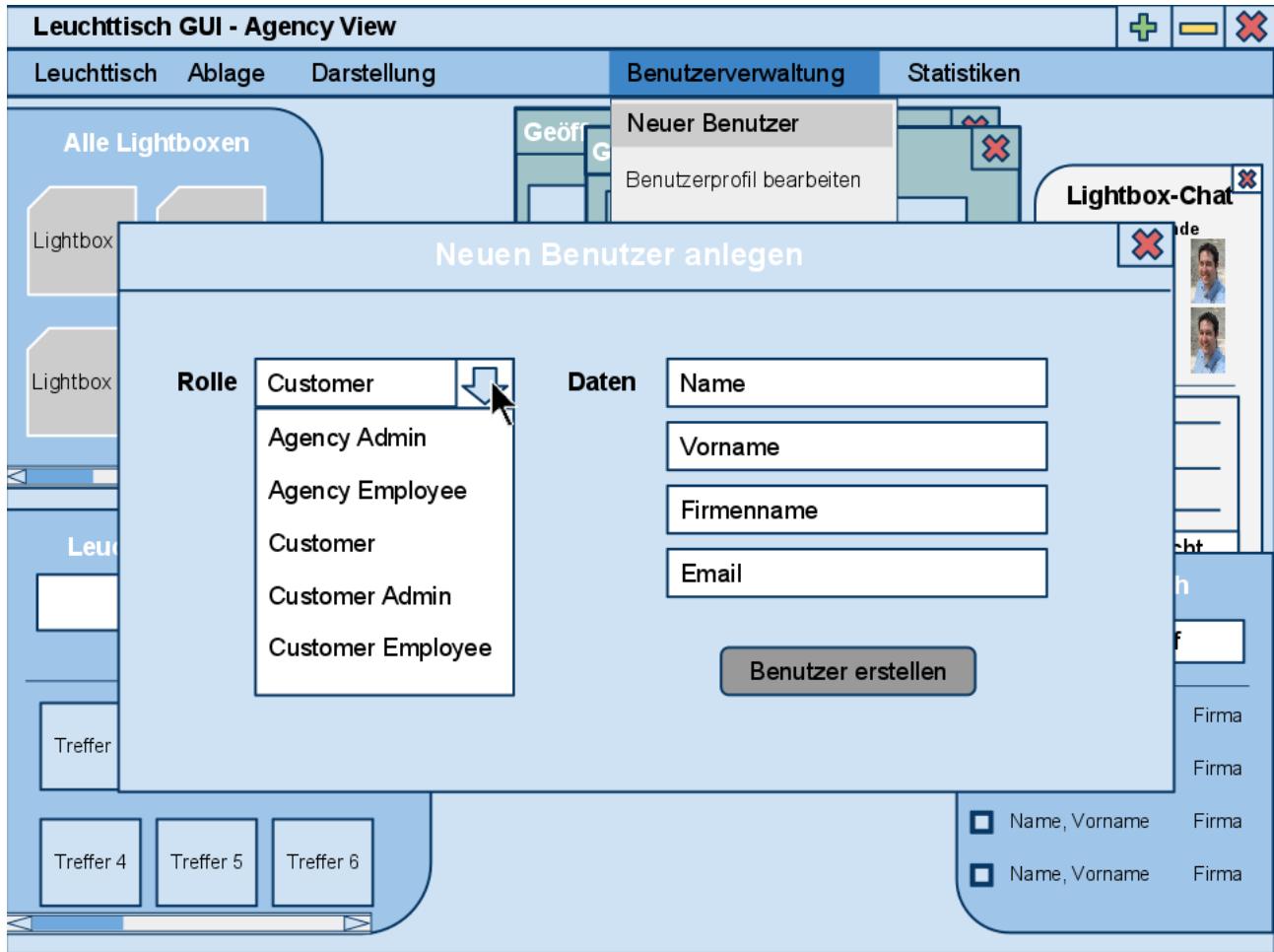
Die Gestaltung / Funktionalität der Unterpunkte der Menüpunkte 1-3 ist uns zu diesem Zeitpunkt noch nicht ganz klar, deshalb belassen wir die Implementierung dieser als *OPTIONAL*.

## 11.2 Benutzerverwaltung



Die Benutzerverwaltung ist ausschließlich den **Agency-Admins** und den **Customer-Admins** zugänglich, in allen anderen Fällen ist dieser Menüpunkt schattiert und unzugänglich.

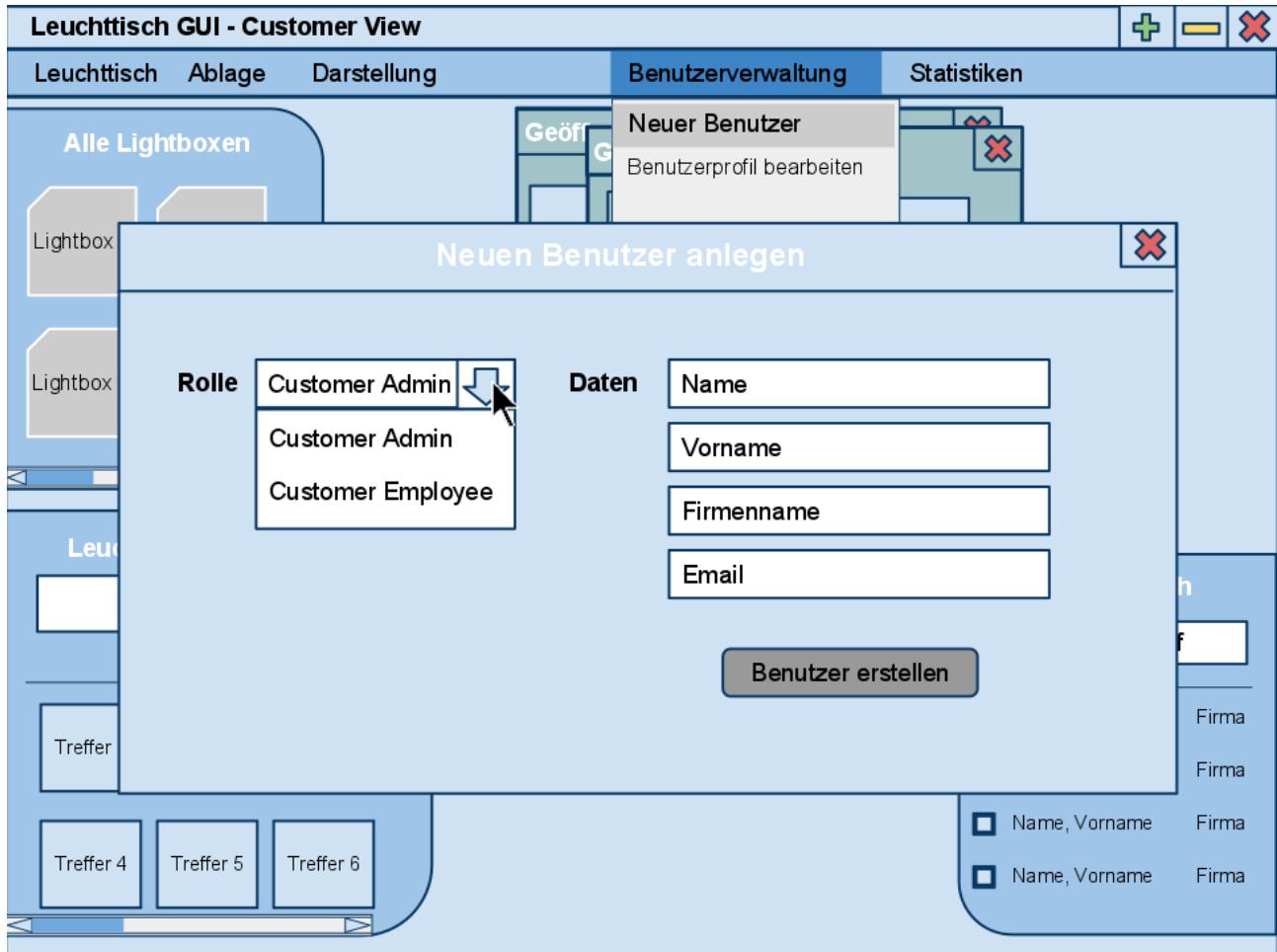
Klickt ein **Agency-Admin** auf **Neuer Benutzer** dieses Feld, so erscheint folgender Dialog:



Wie man sieht, kann ein **Agency-Admin** Benutzer in folgenden Rollen anlegen:

- **Agency Admin**
- **Agency Employee**
- **Customer** -> (Damit ist eine FIRMA/Agentur gemeint)
- **Customer Admin**
- **Customer Employee**

Klickt ein **Customer-Admin** auf **Neuer Benutzer**, erscheint der selbe Dialog mit weniger Optionen:



Ein **Customer-Admin** kann also nur Benutzer mit folgenden Rollen anlegen:

- **Customer Admins**
- **Customer Employees**

Der zweite Unterpunkt dieses Menüs ist:

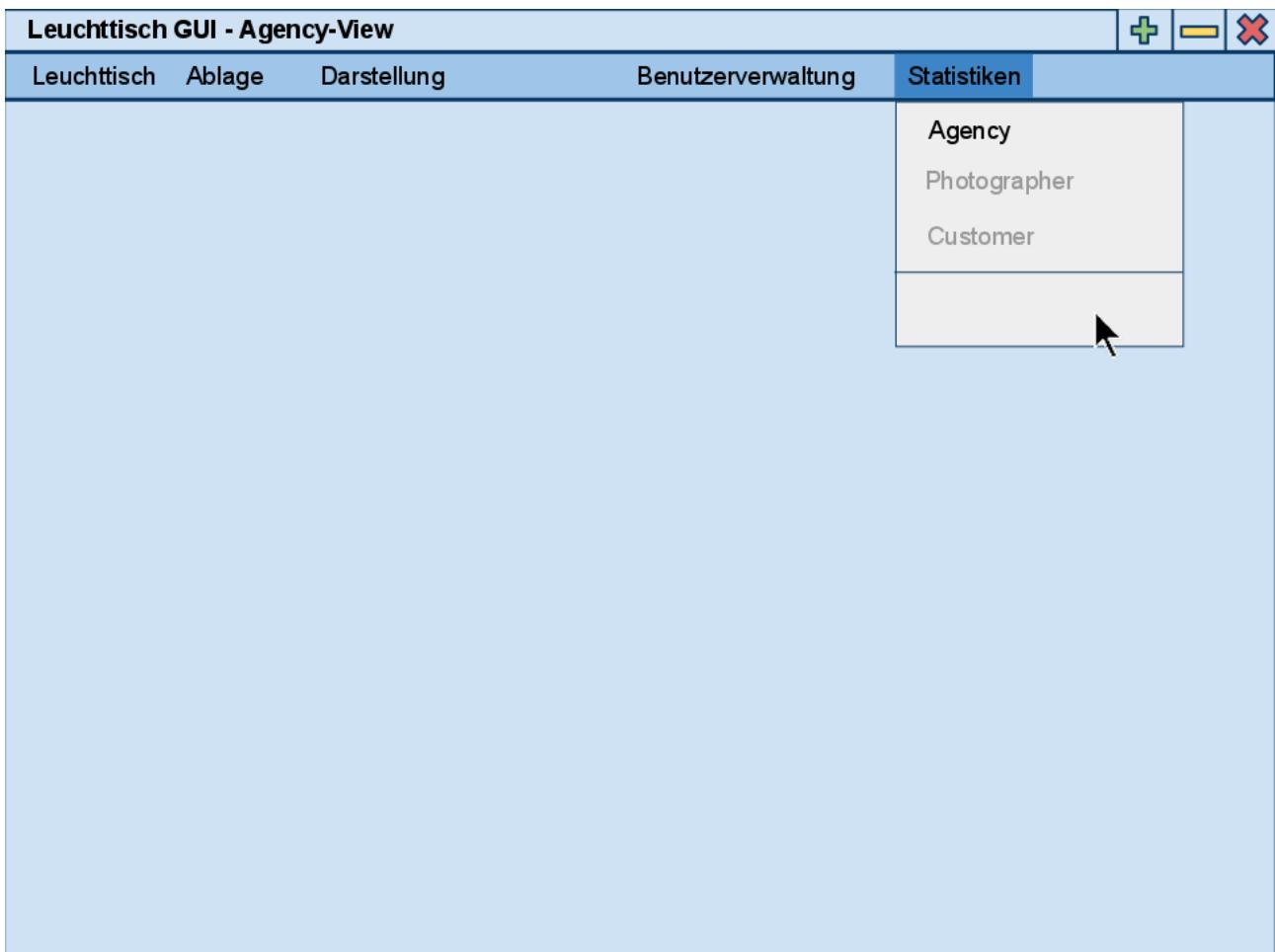
#### **Benutzerprofil bearbeiten**

Jeder Benutzer hat die Möglichkeit sein persönliches Profil zu bearbeiten. Klickt er auf die Schaltfläche **Benutzerprofil bearbeiten** erscheint folgender Dialog:



Hier können sämtliche aufgeführten Daten des jeweiligen eingeloggten Benutzers bearbeitet und abgespeichert werden.

# 12 Statistiken



Je nach Rolle des eingeloggten Benutzers, kann einer der drei Unterpunkte angewählt werden.

## 12.1 Agency-View

Es öffnet sich nach Klick auf die Schaltfläche **Agency** folgendes Panel:

Statistik	Wert	Trend (Vergleich Vormonat)
Umsatz	12.323.54€	+20,5%
Anzahl angebotene Fotos	321	+5,4%
Anzahl verkauft Fotos	190	-4,4%
Anzahl neue Customer	10	+0,0%
Anzahl neue Photographer	5	-15,4%

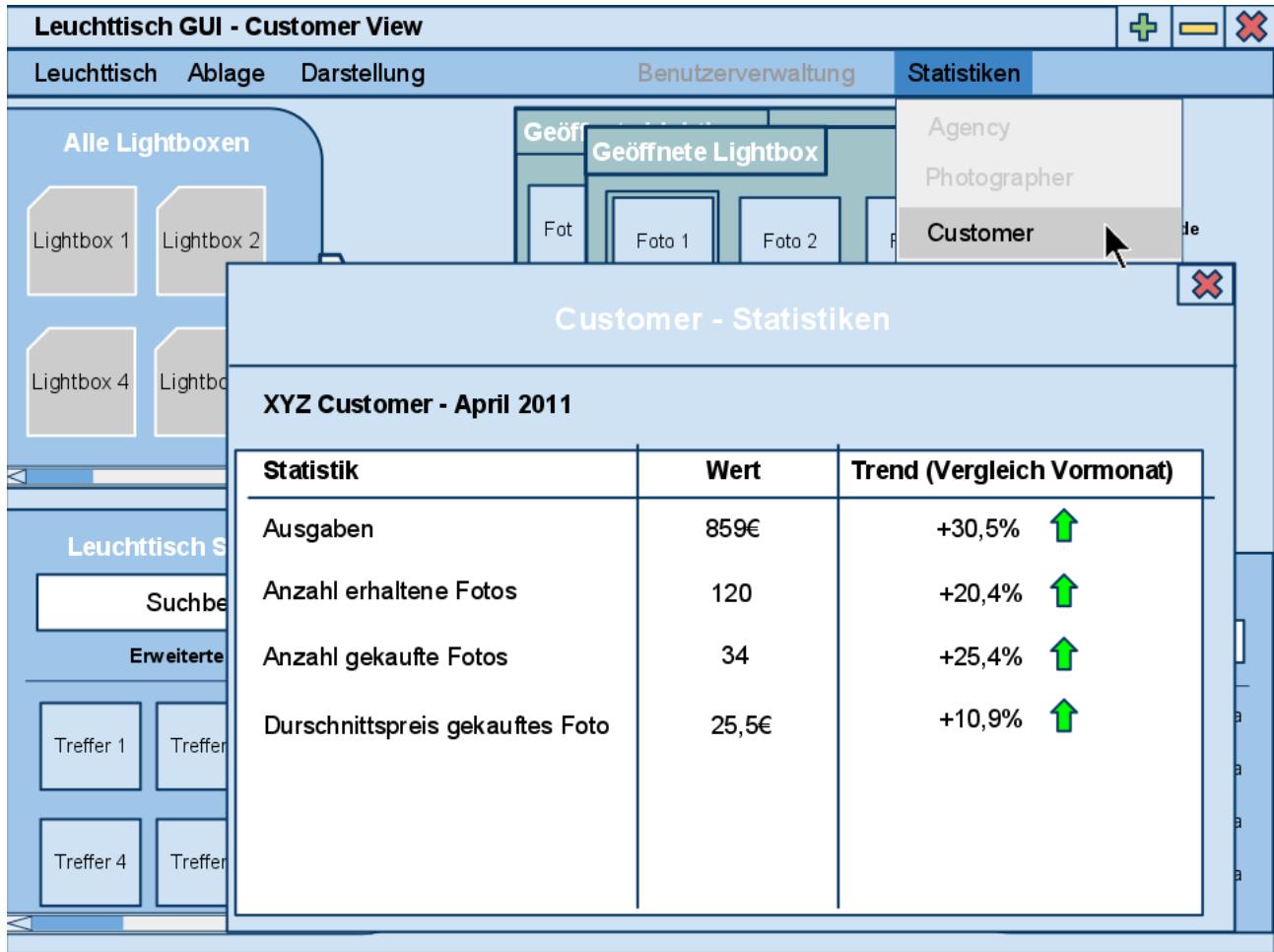
Hier sind alle relevanten oder interessanten **Statistiken** der **Agency** - hauptsächlich wirtschaftlicher Natur - des aktuellen Monats aufgeführt:

- **Umsatz**
- **Anzahl angebotene Fotos**
- **Anzahl verkauft Fotos**
- **Anzahl neue Customer**
- **Anzahl neue Photographer**

Daneben lassen sich die Zahlen zu den einzelnen Statistiken finden. Am rechten Rand wird ein Trend, welcher in Relation zu den Werten aus dem Vormonat steht, angezeigt. Dieser kann Positiv, Negativ, oder Neutral sein.

## 12.2 Customer-View

Es öffnet sich nach Klick auf die Schaltfläche **Customer** folgendes Panel:

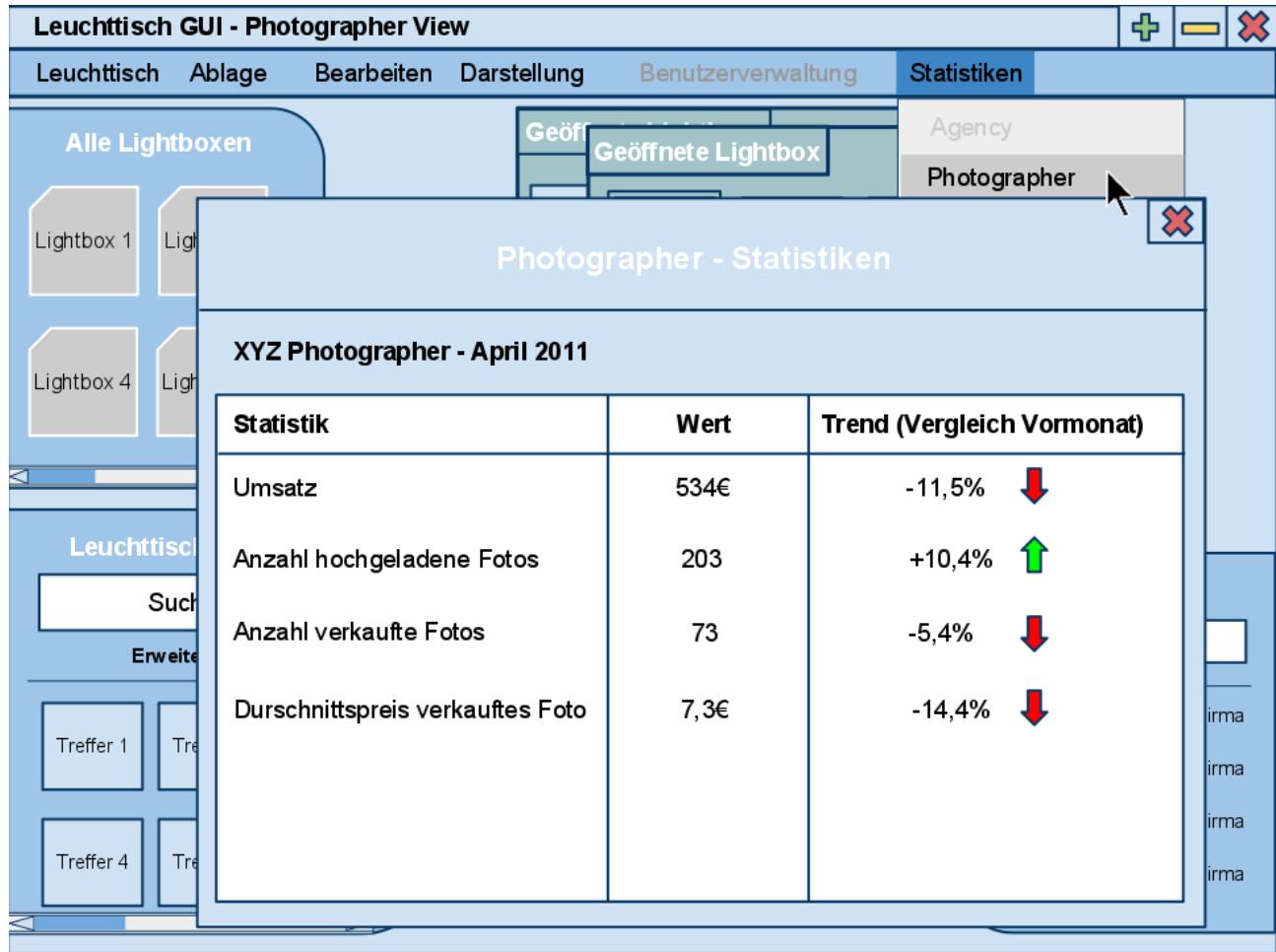


Hier sind alle relevanten oder interessanten **Statistiken** des **Customer** des aktuellen Monats aufgeführt:

- **Ausgaben**
- **Anzahl erhaltene Fotos**
- **Anzahl gekaufte Fotos**
- **Durchschnittspreis gekauftes Foto**

## 12.3 Photographer-View

Es öffnet sich nach Klick auf die Schaltfläche **Photographer** folgendes Panel:



Hier sind alle relevanten oder interessanten **Statistiken** des **Photographer** des aktuellen Monats aufgeführt:

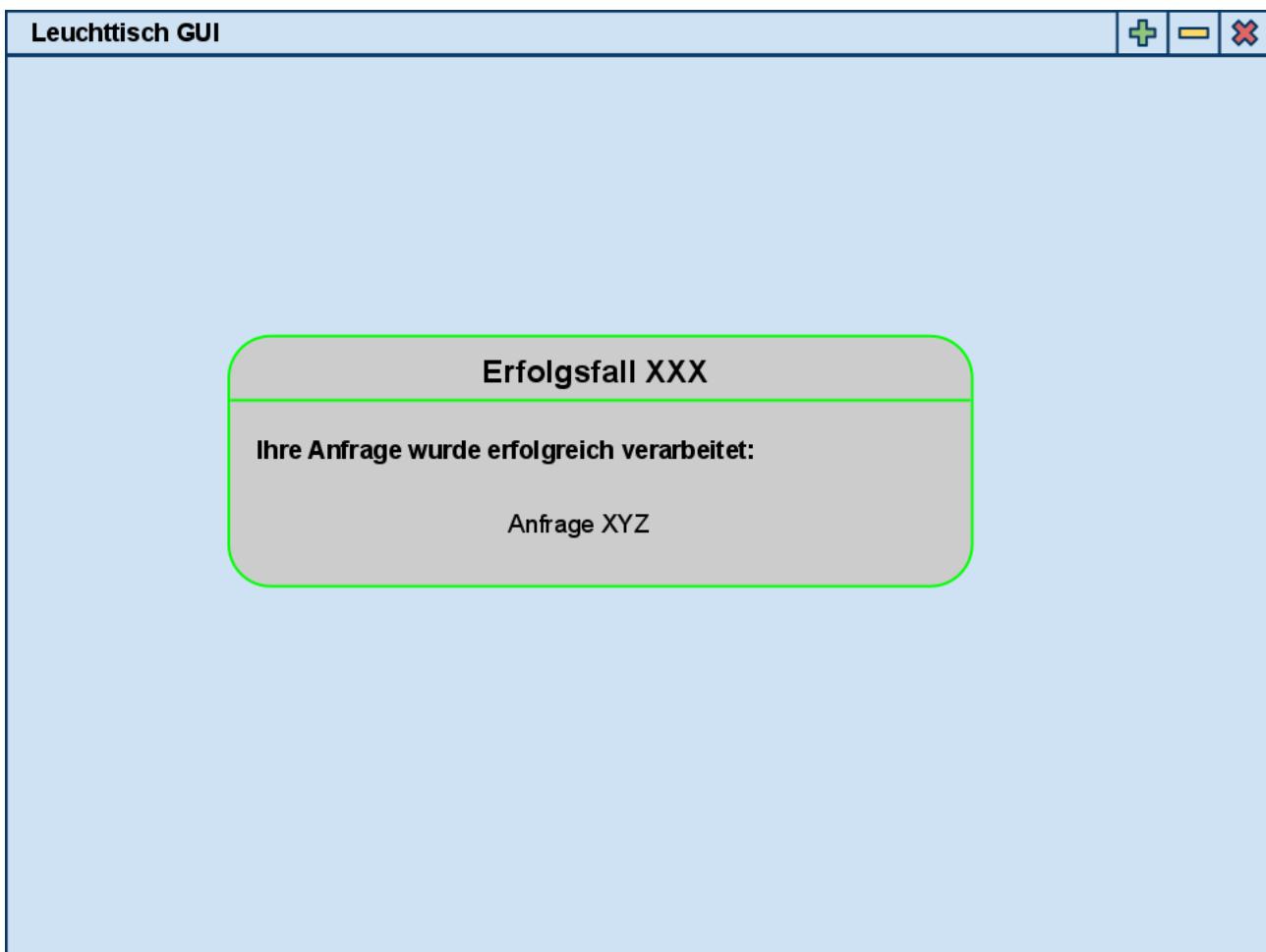
- **Umsatz**
- **Anzahl hochgeladene Fotos**
- **Anzahl verkaufte Fotos**
- **Durchschnittspreis verkauftes Foto**

# 13 Dialogboxen

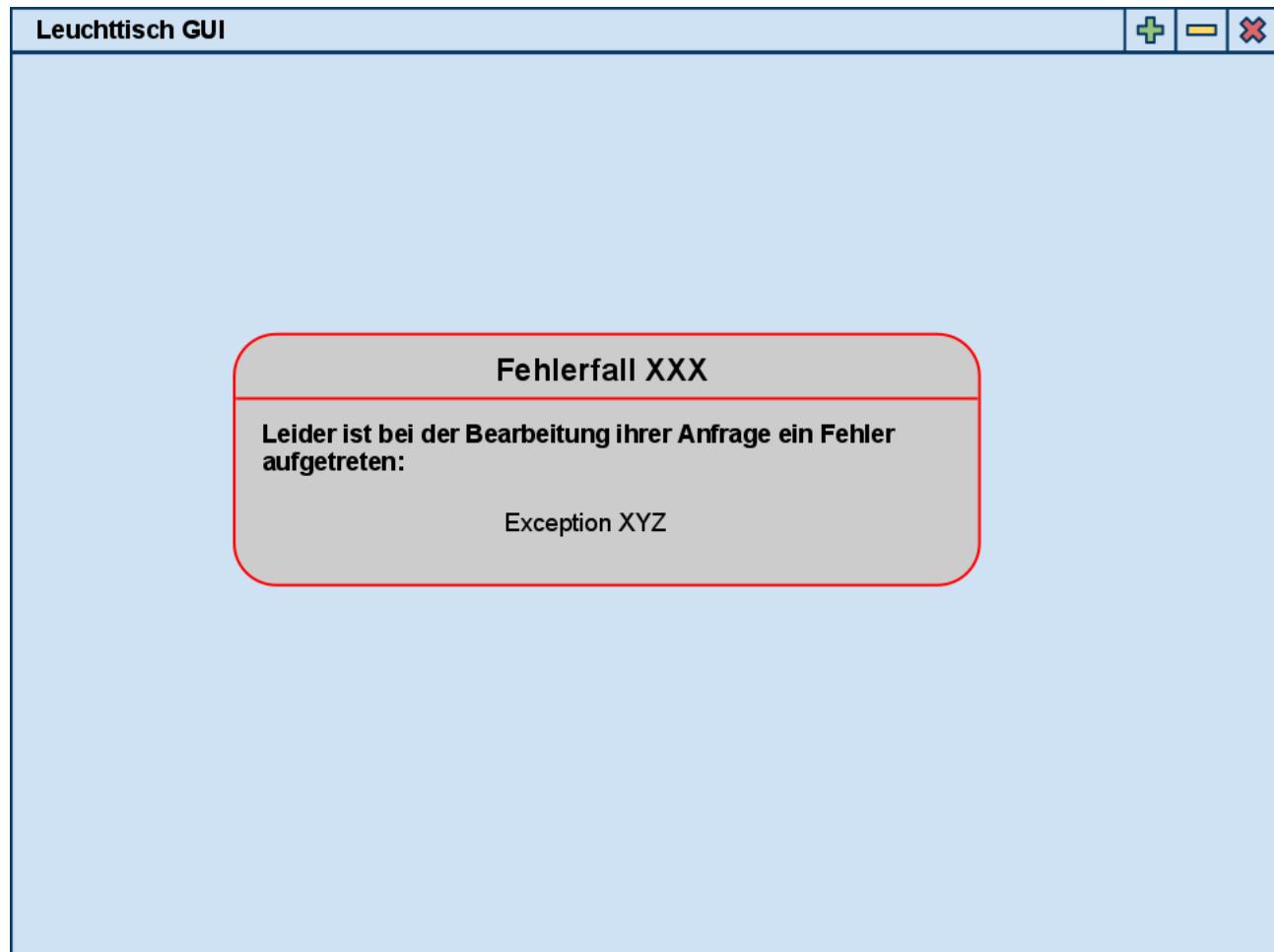
## 13.1 Erfolg

Da bei der Client bei fast jeder Aktion seine Daten vom Server bezieht, ist es an manchen Stellen natürlich wünschenswert, dem Benutzer mit zu teilen, ob seine [Anfrage/Aktion?](#) erfolgreich war, oder nicht. Deshalb wird es unterschiedliche Dialogboxen geben, welche dem Benutzer auf seine Aktion Rückmeldung geben.

## 13.2 Erfolgsfall



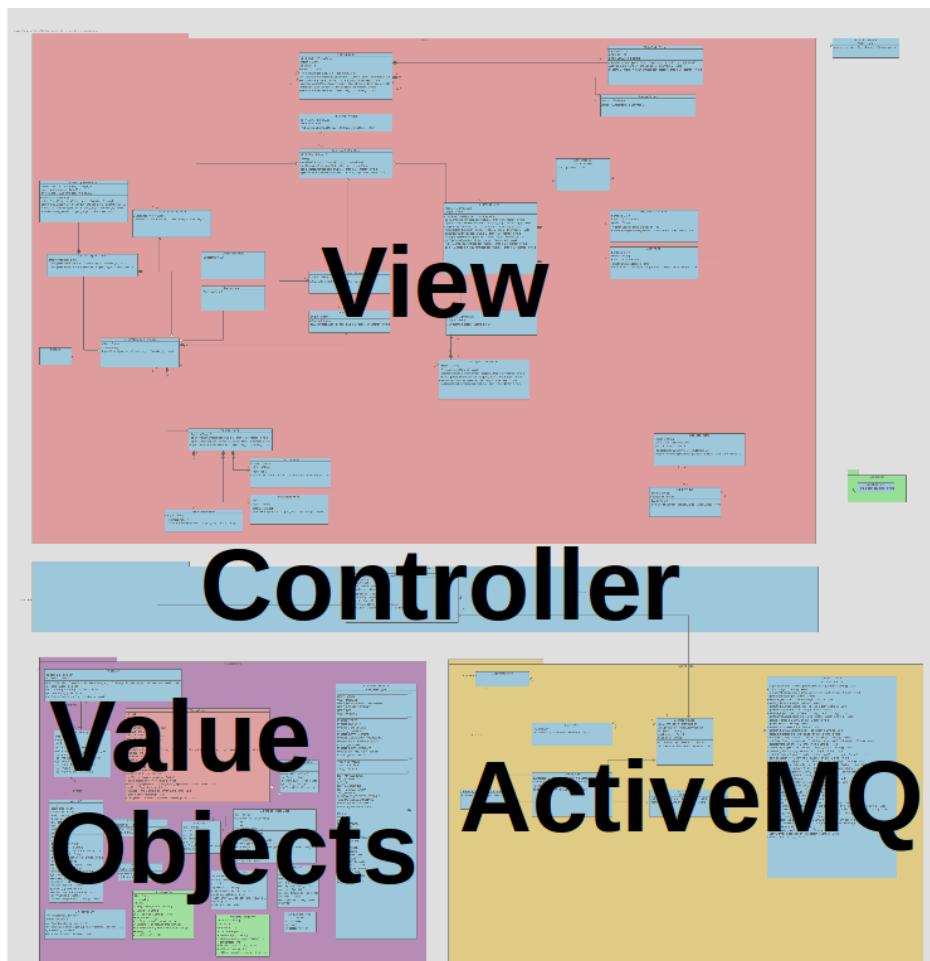
## 13.3 Fehlerfall

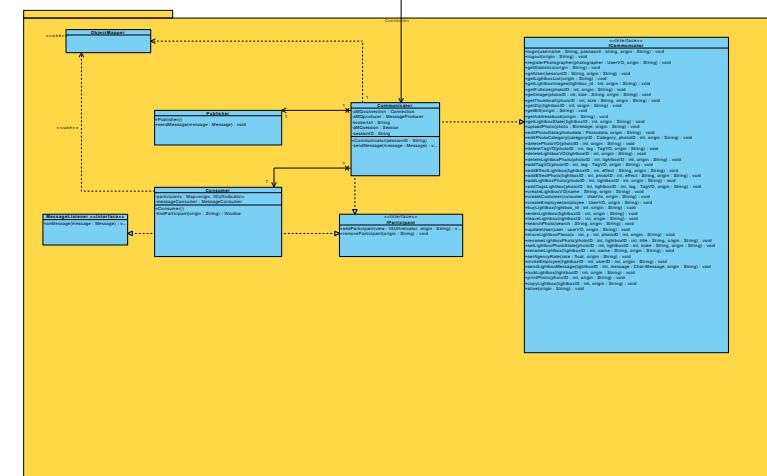
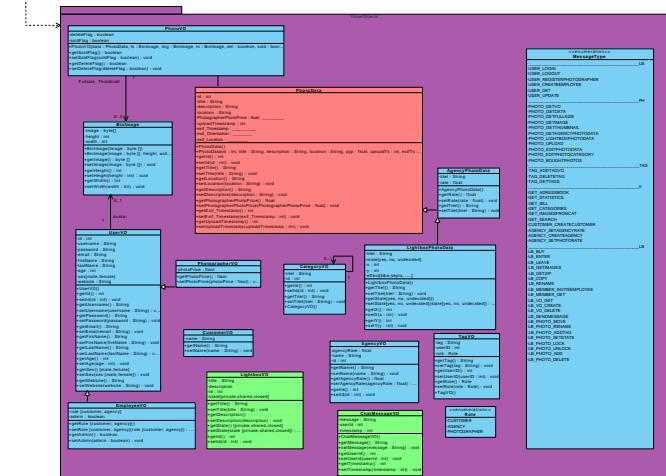
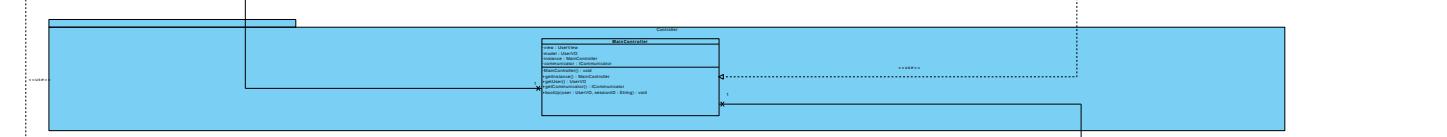
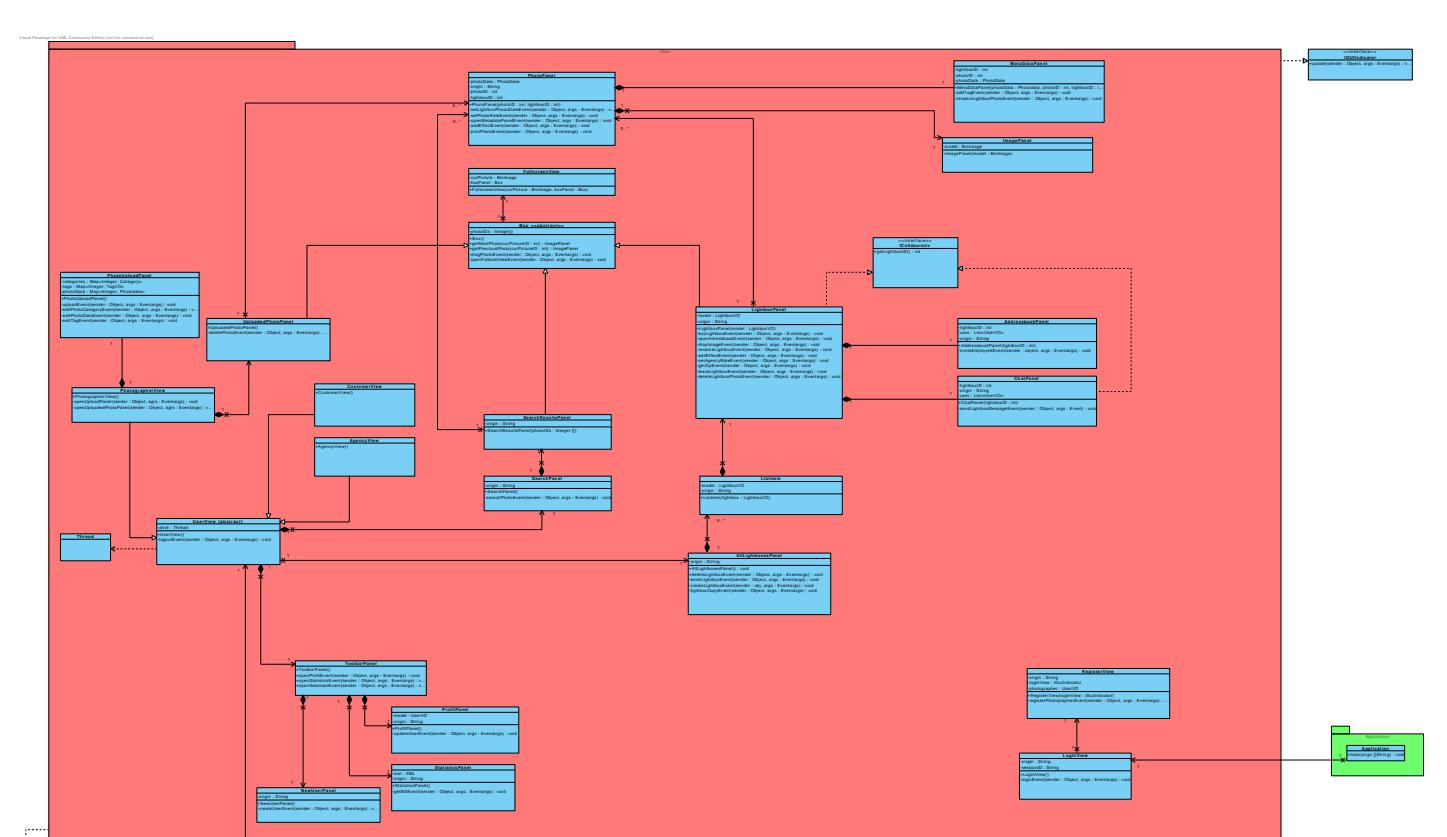




## Aufbau Client

siehe separater Ausdruck







# Architekturbeschreibung Client

Anhang zum Leuchttisch Designdokument

Dieser Angang beschreibt den Aufbau der Architektur des Client.

## Inhaltsverzeichnis

1 package: Application.....	2
1.1 Application.....	2
2 package: View.....	2
2.1 IGUILIndicator.....	2
2.2 ICollaborativ.....	3
2.3 Login-View.....	3
2.4 RegisterView.....	4
2.5 PhotoUploadPanel (Photographer).....	4
2.6 PhotographerView.....	5
2.7 UploadedPhotoPanel.....	5
2.8 UserView{abstract}.....	5
2.9 AgencyView.....	6
2.10 CustomerView.....	6
2.11 ToolbarPanel.....	6
2.12 NewUserPanel.....	6
2.13 ProfilPanel.....	6
2.14 StatisticsPanel.....	7
2.15 Box {abstract}.....	7
2.16 FullscreenView.....	7
2.17 SearchPanel.....	8
2.18 SearchResultsPanel.....	8
2.19 PhotoPanel.....	8
2.20 MetaDataPanel.....	9
2.21 ImagePanel.....	9
2.22 LightboxPanel.....	10
2.23 AddressbookPanel.....	10
2.24 ChatPanel.....	11
2.25 Listitem.....	11
2.26 AllLightboxesPanel.....	11
3 package: ValueObjects.....	12
3.1 PhotoVO.....	12
3.2 PhotoData.....	12
3.3 BinImage.....	13
3.4 AgencyPhotoData.....	13
3.5 LightboxPhotoData.....	13
3.6 LightboxVO.....	14
3.7 ChatMessageVO.....	14
3.8 CategoryVO.....	14
3.9 TagVO.....	14
3.10 UserVO.....	14
3.11 PhotographerVO.....	15
3.12 EmployeeVO.....	15
3.13 AgencyVO.....	15
3.14 CustomerVO.....	15

3.15 Role (Enum).....	16
3.16 MessageType.....	16
4 package: Controller.....	16
4.1 MainController.....	16
5 package: Connection.....	16
5.1 ICommunicator <<Interface>>.....	16
5.2 Communicator.....	17
5.3 Publisher.....	17
5.4 Consumer.....	17
5.5 IParticipant <<Interface>>.....	18
6 Glossar.....	18

# 1 package: Application

Das Application-Package beinhaltet eine Klasse, welche die Main-Methode des Clients enthält. Verantwortlich ist dieses Package einzig und allein für die Instanziierung des Login-Panels aus dem View-Package. Der nachfolgende Ausschnitt soll einen schnellen Überblick über die Richtung der Kommunikation geben.

## 1.1 Application

Die Klasse Application hat nur eine Aufgabe, welche darin besteht die GUI, bzw. das Login-Panel / die Login-View zu instanzieren. Ist dieses geschehen, ist auch die Arbeit dieser Klasse für die restliche Laufzeit des Programms erledigt.

Attribute:

- Keine

Methoden:

- public void main(String args[]): Funktion wird beim ersten Ausführen des Programms aufgerufen. Es werden keine Argumente mitgegeben.

# 2 package: View

In diesem Abschnitt werden die wichtigsten Klassen aus der Präsentationsschicht, dem View-Package, näher beschrieben. Das Packages ist für die Darstellung der Daten verantwortlich.

Es gibt eine Vielzahl von verschiedenen Panels (=Überbegriff für Fenster), welche jeweils bestimmte Aufgaben übernehmen. Beschreibung im Folgenden.

## 2.1 IGUILIndicator

Alle Anzeigeklassen implementieren das Interface, damit alle die update()- Methode umsetzen und über dieses Interface identifizierbar sind.

Methoden:

- public void update(Object obj, String messageType): Die Methode bekommt vom Consumer ein Object rein, welches je nach MessageType gecastet wird. Bietet Schnittstelle für Änderungen am Zustand der Klasse.

## 2.2 ICollaborativ

Damit man Antworten vom Server, IGuiIndicatoren zuordnen kann, welche keine Anfrage geschickt haben, können diese durch dieses Interface schnell gefunden werden. Hierbei handelt es sich um: Chat und Lightbox.

## 2.3 Login-View

Die Login-View ist für die Darstellung der Login-Maske zuständig und repräsentiert ein Fenster in dem der Benutzer seine Login-Daten eingeben und absenden kann. Desweiteren hat der Benutzer die Möglichkeit sich als Photographer zu registrieren.

Detaillierte Beschreibung vom Server zum Server am Beispiel des LoginViews: (ist für jeden Fall äquivalent, ist auch ersichtlich über die Sequenzdiagramme)

**Weg der Kommunikation zum Server:** Die Login-View bedient sich der Singleton Main-Controller-Klasse aus dem Controller-Package, um mit dem Connection-Package in Verbindung zu treten.

1. über die Communicator Klasse : Login-View meldet sich als Teilnehmer an über addParticipant(this, origin)
2. Communicator Klasse leitet die Anmeldung an den Consumer weiter, welcher die Liste mit Teilnehmern hält
3. Login-Daten werden der login(username, passwort, origin)- Methode aus der Communicator Klasse übergeben, welche die Message zusammenbaut und Kommunikation mit dem Server anstößt (durch die Publisher Klasse)

### Weg der Kommunikation vom Server:

Die Consumer Klasse, welche von der Communicator Klasse instanziert wird, empfängt Messages vom Server über das Topic. Eine Message besteht aus Properties und Body. Die Properties bestehen wiederum aus Origin, MessageType und SessionID. Mithilfe der Teilnehmerliste (Participants) kann die Klasse den zugehörigen Teilnehmer finden, um diesen zu benachrichtigen.

1. die Message wird ausgepackt: JSON String TO Objekt mithilfe des ObjectMappers
2. das Objekt und der MessageType werden über die Update-Methode des Login-Views übergeben ==
3. mithilfe der global bekannten MessageTypes (<<Enumeration>>) kann sich die View den Richtigen heraussuchen und das Object casten
4. Response vom Server: False oder die SessionID
  1. False: User wird benachrichtigt
  2. SessionID : speichere diese als Attribut

Danach hat der LoginView die SessionID und durchläuft den gleichen Prozess nur mit dem

ICommunicator Aufruf: getUser(sessionID, origin), um alle Daten vom User zu bekommen. (vor allem die Rolle des Users). Dieser Aufruf geschieht automatisch, der User hat keine Interaktion zu leisten.

Anmerkung: Ein View kann sich auch wieder abmelden, wenn auf keine Antwort mehr warten (removeParticipant(origin)).

Attribute:

- private String origin
- private String Session-ID: Wird hier abgelegt, weil die LoginView nach der Login-Anfrage auf das UserVO wartet. Wenn der UserVO eintrifft, benötigt die LoginView die SessionID, um den MainController hochzufahren (bootUp(sessionID, UserVO)).

Methoden:

- public LoginView(): Konstruktor
- private void loginEvent(Object sender, EventArgs args): Methode wird aufgerufen, sobald der „Senden“ Button in der Login-View geklickt wurde. Die Methode liest die eingegebenen Daten aus: Username, Passwort

## 2.4 RegisterView

Die Registrierungsmaske für die Fotografen.

Attribute:

- private String origin
- private IGUILIndicator loginView: Wenn die Regisitrierung erfolgreich war, schließt der RegisterView den LoginView, da der Photographer direkt eingelogged wird.
- private UserVO photographer: Der sich registrierende Benutzer.

Methoden:

- public Konstruktor RegisterView(loginView: IGUILIndicator): Konstruktor
- private void registerPhotographerEvent(sender: Object, args: EventArgs): Startet den Registrieungsvorgang.

## 2.5 PhotoUploadPanel (Photographer)

Das PhotoUploadPanel beinhaltet die visuelle Umsetzung: Fotos hochladen, Foto Metdaten eintragen/ändern, Kategorie auswählen, taggen. Wenn der User über den Filebrowser die Bilder auswählt, um diese hochzuladen, werden diese direkt an den Server geschickt. Die Attribute sind dafür da, falls ein Upload fehlschlägt, die Daten temporär gespeichert werden und der Upload erneut ausgeführt werden kann.

Attribute:

- private Map<Integer, Category> categories: Zur Bildkategorisierung benötigt.
- private Map<Integer, TagVO> tags: Speichert die Tags von Bildern.

- Private Map<Integer, Photodata> photoData: Speichert die Informationen zum Bild(Metadaten)

Methoden:

- public Konstruktor PhotoUploadPanel(): Konstruktor.
- private void uploadEvent(sender : Object, args : Eventargs) : Leitet Upload einFilebrowser zum Auswählen von Fotos
- private void editPhotoCategoryEvent(sender : Object, args : Eventargs) : speichert die Kategorie der Bilder.
- private void editPhotoDataEvent(sender : Object, args : Eventargs) : speichert das Ändern der Metadaten eines Fotos.
- private void editTagEvent(sender : Object, args : Eventargs): speichern der Tags für Bild.

## 2.6 PhotographerView

Wenn der User ein Photographer ist, so wird dieser View instanziert. Die Klasse legt die für den Photographer sichtbaren Elemente fest. Diese Klasse erbt von UserView.

Methoden:

- private void openUploadPanel(sender: Object, args : Eventargs): Blendet das Upload Fenster ein, wenn der Entsprechende Button gewählt wurde.
- private void openUploadedPhotoPanel(sender: Object, args: Eventags): Blendet ein Fenster ein welchem alle Fotos enthalten sind, die ein Fotograf hochgeladen hat.

## 2.7 UploadedPhotoPanel

Zeigt die bereits hochgeladen Fotos an.

Methoden:

- public UploadedPhotoPanel(): Konstruktor
- private void deletePhotoEvent(sender: Object, args: Eventargs): Löscht ausgewähltes Photo

## 2.8 UserView{abstract}

Die Basisklasse für die gemeinsamen Eigenschaften aller Benutzer. (Lightbox, Toolbar etc.)

Attribute:

- private Thread alive: in einem bestimmten Zeitintervall wird eine Alive-Message an den Server geschickt. Dies hat den Zweck: Falls der User z.B. seine Verbindung zum Server verliert. Der Server würde dann merken, dass er keine Alive- Messages mehr bekommt und wird den User ausloggen.
- private logoutEvent(sender: Object, args: Eventargs): wird aufgerufen, wenn der

User den Schließen-Button klickt. Der Server wird benachrichtigt. Das Programm wird erst ordnungsgemäß beendet, wenn eine erfolgreiche Bestätigung vom Server zurück kommt. Falls keine positive Nachricht zurück kommt, wird der User benachrichtigt.

## 2.9 AgencyView

Die Ansicht der Agentur wird in dieser Klasse gespeichert. Diese Klasse erbt von UserView.

- private void update(sender : Object, args Eventargs): Beschreibung wie oben

## 2.10 CustomerView

Die Ansicht des Kunden wird von dieser Klasse gespeichert.

- private void update(sender : Object, args Eventargs): Beschreibung wie oben

## 2.11 ToolbarPanel

Die Menüleiste sieht jeder Benutzer. Diese stellt dem Benutzer folgende Funktionen zur Verfügung: Profileinsicht, Benutzer anlegen (je nach Rolle), Statistiken.

Methoden:

- public void openProfilEvent(sender: Object, args: Eventargs): Öffnen des Benutzerprofils.
- public void openStatisticsEvent(sender: Object, args: Eventargs): Statistiken öffnen.
- public void openNewUserEvent(sender: Object, args: Eventargs): Neuen Mitarbeiter anlegen.

## 2.12 NewUserPanel

Diese Ansicht wird durch die Toolbar instanziert. Hier hat man die Möglichkeit einen neuen Nutzer anzulegen.

- private createUserEvent(sender: Object, args: Eventargs): wird aufgerufen, wenn der User auf den Button „erstellen“ klickt. Es wird eine Information an den Server gesendet.
- private void update(sender : Object, args Eventargs): wie oben
- NewUserPanel(): Konstruktor

## 2.13 ProfilPanel

Das ProfilPanel ermöglicht die Anzeige des Profils.

Attribute:

- private UserVO model: Das zugrundeliegende Model

- private String origin: siehe Glossar

Methoden:

- public Konstruktor ProfilPanel(): Konstruktor
- private void updateUserEvent(sender: Object, args: EventArgs): Wird durch eine Benutzerinteraktion aufgerufen. Die Änderung wird dem Server mitgeteilt.

## 2.14 StatisticsPanel

Der Benutzer verfügt über die Möglichkeit sich Statistiken abhängig von seiner Rolle anzeigen zu lassen. Die Statistiken werden beim Instanziieren durch die Toolbox direkt vom Server geholt.

Attribute:

- private XML stat: Die Statistiken als xml

Methoden:

- public Konstruktor StatisticsPanel(): Konstruktor
- private void getBillEvent(sender: Object, args: EventArgs): beim Klick auf Rechnung anzeigen, wird diese Funktion aufgerufen.

## 2.15 Box {abstract}

Die Basisklasse für alle Panels, die von Box erben und Fotos verwalten.

Attribute:

- private Integer Array photoIDs: ID der Fotos

Methoden:

- public Image getNextPhoto(curPictureID: int): Nächstes Foto.
- public Image getPreviousPhoto(curPictureID: int): Vorheriges Foto.
- private void dragPhotoEvent(sender: Object, args: EventArgs): Wird aufgerufen, wenn der User ein Foto bewegt. Die Veränderungen werden an den Server geschickt und jeder der eine Freigabe zu dieser Leitbox erhalten hat, wird benachrichtigt. Das Bild wird gelocked.

## 2.16 FullscreenView

Vollbildmodus um Foto in maximal möglicher Größe anzeigen zu lassen. Der User kann die aktuelle Liste von Fotos durch navigieren. Die Navigation bedient sich der Box Methoden: getNext -und getPreviousPhotoEvent().

Attribute:

- private Image curPicture: Aktuelles Photo.
- private IGUILIndicator boxPanel: Aktuelle Box.

Methoden:

- public Konstruktor FullscreenView(curPicture: Image, lightboxPanel: IGUILIndicator): Konstruktor

## 2.17 SearchPanel

Die Suche ermöglicht das Suchen von Fotos in der Datenbank, welche an den Server gebunden ist. Durch Eingabe von Tags oder Auswählen einer Kategorien wird die Suche verfeinert. Das SearchPanel instanziiert das SearchResultsPanel, um die Treffer anzeigen zu können. Über die Update Methode erhält diese Klasse alle Treffer der Suchanfrage. (Repräsentiert durch die Photo-IDS). Diese werden dann beim Instanziieren des SearchResultsPanels im Konstruktur mit übergeben.

Attribut:

- private origin: String

Methoden:

- public SearchPanel(): Konstruktor
- private void searchPhotoEvent(sender: Object, args: EventArgs): Startet den Suchvorgang

## 2.18 SearchResultsPanel

Dieses Panel erbt von Box. Die Box Klasse stellt für alle ein Array zur Verfügung (siehe Klasse Box). Die Klassen die von Box erben, können dieses Array nutzen und befüllen. Je nach Anzahl der Treffer instanziiert sich dieses Panel eine Menge von Photopanels, welche sich über die PhotoID die dazugehörige PhotoData vom Server anfragt. Das Anzeigen erfolgt über das ImagePanel, welches ein [BinImage?](#) in kleinerer Auflösung anzeigt.

Attribut:

- private origin: String

Methoden:

- public Konstruktor SearchResultsPanel(photoIDs: Integer[]): Konstruktor bekommt die Photo Ids übergeben und kann pro Foto die detaillierten Informationen (Photodata) vom Server anfragen.

## 2.19 PhotoPanel

Hält für jedes Foto die Photodata und erzeugt sich ein ImagePanel, welches das [BinImage?](#) in kleinerer Auflösung anzeigt.

Attribute:

- private PhotoData photodata: die Informationen zum Foto.
- private String origin: siehe Glossar
- private int photoID: ID des Fotos

- private int LightboxID

Methoden:

- public Konstruktor PhotoPanel(photoID: int, lightboxID: int): Konstruktor
- private void setLightboxPhotoStateEvent(sender : Object, args : Eventargs): Diese Methode wird beim Setzen des Fotostatus genutzt("yes", "no", "undecided") genutzt.
- private void setPhotoRateEvent(sender : Object, args : Eventargs): Wird ausschließlich in der Rolle der Agentur zur Festlegung des Gewinnanteils genutzt.
- private void openMetaDataPanelEvent(sender : Object, args : Eventargs): wenn der User auf den Button Metadaten klickt, wird diese Methode aufgerufen
- private void addEffectEvent(sender : Object, args : Eventargs): wenn der User einen Effekt hinzufügt, wird der Server benachrichtigt.
- private void printPhotoEvent(sender : Object, args : Eventargs): wenn der User auf drucken klickt

## 2.20 MetaDataPanel

Attribute:

- private int lightboxID
- private int photoID
- private Photodata photodata

Methoden:

- public Konstruktor MetaDatenPanel(photodata: Photodata, photoID: int, lightboxID: int): Konstruktor.
- private void addTagLightboxPhotoEvent(sender: Object, args: Eventargs): Beauftragt den Server Tags zu einem Bild hinzuzufügen
- private void renameLightboxPhotoEvent(sender: Object, args: Eventargs): Der User gibt einen neuen Namen für die Lightbox ein und der Server wird benachrichtigt.

## 2.21 ImagePanel

Dieses Panel repräsentiert das tatsächliche Bild und ist rein für dessen Anzeige zuständig, wobei das Bild aus dem [BinImage?](#)-Model bezogen wird.

Attribute:

- private [BinImage?](#) model: Die tatsächliche Bildquelle.

Methoden:

- public Konstruktor Image(model: [BinImage?](#)) Konstruktor

## 2.22 LightboxPanel

Dieses Panel erbt von Box. Eine der Kernklassen , da hier erstens das kollaborative Arbeiten stattfinden kann und dem User viele Funktionen zur Verfügung stehen.

Attribute:

- private LightboxVO model: Das zugrundeliegende LightboxModel
- private String origin: siehe Glossar

Methoden:

- public Konstruktor LightboxPanel(model: LightboxVO): Konstruktor
- private void buyLightboxEvent(sender: Object, args: EventArgs): Wenn der Kunde die Lightbox kaufen möchte, so wird diese Methode angestoßen, um den Server zu benachrichtigen.
- private void openAddressbookEvent(sender: obj, args: EventArgs): Wenn der User die Lightbox für einen Teilnehmer freigeben möchte, wird das AdressbuchPanel geöffnet.
- private void dropImageEvent(sender: Object, args: EventArgs): Ermöglicht die flexible Anordnung der Bilder. Der Server wird benachrichtigt.
- private void renameLightboxEvent(sender: Object, args: EventArgs): User benennt eine Lightbox um. Der Server wird benachrichtigt.
- private void deleteLightboxPhotoEvent(sender: Object, args: EventArgs): User löscht ein Foto in der Lightbox. Der Server wird benachrichtigt.
- public int getLightboxID(): Gibt die ID der Lightbox zurück.
- private void addEffectEvent(sender: Object, args: EventArgs): User fügt Effekt zur gesamten Lightbox hinzu.Der Server wird benachrichtigt.
- private void setAgencyRateEvent(sender: Object, args: EventArgs): Wenn der User ein Agenturmitarbeiter/admin ist kann er einen %- Anteil für die Lightbox bestimmen, welcher den Gewinn für die Agentur darstellt, wenn Fotos verkauft werden. Der Server wird benachrichtigt.
- private void getZipEvent(sender: Object, args: EventArgs): Wenn der Kunde seine gekauften Bilder herunterladen möchte.Der Server wird benachrichtigt.
- private void leaveLightboxEvent(): Schließen einer Lightbox. Der Server wird benachrichtigt.

## 2.23 AddressbookPanel

Das Adressbuch hat nur die Funktion, einen User für eine Lightbox einzuladen, um z.B. mit diesem zu chatten.

Attribute:

- private int lightboxID

- private user: List<UserVO>
- private String origin

Methoden:

- public Konstruktor AddressbookPanel(lightboxID: int): Konstruktor
- private void inviteEmployeeEvent(sender: object, args: Eventargs): Das Hinzufügen von weiteren Teilnehmern wird durch diese Methode angestoßen, wenn der User den „einladen“- Button klickt.

## 2.24 ChatPanel

Die Teilnehmer der geöffneten Lightbox können über den ChatPanel kommunizieren.  
Voraussetzung: die Teilnehmer sind online.

Attribute:

- private String origin
- private int LightboxID
- private user: List<UserVO>

Methoden:

- public Konstruktor ChatPanel(lightboxID: int): Konstruktor
- private void sendLightboxMessageEvent(sender: object, args: Eventargs): wenn ein user eine Nachricht verschicken will, wird diese Methode aufgerufen.

## 2.25 Listitem

Ist Bestandteil der Lightbox-Liste, welche sich im AllLightboxPanel befindet. Das Listitem hält jeweils das LightboxVO. Wenn der User eine Lightbox öffnen will, hat man über die Items Zugriff auf die Objekte.

Attribute:

- private LightboxVO model: Model einer Lightbox.
- private String origin

Methoden:

- public Konstruktor Listitem(lightbox: LightboxVO): Konstruktor

## 2.26 AllLightboxesPanel

Eine Sammlung aller Lightbox auf die ein Benutzer Zugriff hat. Der Konstruktor ruft getAllLightboxList() in der Communicator Klasse auf, um die Liste vom Server über die update-Methode zu erhalten.

Attribute:

- private String origin

Methoden:

- public Konstruktor AllLightboxesPanel(): Konstruktor
- private void deleteLightboxEvent(sender: Object, args: EventArgs): Wird aufgerufen, wenn der User die Lightbox löschen möchte.
- private void enterLightboxEvent(sender: Object, args: EventArgs): Öffnet eine Lightbox aus der Liste. Benachrichtigt den Server, dass die Lightbox betreten wurde.
- private void createLightboxEvent(sender: Object, args: EventArgs): Erstellt eine neue Lightbox. Der Server wird benachrichtigt.
- private void lightboxCopyEvent(sender: Object, args: EventArgs): Die Agentur kann sich eine Lightbox kopieren, wenn die Lightbox gesperrt wurde, weil der Kunde die Lightbox gekauft hat. Der Server wird benachrichtigt.

## 3 package: ValueObjects

Das ValueObjects Package stellt alle ValueObject-Klassen zur Verfügung. Das Package sieht beim Server und beim Client identisch aus. Der Client bearbeitet keine ValueObjects, nur der Server. Der Client muss die ValueObjects jedoch kennen, um die Messages vom Server (welche in der Regel Objecte enthalten) identifizieren zu können. Sie dienen auf Clientseite ausschließlich zum Auslesen von Informationen.

Der Client muss sich die Instanzen je nach MessageType casten, welche vom Server gesendet werden.

### 3.1 PhotoVO

Die PhotoVO Klasse enthält die Photodata, sowie die das Bild als [BinImage?](#).

Attribute:

- private boolean deleteFlag: Um zu signalisieren, ob dass Foto vom Fotografen gelöscht wurde
- privat boolean soldFlag: Gibt an, ob das Foto mindestens einmal verkauft wurde.

### 3.2 PhotoData

Die Klasse PhotoData enthält die Metadaten eines Fotos den Preis des Fotos, und eine kurze Beschreibung.

Attribute:

- private int id: eindeutige Bestimmung des Fotos bei auszuführenden Operationen
- private String title: Name des Fotos
- private String description: Beschreibung zum Foto
- private String location: Ort, an dem das Foto aufgenommen wurde

- private float PhotographerPhotoPrice: Vom Fotografen festgelegter Preis
- private int uploadTimestamp: Datum und Zeit an dem das Foto hochgeladen wurde
- private exif\_Timestamp: Zeitpunkt der Aufnahme
- private exif\_Orientation: Peilung der Aufnahme
- private exif\_Location: Ort an dem das Bild aufgenommen wurde (wird von der Kamera erstellt, falls diese über GPS oder ähnliches verfügt)

### **3.3 BinImage**

Die BinImage-Klasse hat den Zweck das Originalbild in einer kleineren Auflösung bereitzustellen.

Attribute:

- private byte Array image: Beinhaltet das Bild
- private int Array size: Enthält die Größe des BinImages

### **3.4 AgencyPhotoData**

Diese Klasse ist eine spezielle Unterklasse der Photodata Klasse, da die Agentur den Titel des Bildes global setzen kann und zusätzlich einen Prozentsatz bestimmen kann, damit die Agentur an jedem verkauften Bild Einnahmen erzielt.

Attribute:

- private String titel: globaler Name für das Bild
- private float rate: Beinhaltet den prozentualen Anteil des Gewinnes von einem Bild

### **3.5 LightboxPhotoData**

Lightbox spezifische Daten, die direkt an der jeweiligen Lightbox gespeichert werden müssen, da spezielle Eigenschaften nur für die jeweilige Lightbox gelten.

Attribute:

- private String titel: Enthält den Titel eines Bildes, welcher für eine bestimmte Lightbox gilt.
- private Photo\_State state: Enthält den Status, für welchen sich der Kunde entscheidet (kaufen, nicht kaufen, unentschlossen).
- private int x: Enthält die x Position eines Bildes innerhalb einer Lightbox.
- private int y: Enthält die y Position eines Bildes innerhalb einer Lightbox.
- private Photo\_Effect effect: Enthält die Informationen über die Effekte, welche auf ein Foto angewandt wurden wie z.B. einen schwarz - weiß Filter.

## 3.6 LightboxVO

Attribute:

- private int id
- private String titel: Enthält den Titel einer Lightbox.
- private String description: Enthält eine kurze Beschreibung einer Lightbox.
- private Lightbox\_State state: Enthält Informationen über den Status einer Lightbox, ob sie privat, für andere Kunden freigegeben oder geschlossen ist.

## 3.7 ChatMessageVO

Enthält die Nachrichten, die von Benutzer gesendet. Wenn der User eine ChatMessageVO erhält, kann er diese lesen.

Attribute:

- private String message: Enthält die Nachricht eines Benutzers.
- private int userId: Enthält die ID des Benutzers, welcher die Nachricht verfasst hat.
- private int timestamp: Enthält das Datum und die Uhrzeit zu welcher eine Nachricht verfasst wurde.

## 3.8 CategoryVO

Enthält die Kategorien in die ein Bild eingeordnet werden kann.

- private String titel: Enthält den Namen einer Kategorie.
- private int Id: Enthält die ID einer Kategorie über die sie eindeutig bestimmt werden kann.

## 3.9 TagVO

Tag für ein Bild.

- private String tag: Enthält den Inhalt aller Tags
- private int userID: Enthält die ID des Benutzers, welcher den Tag erstellt hat.
- private Role role: Enthält die Rolle des Benutzers.

## 3.10 UserVO

Das Model für den Benutzer und dessen Daten.

Attribute:

- private int id: Enthält die ID eines Benutzers, über die er eindeutig identifiziert werden kann.
- private String username: Enthält den Benutzernamen eines Benutzers.

- private String password: Enthält das Passwort eines Benutzers.
- private String email: Enthält die E-Mail eines Benutzers.
- private String firstName: Enthält den Vornamen eines Benutzers.
- private String lastName: Enthält den Nachnamen eines Benutzers.
- private int age: Enthält das Alter eines Benutzers.
- private User\_Sex sex: Enthält das Geschlecht eines Benutzers.
- private String website: Enthält die Website eines Benutzers.

## 3.11 PhotographerVO

Die Daten des Fotografen die zusätzlich zum UserVO, zu welchem eine Vererbungshierarchie besteht, gespeichert werden.

Attribute:

- private float photoPrice: Enthält einen globalen Einheitspreis für alle Fotos eines Fotografen.

## 3.12 EmployeeVO

Die Daten des Employees die zusätzlich zum UserVO, zu welchem eine Vererbungshierarchie besteht, gespeichert werden.

Attribute:

- private Employee\_Role role: Enthält die Rolle eines Mitarbeiters.
- private boolean admin: Enthält die Information, ob der Mitarbeiter die Rolle eines Admins hat oder nicht.

## 3.13 AgencyVO

Speichert die Agenturdaten.

Attribute:

- private float agencyRate: Enthält die Globale Gebühr die eine Agentur für verkauftte Fotos aufschlägt.
- private String name: Enthält den Namen einer Agentur.
- private int id: Enthält die ID einer Agentur über welche diese zu bestimmen ist.

## 3.14 CustomerVO

Diese Klasse bietet Zugriff auf die Daten des Kunden, welcher von der Agentur angelegt wird.

- private String name: Enthält den Namen der Organisation eines Kunden falls er eine solche besitzt.

## 3.15 Role (Enum)

Bietet einen Aufzählungstyp zur Identifizierung der unterschiedlichen Benutzerrollen.

## 3.16 MessageType

Bietet einen Aufzählungstyp zur Identifizierung der unterschiedlichen Messages, welche die Kommunikation und Initialisierung von Anfragen an den Server über unterschiedliche MessageTypes ermöglicht.

# 4 package: Controller

## 4.1 MainController

Diese Klasse ist nach dem Singleton Pattern definiert. Sie ist für jeden zugänglich und hält als einzige die SessionID und den User. Des Weiteren stellt sie die Kommunikationsschnittstelle zwischen View und Connection dar.

Attribute:

- private UserView view: Ist abhängig vom angemeldeten Benutzer. z.B. User ist Photographer => PhotographerView wird instanziert.
- private UserVO model: Das zurgrundeliegende Benutzermodel.
- private MainController instance
- private ICommunicator Communicator: Die Hauptkommunikationseinheit.
- private String sessionID: Die Session ID des Benutzer.

Methoden:

- private Konstruktor MainController(): Konstruktor.
- public MainController getInstance(): Returned die MainController. (Singleton)
- public UserVO getUser(): Gibt den Benutzer als UserVO zurück.
- public ICommunicator getCommunicator(): Liefert die Hauptkommunikationsschnittstelle zurück.
- public void bootUP(user: UserVO, sessionID: String): Wird aufgerufen, wenn der LoginView oder der RegisterView sein UserVO vom Server bekommen hat.
- public String getSessionID(): Returned die SessionID.

# 5 package: Connection

## 5.1 ICommunicator <<Interface>>

Dieses Interface muss die Communicator Klasse implementieren. Es wird somit

gewährleistet, dass die View über die Methoden eine Anfrage senden kann. Die einzelnen Methoden sind identisch zu den Request auf dem Server. (siehe Beschreibung Backend und Schnittstellendefinition)

## 5.2 Communicator

In dieser Klasse wird die Message erzeugt und über die sendMessage- Methode an den Publisher weitergeleitet. Der Publisher wird in dieser Klasse erzeugt.

Attribute:

- private Connection aMQconnection: activeMQ-Verbindung
- private MessageProducer aMQproducer: Message Producer um Nachrichten zu erzeugen
- private String brokerURL: Die Verbindungsurl.
- private aMQsession: Die aktuelle Session mit der AMQ.

Verweis auf: Kommunikation Frontend-Backend Menüpunkt

Methoden:

- private void sendMessage(message: Message): Leitet die Message an den Publisher weiter.

## 5.3 Publisher

Der Publisher dient zur Kommunikation nach außen und verschickt die Anfrage in Form von Messages an den Server.

Methoden:

- public void sendMessage(message: Message): Versendet die Nachrichten.

## 5.4 Consumer

Der Consumer bildet das Herzstück für eingehende Verbindungen und verteilt einkommende Nachrichten. Die Messages werden den Views zugeordnet.

Attribute:

- private Map<origin IGUILIndicator> participants: Eine Liste mit angemeldeten IGUILIndicatoren, welche auf eine Antwort vom Server warten.
- private MessageConsumer messageConsumer:

Methoden

- private IGUILIndicator findParticipant(origin: String): Findet den wartenden Teilnehmer identifiziert über den Origin.

## 5.5 IParticipant <<Interface>>

Dieses Interface müssen Communicator und Consumer implementieren. Die View meldet sich bei der Communicator Klasse an. Diese reicht die Anmeldung an den Consumer weiter, da die View den Consumer nicht kennt.

Methoden:

- public void addParticipant(view: IGUILIndicator, orig in: String): Fügt der Liste Teilnehmer hinzu.
- public void removeParticipant(origin: String): Entfernt Kommunikationsteilnehmer aus einer Liste von Teilnehmern.

# 6 Glossar

### Origin

Dient der eindeutigen Zuordnung, wenn der Client etwas vom Server anfragt und dieser antwortet, kann der Origin dem Aufrufer wieder eindeutig zugeordnet werden.

### Topic

Viele Sender können zu einem Thema (Topic) Nachrichten einstellen (publish). Genauso kann es viele Interessenten (Subscriber) geben, die an diesen Nachrichten interessiert sind.

### Message

Eine Message Queue wird oftmals für eine Point-to-Point-Kommunikation genutzt: Die Sender stellen Nachrichten in die Nachrichten-Warteschlange. Der Empfänger holt diese Nachricht ab, verarbeitet sie und löscht sie aus der Warteschlange.

### JSON

Die Abkürzung JSON steht für JavaScript Object Notation. Mit JSON kann man Informationen in einer Programmiersprache, die eine JSON Bibliothek anbietet serialisieren und in einer anderen Programmiersprache wieder deserialisieren.

### Message-Types

Eine eindeutige Definition und Festlegung von Typen, damit Server und Client Messages identifizieren können. (Auflistung: siehe Architektur)

### Session-ID

Dient der eindeutigen Identifizierung eines eingeloggten Benutzers.



# Kommunikation Client Server

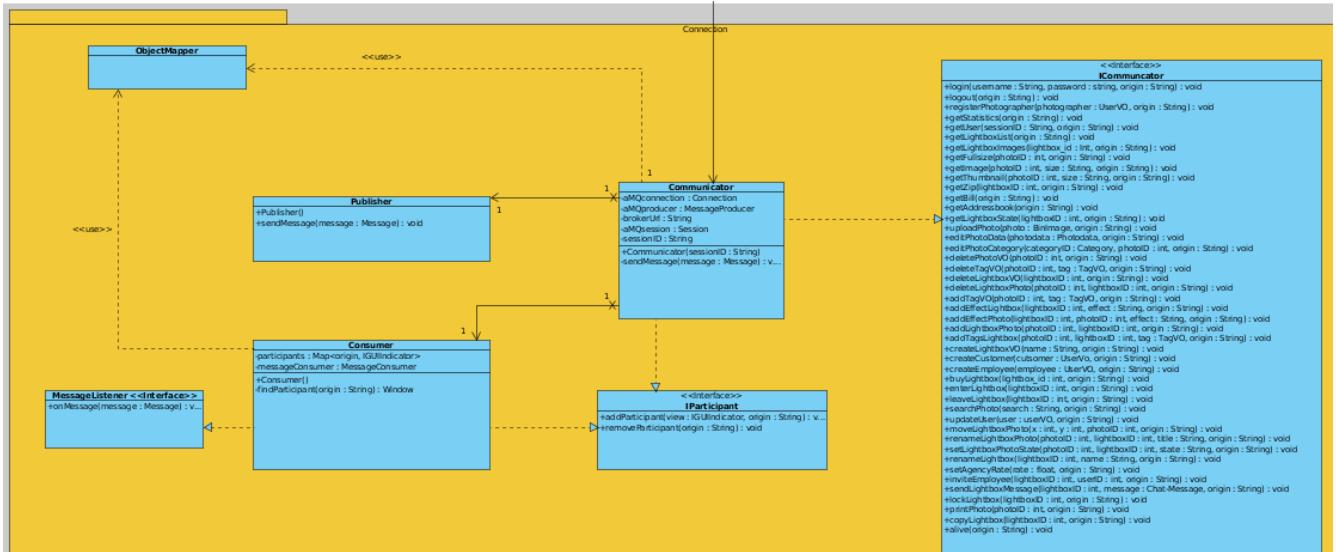
Anhang zum Leuchttisch Designdokument

Dieser Angang beschreibt die Kommunikation zwischen Client und Server via ActiveMQ.

## Inhaltsverzeichnis

1 Client.....	2
1.1 Communicator.....	2
1.2 Publisher.....	2
1.3 Consumer.....	3
2 Server.....	4
2.1 Consumer.....	4
2.2 Publisher.....	4
2.3 MessageJob.....	4
2.4 Response Job.....	5

# 1 Client



In der ActiveMQ Communication wird die Kommunikation mit dem Messaging Service ermöglicht. Das Package ist zuständig für das Senden und Empfangen von Nachrichten.

## **1.1 Communicator**

Der Communicator dient als zentrale Anlaufstelle für die Kommunikation. Er wird als Instanz beim MainController (Singleton) gehalten und implementiert das Interface ICommunicator, welches die Schnittstellen-Methoden definiert, über die Anfragen an den Server gestellt werden können. Der Communicator hält sich jeweils eine Publisher- und Consumer-Instanz. Außerdem implementiert er das Interface IParticipant, welches sicherstellt, dass Methoden zum hinzufügen und entfernen von Participants vorhanden sind. Aufrufe dieser Methoden werden an den Consumer weitergeleitet, welcher die Participants verwaltet.

## **1.2 Publisher**

Wenn durch eine Aktion bei der Benutzeroberfläche Informationen vom Server benötigt werden, oder dort Änderungen vorgenommen werden müssen (Persistenz), fragt der aufrufende IGUILIndicator beim MainController den Communicator an und meldet sich bei ihm mit seinem originator-String als Participant an, damit die eingehende respondeMessage des Servers diesem wieder zugeordnet werden kann. Danach ruft er die entsprechende Schnittstellen-Methode beim Communicator auf.

An die aufgerufene Methode werden immer die benötigten Parameter übergeben, die gebraucht werden, um die entsprechende Message für den Server zu erzeugen. Innerhalb der Schnittstellen-Methoden findet der komplette Aufbau der aMQ-Message für den Server statt. Danach wird mit der privaten sendMessage-Methode die zu sendende Message an den Publisher weitergereicht, welcher sie abschickt.

In dem Moment, wo eine Anfrage an den Communicator anhand einer der gegebenen

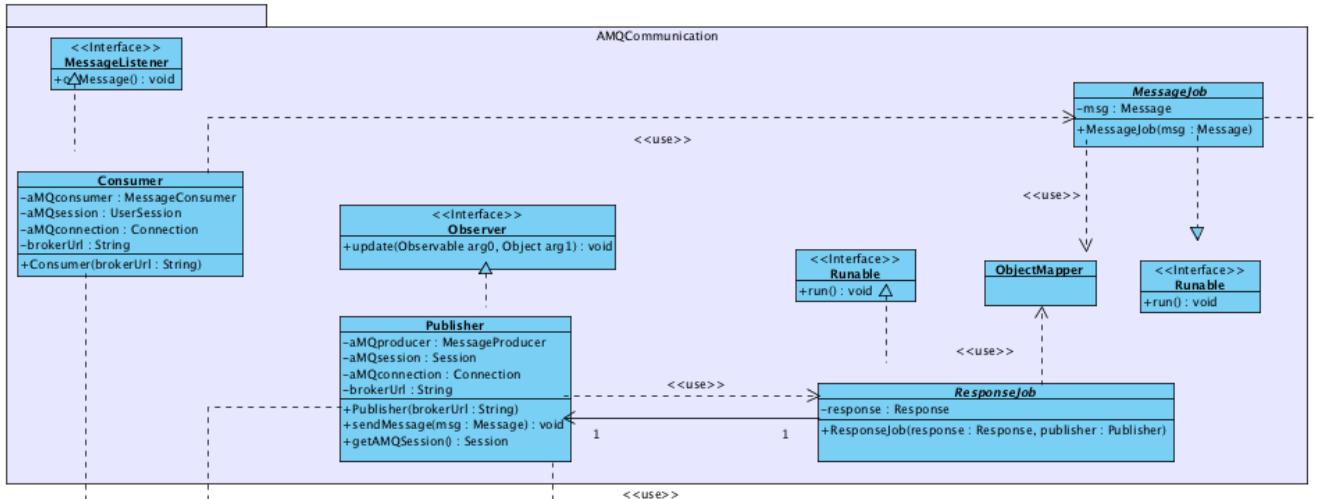
Methoden gestellt wird, wird also automatisch eine Message erstellt und abgeschickt.

## 1.3 Consumer

Der Consumer des Clients nimmt Messages des Servers an. Durch den in der Message enthaltenen originator kann bestimmt werden, von wem die Message ursprünglich aufgerufen wurde und damit ist klar, wer die Daten der Message wieder zurückbekommt. Kommt eine Message beim Consumer an, holt er sich den originator und das mit der Message mitgesendete Objekt. Das Objekt wird mit dem ObjectMapper von JSON deserialisiert. Daraufhin sucht sich der Consumer anhand des Originators den passenden Participant und schickt ihm das Objekt. Der jeweilige Participant kann dann anhand des messageTyps feststellen, wie er das Objekt handzuhaben hat.

Der Consumer implementiert das Interface IParticipant, welches sicherstellt, dass Methoden zum hinzufügen und entfernen von Participants vorhanden sind.

# 2 Server



In der ActiveMQ Communication wird die Kommunikation mit dem Messaging Service ermöglicht. Das Package ist zuständig für das Senden und Empfangen von Nachrichten.

## 2.1 Consumer

Der Consumer implementiert das MessageListener Interface und reagiert auf eingehende Nachrichten von ActiveMQ. Er besitzt zur Kommunikation mit dem ActiveMQ Broker einen MessageConsumer, eine Usersession und eine Connection. Mit diesen verbindet er sich über eine BrokerURL an einem Topic. Der Consumer benutzt die Instanz des Servers um auf den ThreadPool zugreifen zu können und die MessageJob Klasse.

## 2.2 Publisher

Der Publisher implementiert das Observer Interface und wartet auf Responses, welche er als ResponseJob verarbeiten kann. Er benutzt die "getAMQSession" Methode um Nachrichten erstellen und die "send" Methode um Nachrichten versenden zu können.

## 2.3 MessageJob

Implementiert das Runnable Interface und benutzt ObjectMapper um eingehende Daten per JSON deserialisieren zu können. Der Message Job wird im ThreadPool verarbeitet und baut ein auf die Anfrage individuell abgestimmten Request: Bei einer eingehenden Message wird ein MessageJob erzeugt, dem die Message übergeben wird. Der MessageJob stellt anhand des messageType fest, um was für eine Anfrage es sich handelt und holt sich die entsprechenden zu erwartenden Daten aus der Message und wenn nötig, werden diese noch per JSON deserialisiert. Mit den gewonnenen Daten wird daraufhin ein Request entsprechend der messageID erzeugt und dieser wieder dem ThreadPool zur Abarbeitung übergeben.

## 2.4 Response Job

Der Response Job implementiert das Runnable Interface und kennt den Publisher. Er benutzt den Object Mapper um die Daten für den Versand zu serialisieren. Der Response Job wird im ThreadPool verarbeitet und erzeugt seine respondMessage entsprechend des messageType. Ein ResponsJob wird vom Publisher erzeugt und bekommt einen Response mit übergeben. Anhand des messageTypes holt er sich vom Response die für die responseMessage benötigten Daten und serialisiert sie gegebenenfalls mit JSON. Anschließend übergibt er die erzeugte responseMessage an den Publisher, der sie dann verschickt.



# Schnittstellen

Anhang zum Leuchttisch Designdokument

Dieser Angang definiert die Methoden, die der Server zur Kommunikation mit dem Client zur Verfügung stellt.

## Inhaltsverzeichnis

1 Datentypen.....	4
1.1 Einfache Datentypen.....	4
1.1.1 Float.....	4
1.1.2 DateTime.....	4
1.1.3 binary.....	4
1.1.4 String.....	4
1.1.5 Boolean.....	4
1.1.6 Integer.....	4
1.1.7 Double.....	4
1.2 Agency.....	4
1.3 AgencyPhotoData.....	5
1.4 Bill.....	5
1.5 Category.....	5
1.6 Chat-Message.....	5
1.7 Customer.....	6
1.8 Employee.....	6
1.9 ErrorMessage.....	6
1.10 Image.....	7
1.11 Lightbox.....	7
1.12 LightboxPhotoData.....	7
1.13 Photo.....	8
1.14 PhotoData.....	8
1.15 Photographer.....	9
1.16 PurchasedPhoto.....	9
1.17 Response.....	9
1.18 SucceedMessage.....	10
1.19 Tag.....	10
1.20 User.....	10
1.21 User-Info.....	11
2 API.....	12
2.1 Agency.....	12
2.1.1 Agency.create().....	12
2.1.2 Agency.setPhotoData().....	12
2.1.3 Agency.setRate().....	12
2.2 Customer.....	13
2.2.1 Customer.create().....	13
2.2.2 Customer.getBoughtPhotos().....	13
2.3 Employee.....	13
2.3.1 Employee.create().....	14
2.3.2 Employee.update().....	14
2.4 Lightbox.....	14
2.4.1 Lightbox.addEditor().....	14
2.4.2 Lightbox.addEffect().....	15

2.4.3 Lightbox.addPhoto().....	15
2.4.4 Lightbox.addPhotoEffect().....	15
2.4.5 Lightbox.addPhotoTag().....	16
2.4.6 Lightbox.buy().....	16
2.4.7 Lightbox.copy().....	17
2.4.8 Lightbox.create().....	17
2.4.9 Lightbox.delete().....	17
2.4.10 Lightbox.deletePhoto().....	18
2.4.11 Lightbox.enter().....	18
2.4.12 Lightbox.getContactList().....	18
2.4.13 Lightbox.getImages().....	19
2.4.14 Lightbox.getInvitedEmployees().....	19
2.4.15 Lightbox.getLightBox().....	19
2.4.16 Lightbox.getList().....	20
2.4.17 Lightbox.getPhotoData().....	20
2.4.18 Lightbox.getPhotoEffects().....	20
2.4.19 Lightbox.getZip().....	21
2.4.20 Lightbox.leave().....	21
2.4.21 Lightbox.lockPhoto().....	21
2.4.22 Lightbox.movePhoto().....	22
2.4.23 Lightbox.print().....	22
2.4.24 Lightbox.rename().....	23
2.4.25 Lightbox.renamePhoto().....	23
2.4.26 Lightbox.sendLightBoxMessage().....	23
2.4.27 Lightbox.setPhotoRating().....	24
2.4.28 Lightbox.unlockPhoto().....	24
2.5 Photo.....	24
2.5.1 Photo.addTag().....	25
2.5.2 Photo.delete().....	25
2.5.3 Photo.deleteTag().....	25
2.5.4 Photo.editCategory().....	26
2.5.5 Photo.editData().....	26
2.5.6 Photo.getAgencyData().....	26
2.5.7 Photo.getCategories().....	27
2.5.8 Photo.getFullsize().....	27
2.5.9 Photo.getImagesInCategory().....	27
2.5.10 Photo.getPhoto().....	28
2.5.11 Photo.getPhotoData().....	28
2.5.12 Photo.getTags().....	28
2.5.13 Photo.getThumbnail().....	29
2.5.14 Photo.search().....	29
2.5.15 Photo.upload().....	29
2.6 Photographer.....	30
2.6.1 Photographer.register().....	30
2.6.2 Photographer.update().....	30
2.7 User.....	30
2.7.1 User.get().....	31
2.7.2 User.getBills().....	31

2.7.3 User.getUsername().....	31
2.7.4 User.login().....	31
2.7.5 User.logout().....	32
2.7.6 User.update().....	32

# 1 Datentypen

Diese Datentypen werden von den Methoden der API zum Datenaustausch verwendet.

## 1.1 Einfache Datentypen

### 1.1.1 **Float**

Repräsentiert ein Float

### 1.1.2 **DateTime**

Repräsentiert ein Datum mit Uhrzeit. Das Format ist YYYY-MM-DDTHH:MM:SS+ZZZZ, z.B. 2011-03-28T17:58:23+0001

### 1.1.3 **binary**

Repräsentiert binäre Daten, z.B. Bilddaten eines Fotos

### 1.1.4 **String**

Repräsentiert ein String

### 1.1.5 **Boolean**

Repräsentiert ein Boolean

### 1.1.6 **Integer**

Repräsentiert ein Integer

### 1.1.7 **Double**

Repräsentiert ein Double

## 1.2 Agency

(Object) Repräsentiert eine Agentur

### Attribute

Name	Typ	Beschreibung	Beispiel
created	<a href="#">DateTime</a>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
id	<a href="#">Integer</a>	Die ID der Agentur	33454
modified	<a href="#">DateTime</a>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//

name	<u>String</u>	Der Name der Agentur	<i>The New Super Agency</i>
------	---------------	----------------------	-----------------------------

---

## 1.3 AgencyPhotoData

(Object) Repräsentiert Informationen eines agenturspezifischen Fotos

### Attribute

Name	Typ	Beschreibung	Beispiel
id	<u>Integer</u>	Id des Fotos	23334
rate	<u>Float</u>	Gebühr der Agentur	True   False
titel	<u>String</u>	Agenturspezifische Titel des Bildes	<i>Neue Abendsonne</i>

---

## 1.4 Bill

(Object) Repräsentiert eine Rechnung

### Attribute

Name	Typ	Beschreibung	Beispiel
buyer	<u>Employee</u>	Der Käufer des Bildes	-
date	<u>DateTime</u>	Das Datum der Rechnung	2010-03-29T14:05:34+0000//
id	<u>Integer</u>	ID der Rechnung	8877
photos	<u>PurchasedPhoto</u> [ ]	Eine Liste mit gekauften Fotos	-

---

## 1.5 Category

(Object) Repräsentiert eine Kategorie

### Attribute

Name	Typ	Beschreibung	Beispiel
id	<u>Integer</u>	ID der Kategorie	55443
parentID	<u>Integer</u>	Die übergeordnete Kategorie	<i>Landschaft</i>
titel	<u>String</u>	Name der Kategorie	<i>Berglandschaft</i>

---

## 1.6 Chat-Message

(Object) Repräsentiert eine Chat Nachricht

### Attribute

Name	Typ	Beschreibung	Beispiel
date	<u>DateTime</u>	Datum	2010-03-29T14:05:34+0000//

---

<b>id</b>	<a href="#">Integer</a>	Die ID der Message 8877
<b>message</b>	<a href="#">String</a>	Die Chat Nachricht Ja, dieses Foto finde ich auch sehr gut.
<b>userId</b>	<a href="#">Integer</a>	userID 9876

---

## 1.7 Customer

(Object) Repräsentiert einen Kunden

### Attribute

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
created	<a href="#">DateTime</a>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
<b>id</b>	<a href="#">Integer</a>	Die ID des Kunden	33454
modified	<a href="#">DateTime</a>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//
name	<a href="#">String</a>	Der Name des Kunden	<i>Die Websitenbauer</i>

---

## 1.8 Employee

(Object) Repräsentiert ein Mitarbeiter Profil

### Attribute

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
admin	<a href="#">Boolean</a>	Gibt an ob der Mitarbeiter ein Admin ist	<i>True False</i>
age	<a href="#">Integer</a>	Das Alter des Mitarbeiters	45
created	<a href="#">DateTime</a>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
email	<a href="#">String</a>	Die E-Mail Adresse des Mitarbeiters	klas.s@...
firstname	<a href="#">String</a>	Der Vorname des Mitarbeiters	Klaus
<b>id</b>	<a href="#">Integer</a>	userID des Mitarbeiters	34443
lastname	<a href="#">String</a>	Der Nachname des Mitarbeiters	Schmidt
modified	<a href="#">DateTime</a>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//
role	<a href="#">String</a>	Beschreibt die Rolle des Mitarbeiters	<i>Customer Agency</i>
sex	<a href="#">String</a>	Das Geschlecht des Mitarbeiters	<i>M F</i>
username	<a href="#">String</a>	Benutzername des Mitarbeiters	<i>k_schmidt</i>
website	<a href="#">String</a>	Die Website des Mitarbeiters	<i>www.k-schmidt.de</i>

---

## 1.9 ErrorMessage

(Object) Repräsentiert eine Message mit einer Fehlermeldung

### Attribute

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
data	<a href="#">String</a>	Beliebige zusätzliche Daten für die	<i>Disk quota exceeded</i>

	Fehlermeldung	
id	<u>Integer</u>	Die ID der Fehlermeldung
string	<u>String</u>	Fehlermeldung <i>Es trat ein Fehler bei der Bildübertragung auf.</i>

---

## 1.10 Image

(Object) Repräsentiert das Bild in Bilddaten

### Attribute

Name	Typ	Beschreibung	Beispiel
data	<u>binary</u>	Daten des Fotos	-

---

## 1.11 Lightbox

(Object) Repräsentiert eine Lightbox

### Attribute

Name	Typ	Beschreibung	Beispiel
created	<u>DateTime</u>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
description	<u>String</u>	Beschreibung der Lightbox	Das ist die Beschreibung der Lightbox.
id	<u>Integer</u>	ID der Lightbox	28388
modified	<u>DateTime</u>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//
name	<u>String</u>	Name der Lightbox	Projekt2011
state	<u>String</u>	Aktueller Zustand der Lightbox	private   shared   closed

---

## 1.12 LightboxPhotoData

(Object) Repräsentiert die Lightboxspezifischen Attribute eines Photos

### Attribute

Name	Typ	Beschreibung	Beispiel
created	<u>DateTime</u>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
lightboxID	<u>Integer</u>	Id der Lightbox	23334
modified	<u>DateTime</u>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//
order	<u>Integer</u>	Gibt an, an welcher Stelle sich das Foto befindet	-1 0 1
photoID	<u>Integer</u>	Id des Fotos	23334
rating	<u>String</u>	Gibt an, wie ein Kunde das Foto bewertet	-1 0 1

<b>titel</b>	<u>String</u>	hat Lightboxspezifischer Titel des Bildes	<i>Neue Abendsonne</i>
--------------	---------------	--	------------------------

---

## 1.13 Photo

(Object) Repräsentiert ein Photo mit Daten und Bilddaten

### Attribute

Name	Typ	Beschreibung	Beispiel
created	<u>DateTime</u>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
deleteFlag	<u>Boolean</u>	Gibt an, ob das Foto vom Fotografen gelöscht wurde	True   False
description	<u>String</u>	Beschreibung des Fotos	23334
exif_Location	<u>String</u>	exif_Location	50.374646,8.3836264
exif_Orientation	<u>String</u>	exif_Orientation	Hochformat   Querformat
exif_Timestamp	<u>String</u>	exif_Timestamp	27.01.2009 09:08:16
height	<u>Integer</u>	Höhe des Fotos	500
id	<u>Integer</u>	Id des Fotos	23334
image	<u>Image</u>	Die eigentlichen Bilddaten	-
modified	<u>DateTime</u>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//
name	<u>String</u>	Name des Fotos	Foto001_super
price	<u>Float</u>	Preis des Fotos	4.90
soldFlag	<u>Boolean</u>	Gibt an, ob das Foto mindestens einmal verkauft wurde.	True   False
width	<u>Integer</u>	Breite des Fotos	300

---

## 1.14 PhotoData

(Object) Repräsentiert die Daten eines Fotos

### Attribute

Name	Typ	Beschreibung	Beispiel
created	<u>DateTime</u>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
deleteFlag	<u>Boolean</u>	Gibt an, ob das Foto vom Fotografen gelöscht wurde	True   False
description	<u>String</u>	Beschreibung des Fotos	23334
exif_Location	<u>String</u>	exif_Location	??
exif_Orientation	<u>String</u>	exif_Orientation	??
exif_Timestamp	<u>String</u>	exif_Timestamp	??
height	<u>Integer</u>	Höhe des Fotos	500
id	<u>Integer</u>	Id des Fotos	23334
modified	<u>DateTime</u>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//

name	<u>String</u>	Name des Fotos	<i>Foto001_super</i>
price	<u>Float</u>	Preis des Fotos	<i>4.90</i>
soldFlag	<u>Boolean</u>	Gibt an, ob das Foto mindestens einmal verkauft wurde.	<i>True   False</i>
width	<u>Integer</u>	Breite des Fotos	<i>300</i>

---

## 1.15 Photographer

(Object) Repräsentiert ein Fotografen Profil

### Attribute

Name	Typ	Beschreibung	Beispiel
age	<u>Integer</u>	Das Alter des Fotografen	<i>45</i>
created	<u>DateTime</u>	Das Erstellungsdatum	<i>2010-03-29T14:05:34+0000//</i>
email	<u>String</u>	E-Mail Adresse des Fotografen	<i>klas.s@...</i>
firstname	<u>String</u>	Der Vorname des Fotografen	<i>Klaus</i>
id	<u>Integer</u>	userID des Fotografen	<i>34443</i>
lastname	<u>String</u>	Der Name des Fotografen	<i>Schmidt</i>
modified	<u>DateTime</u>	Das Datum der letzten Änderung	<i>2011-10-04T19:24:24+0000//</i>
photoPrice	<u>Float</u>	Globaler Preis für die Fotos des Fotografen	<i>2.95</i>
sex	<u>String</u>	Das Geschlecht des Fotografen	<i>M F</i>
username	<u>String</u>	Nutzername des Ftoografen	<i>k_schmidt</i>
website	<u>String</u>	Die Webiste des Fotografen	<i>www.k-schmidt.de</i>

---

## 1.16 PurchasedPhoto

(Object) Repräsentiert ein verkauftes Foto

### Attribute

Name	Typ	Beschreibung	Beispiel
id	<u>Integer</u>	Die ID der Message	<i>8877</i>
price	<u>Float</u>	Der globale Preis des Fotografen für dieses Bild, oder der fotospezifisch festgelegte Preis	<i>3.90</i>
rate	<u>Float</u>	Die globale Gebühr der Agentur, oder die agenturspezifische Gebühr für das Foto	<i>1.00</i>

---

## 1.17 Response

(Object) Repräsentiert eine Antwort

### Attribute

Name	Typ	Beschreibung	Beispiel
------	-----	--------------	----------

error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Enthält das Ergebnis der Anfrage	-

---

## 1.18 SucceedMessage

(Object) Repräsentiert eine Message die einen Erfolg wiedergibt

### Attribute

Name	Typ	Beschreibung	Beispiel
id	<a href="#">Integer</a>	Die ID der Erfolgsmeldung	1

---

## 1.19 Tag

(Object) Repräsentiert ein Tag

### Attribute

Name	Typ	Beschreibung	Beispiel
id	<a href="#">Integer</a>	Die ID des Tags	23344
name	<a href="#">String</a>	Der Tag-String	Garten
role	<a href="#">String</a>	Die Rollenzugehörigkeit des Tags	CUSTOMER   AGENCY   PHOTOGRAPHER
userID	<a href="#">Integer</a>	userID des Besitzers	234544

---

## 1.20 User

(Object) Repräsentiert ein User-Objekt

### Attribute

Name	Typ	Beschreibung	Beispiel
age	<a href="#">Integer</a>	Das Alter des Users	45
created	<a href="#">DateTime</a>	Das Erstellungsdatum	2010-03-29T14:05:34+0000//
email	<a href="#">String</a>	Die E-Mail Adresse des Users	klas.s@...
firstname	<a href="#">String</a>	Der Vorname des Users	Klaus
id	<a href="#">Integer</a>	userID des Users	34443
lastname	<a href="#">String</a>	Der Nachname des Users	Schmidt
modified	<a href="#">DateTime</a>	Das Datum der letzten Änderung	2011-10-04T19:24:24+0000//
sex	<a href="#">String</a>	Das Geschlecht des Users	M F
username	<a href="#">String</a>	Der Nutzernname des Users	k_schmidt
website	<a href="#">String</a>	Die Website des Users	www.k-schmidt.de

## 1.21 User-Info

(Object) Repräsentiert eine einfaches User-Objekt, welches anderen Nutzern zu Verfügung steht

### Attribute

Name	Typ	Beschreibung	Beispiel
email	<u>String</u>	Die E-Mail Adresse des Users	<i>klas.s@...</i>
firstname	<u>String</u>	Der Vorname des Users	<i>Klaus</i>
id	<u>Integer</u>	userID des Users	<i>34443</i>
lastname	<u>String</u>	Der Nachname des Users	<i>Schmidt</i>
sex	<u>String</u>	Das Geschlecht des Users	<i>M F</i>
username	<u>String</u>	Der Nutzername des Users	<i>k_schmidt</i>
website	<u>String</u>	Die Website des Users	<i>www.k-schmidt.de</i>

---

## 2 API

Nachfolgend findet sich die Liste der Methoden.

### 2.1 Agency

Enthält Methoden die für Agency benötigt werden.

#### 2.1.1 Agency.create()

Eine Agentur wird von der LeuchttischAG angelegt

##### Request

Name	Typ	Beschreibung	Beispiel
name	<a href="#">String</a>	Der Name der Agentur	-

##### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null., wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Erstellen einer Agentur erfolgreich war.	-

#### 2.1.2 Agency.setPhotoData()

Eine Agentur kann Gebühren für ein Foto festlegen

##### Request

Name	Typ	Beschreibung	Beispiel
data	<a href="#">AgencyPhotoData</a>	Die agenturspezifische n Gebühren für ein Foto	54322
session	<a href="#">String</a>	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

##### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Setzen einer fotospezifischen Gebühr einer Agentur erfolgreich war.	-

#### 2.1.3 Agency.setRate()

Eine Agentur kann globale Gebühren festlegen

## Request

Name	Typ	Beschreibung	Beispiel
rate	Float	Die globalen Gebühren für Fotos	2.90
session	String	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

## Response

Name	Typ	Beschreibung	Beispiel
error	ErrorMessage	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	Boolean	Gibt an, ob es sich um einen Fehler handelt	False
result	SucceedMessage	Liefert Aussage, ob das Setzen der Agentur Rate erfolgreich war.	-

## 2.2 Customer

Enthält Methoden die für einen Kunden benötigt werden.

### 2.2.1 Customer.create()

Eine Agentur-Administrator kann einen Kunden anlegen

## Request

Name	Typ	Beschreibung	Beispiel
customer	Customer	Der neu erstellte Kunde	-
session	String	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

## Response

Name	Typ	Beschreibung	Beispiel
error	ErrorMessage	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	Boolean	Gibt an, ob es sich um einen Fehler handelt	False
result	SucceedMessage	Liefert Aussage, ob das Erstellen eines Kunden erfolgreich war.	-

### 2.2.2 Customer.getBoughtPhotos()

Ein Mitarbeiter eines Kunden kann die gekauften Fotos abrufen

## Request

Name	Typ	Beschreibung	Beispiel
session	String	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

## Response

Name	Typ	Beschreibung	Beispiel
error	ErrorMessage	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	Boolean	Gibt an, ob es sich um einen Fehler handelt	False

result    [Integer\[\]](#)    Eine Liste mit IDs der gekauften Photos    -

## 2.3 Employee

Enthält Methoden die für alle Mitarbeiter benötigt werden.

### 2.3.1 Employee.create()

Ein Mitarbeiter kann jeweils entweder vom Agentur-Administrator oder vom Kunden-Administrator erstellt werden.

#### Request

Name	Typ	Beschreibung	Beispiel
session	<a href="#">String</a>	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
user	<a href="#">Employee</a>	Der neu erstellte Mitarbeiter	-

#### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Erstellen eines Mitarbeiters erfolgreich war.	-

### 2.3.2 Employee.update()

Jeder Mitarbeiter kann seine Profildaten ändern.

#### Request

Name	Typ	Beschreibung	Beispiel
Employee	<a href="#">Employee</a>	Das Profil des Mitarbeiters	-
session	<a href="#">String</a>	Die Session-ID des Mitarbeiters	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

#### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Ändern erfolgreich war.	-

## 2.4 Lightbox

Enthält Methoden die für Lightboxen benötigt werden.

## 2.4.1 Lightbox.addEditor()

Fügt der Lightbox einen Mitarbeiter hinzu

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
userID	<a href="#">Integer</a>	Die Session-ID des Mitarbeiters, der hinzugefügt wird	asiajJFDISmdu34858bdjdcb4r5bddcb

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Einladen erfolgreich war.	-

## 2.4.2 Lightbox.addEffect()

Auf eine Lightbox (also auf alle in ihr enthaltenen Bilder) lassen sich bestimmte Effekte anwenden, z.B. sepia, schwarz/weiß.

### Request

Name	Typ	Beschreibung	Beispiel
effect	<a href="#">String</a>	Foto-Effekt für Bilder	-
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Hinzufügen eines Effekts erfolgreich war.	-

## 2.4.3 Lightbox.addPhoto()

Es können Bilder einer Lightbox hinzugefügt werden.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	32233
photoID	<a href="#">Integer</a>	Die ID des Photos	33323
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

## Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Hinzufügen erfolgreich war.	-

## 2.4.4 Lightbox.addPhotoEffect()

Auf ein Foto in einer Lightbox lassen sich bestimmte Effekte anwenden, z.B. sepia, schwarz/weiß.

### Request

Name	Typ	Beschreibung	Beispiel
effect	<a href="#">String</a>	Der Effekt für das Foto	-
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
photID	<a href="#">Integer</a>	Die ID des Fotos	34556
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Hinzufügen eines Effekts erfolgreich war.	-

## 2.4.5 Lightbox.addPhotoTag()

Ein Photo kann durch hinzufügen eines ‘Tags’ ergänzt werden, welche bei der Suche nach Photos berücksichtigt werden.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	3332
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
tag	<a href="#">Tag</a>	Der Tag, der das Bild erhalten soll	-

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob Hinzufügen des Tags erfolgreich war.	-

## 2.4.6 Lightbox.buy()

Eine Lightbox kann vom Kunden gekauft werden.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Kaufen erfolgreich war.	-

## 2.4.7 Lightbox.copy()

Eine gekaufte Lightbox kann nicht mehr verändert werden, deshalb haben "Agenten" einer Agentur die Möglichkeit sich eine Kopie zu erzeugen.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Integer</a>	Die ID der Lightbox	33443

## 2.4.8 Lightbox.create()

Ein Nutzer kann eine neue Lightbox anlegen

### Request

Name	Typ	Beschreibung	Beispiel
name	<a href="#">String</a>	Name der Lightbox	Projekt2011
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist -	-

		das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Lightbox</a>	Liefert die Erstellte Lightbox	-

## 2.4.9 Lightbox.delete()

Ein Besitzer einer Lightbox kann die Lightbox löschen.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	32233
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Löschen erfolgreich war.	-

## 2.4.10 Lightbox.deletePhoto()

Es können Bilder aus der Lightbox entfernt werden.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	32233
photoID	<a href="#">Integer</a>	Die ID des Photos	33323
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Löschen erfolgreich war.	-

## 2.4.11 Lightbox.enter()

Sobald ein Nutzer eine Lightbox öffnet, tritt er ihr bei.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

Benutzers

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut - null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Eintreten erfolgreich war.	-

**2.4.12 Lightbox.getContactList()**

In einer Lightbox kann das Adressbuch geöffnet werden

**Request**

Name	Typ	Beschreibung	Beispiel
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Integer[]</a>	Eine Liste mit userIDs der Kontakte	-

**2.4.13 Lightbox.getImages()**

Es können alle Photos einer Lightbox angefragt werden.

**Request**

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Photo[]</a>	Alle Fotos einer Lightbox.	-

**2.4.14 Lightbox.getInvitedEmployees()**

Alle eingeladenen Nutzers der Lightbox.

**Request**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">User-Info[]</a>	Eine Liste mit eingeladenen Nutzern	-

**2.4.15 Lightbox.getLightBox()**

Wenn eine Lightbox angefragt wird, werden alle ihre und in ihr enthaltenen Informationen zurückgeschickt, außer die Bilddaten. Die Bilddaten werden über eine separate Anfrage gemacht.

**Request**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Lightbox</a>	Die Lightbox	-

**2.4.16 Lightbox.getList()**

Wenn eine Liste von Lightboxen angefragt wird, werden Informationen zu allen Lightboxen, auf die der User Zugriff hat, zurückgegeben.

**Request**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Lightbox[]</a>	Liste mit Lightboxen	-

## 2.4.17 Lightbox.getPhotoData()

Eine Anfrage an die Lightboxspezifischen Daten des Fotos.

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Fotos.	636366
session	<a href="#">String</a>	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut - null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">LightboxPhotoData</a>	Liefert die Lightboxspezifischen Daten des Fotos.	-

## 2.4.18 Lightbox.getPhotoEffects()

Es kann angefragt werden, was für Photo-Effekte zur Verfügung stehen. Es wird eine liste von den verfügbaren Effekten zurückgeliefert.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">String[]</a>	Liste mit allen Effekten für Photos	-

## 2.4.19 Lightbox.getZip()

Es können alle Photos einer gekauften Lightbox zum Download als Zip-Datei angefragt werden. Dabei kann optional ein Ausgabeformat (jpg,png,...) für die Bilder angegeben werden. (Standard ist jpg). Zurückgegeben wird eine Zip-Datei mit allen Photos der Lightbox.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Enthält das Ergebnis der Anfrage	-
zip	<a href="#">binary</a>	Die Daten der ZIP-Datei	-

## 2.4.20 Lightbox.leave()

Sobald ein Nutzer eine Lightbox schließt, verlässt er sie.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Verlassen erfolgreich war.	-

## 2.4.21 Lightbox.lockPhoto()

Sobald ein Nutzer in einer Lightbox ein Foto anfasst oder er es umbenennen will, wird es für andere Nutzer gesperrt.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
photoID	<a href="#">Integer</a>	Die ID des Lightbox Fotos	2322
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Blockieren des Bildes erfolgreich war.	-

## 2.4.22 Lightbox.movePhoto()

Es können alle Photos in einer Lightbox bewegt werden.

## Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
photoID	<a href="#">Integer</a>	Die ID des Photos	33221
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
x	<a href="#">Float</a>	Die x Koordinate des Photos	30.432
y	<a href="#">Float</a>	Die y Koordinate des Photos	50.345

## Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut - null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Bewegen erfolgreich war.	-

## 2.4.23 Lightbox.print()

Es können entweder alle Photos einer Lightbox gedruckt werden oder es können Photos zum Drucken markiert werden. Der Server bekommt dann optional eine ID-Liste der zu Druckenden Photos mit. Zurückgegeben wird eine PDF-Datei mit den ausgewählten Fotos.

## Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
photoIDs	<a href="#">Integer</a> [ ]	Liste mit IDs der Fotos	-
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

## Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
pdf	<a href="#">binary</a>	Die Daten des PDFs	-
result	<a href="#">SucceedMessage</a>	Enthält das Ergebnis der Anfrage	-

## 2.4.24 Lightbox.rename()

Eine Lightbox kann umbenannt werden.

## Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532

name	<a href="#">String</a>	Neuer Name der Lightbox	Projekt03
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Setzen des Namens erfolgreich war.	-

**2.4.25 Lightbox.renamePhoto()**

Es können Fotos in einer Lightbox individuelle Titel gegeben werden

**Request**

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	1221
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
title	<a href="#">String</a>	Der Titel des Bildes	Wunderschöne Abendsonne

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Bennen einer Lightbox erfolgreich war.	-

**2.4.26 Lightbox.sendLightBoxMessage()**

Innerhalb einer Lightbox können sich die geraden aktiven Bearbeiter in einem Chat austauschen. Chat-Nachrichten sind einem User und einer Lightbox zugewiesen.

**Request**

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
message	<a href="#">Chat-Message</a>	Die Chat-Nachricht.	-
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False

result [SucceedMessage](#) Liefert Aussage, ob das Senden der Nachricht erfolgreich war.

## 2.4.27 Lightbox.setPhotoRating()

Ein Photo kann bewertet werden, wodurch das Interesse und die Kaufpriorität des Kunden verdeutlicht werden kann.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	3332
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
state	<a href="#">String</a>	Der Zustand, der das Bild erhalten soll	-1 0 1

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob Setzen des Zustands erfolgreich war.	-

## 2.4.28 Lightbox.unlockPhoto()

Sobald ein Nutzer in einer Lightbox ein Foto loslässt oder er es nicht mehr umbenennen will, wird es für andere Nutzer gesperrt.

### Request

Name	Typ	Beschreibung	Beispiel
lightboxID	<a href="#">Integer</a>	Die ID der Lightbox	34532
photoID	<a href="#">Integer</a>	Die ID des Lightbox Fotos	2322
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Freigeben des Bildes erfolgreich war.	-

## 2.5 Photo

Enthält Methoden die für Fotos benötigt werden.

## 2.5.1 Photo.addTag()

Ein Photo kann durch hinzufügen eines ‘Tags’ ergänzt werden, welche bei der Suche nach Photos berücksichtigt werden.

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
tag	<a href="#">Tag</a>	Der Tag, der das Bild erhalten soll	-

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob Hinzufügen des Tags erfolgreich war.	-

## 2.5.2 Photo.delete()

Ein Fotograf kann seine Fotos löschen

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Löschen erfolgreich war.	-

## 2.5.3 Photo.deleteTag()

Einem Photo kann ein Tag entfernt werden.

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
tag	<a href="#">Tag</a>	Der Tag, der dem Bild entfernt werden soll	-

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Löschen des Tags erfolgreich war.	-

## 2.5.4 Photo.editCategory()

Einem Photo kann eine Kategorie zugeordnet werden

### Request

Name	Typ	Beschreibung	Beispiel
category	<a href="#">Category</a>	Die Kategorie, die das Photo erhalten soll.	<i>Wunderschöne Abendsonne</i>
photoID	<a href="#">Integer</a>	Die ID des Photos	<i>53536</i>
session	<a href="#">String</a>	Die Session-ID des Benutzers	<i>yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC</i>

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Hinzufügen einer Kategorie erfolgreich war.	-

## 2.5.5 Photo.editData()

Die Daten des Fotos können vom Fotografen bearbeitet werden.

### Request

Name	Typ	Beschreibung	Beispiel
photoData	<a href="#">PhotoData</a>	Die Daten des Photos	-
session	<a href="#">String</a>	Die Session-ID des Benutzers	<i>yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC</i>

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Bearbeiten erfolgreich war.	-

## 2.5.6 Photo.getAgencyData()

Eine Anfrage an die Agenturspezifischen Daten des Fotos.

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Fotos.	<i>636366</i>

session	<a href="#">String</a>	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
---------	------------------------	----------------------------	----------------------------------

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">AgencyPhotoData</a>	Liefert die Agenturspezifischen Daten.	-

**2.5.7 Photo.getCategories()**

Es können alle Kategorien abgefragt werden

**Request**

Name	Typ	Beschreibung	Beispiel
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Category</a>	Liefert das erste Element des Kategorien Baums.	-

**2.5.8 Photo.getFullsize()**

Eine Anfrage an das Foto als Original.

**Request**

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Fotos.	636366
session	<a href="#">String</a>	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

**Response**

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Image</a>	Liefert die Bild-Daten.	-

**2.5.9 Photo.getImagesInCategory()**

Es können Photos nach Kategorie angefragt werden.

**Request**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
categoryID	<a href="#">Integer</a>	Die ID der Kategorie	<i>Wunderschöne Abendsonne</i>
session	<a href="#">String</a>	Die Session-ID des Benutzers	<i>yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC</i>

**Response**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Integer[]</a>	Liste mit photoIDs von Fotos, die in der Kategorie sind.	-

**2.5.10 Photo.getPhoto()**

Eine Anfrage an ein Photo.

**Request**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
photoID	<a href="#">Integer</a>	Die ID des Fotos.	<i>636366</i>
session	<a href="#">String</a>	Die Session-ID des Nutzers	<i>yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC</i>

**Response**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Photo</a>	Liefert das Photo.	-

**2.5.11 Photo.getPhotoData()**

Eine Anfrage an eine PhotoData.

**Request**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
photoID	<a href="#">Integer</a>	Die ID des Fotos.	<i>636366</i>
session	<a href="#">String</a>	Die Session-ID des Nutzers	<i>yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC</i>

**Response**

<b>Name</b>	<b>Typ</b>	<b>Beschreibung</b>	<b>Beispiel</b>
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">PhotoData</a>	Liefert die Photo-Daten.	-

## 2.5.12 Photo.getTags()

Alle Tags eines Fotos

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Photos	53536
role	<a href="#">String</a>	Die jeweilige Rolle	Customer Agency
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">Tag[]</a>	Die Tags zu dem Foto	-

## 2.5.13 Photo.getThumbnail()

Eine Anfrage an ein Thumbnail von einem Foto in einer bestimmten Größe.

### Request

Name	Typ	Beschreibung	Beispiel
photoID	<a href="#">Integer</a>	Die ID des Fotos.	636366
session	<a href="#">String</a>	Die Session-ID des Nutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC
size	<a href="#">Integer</a>	Die ID des Fotos.	800

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	False
result	<a href="#">Image</a>	Liefert ein Thumbnail eines Fotos.	-

## 2.5.14 Photo.search()

Fotos können anhand von Suchbegriffen gesucht werden. Hierbei werden auch die Tags ausgelesen.

### Request

Name	Typ	Beschreibung	Beispiel
search	<a href="#">String</a>	Das Suchwort	Abendsonne
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel

error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Integer[]</a>	photoIDs der gefundenen Photos	-

## 2.5.15 Photo.upload()

Die Fotografen können Fotos hochladen

### Request

Name	Typ	Beschreibung	Beispiel
photo	<a href="#">Image</a>	Die Bilddaten des Fotos.	-
session	<a href="#">String</a>	Die Session-ID des Photographers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Hochladen erfolgreich war.	-

## 2.6 Photographer

Enthält Methoden die für Photographer benötigt werden.

### 2.6.1 Photographer.register()

Die Fotografen legen sich selber Accounts im System an.

### Request

Name	Typ	Beschreibung	Beispiel
profile	<a href="#">Photographer</a>	Das Profil des Fotografens	-

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">String</a>	Der Benutzername des Fotografens	KlausKleber//
session	<a href="#">String</a>	Die Session-ID des	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

Benutzers

## 2.6.2 Photographer.update()

Jeder Fotograf kann seine Profildaten ändern.

### Request

Name	Typ	Beschreibung	Beispiel
photographer	<a href="#">Photographer</a>	Das Profil des Fotografen	-
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Ändern erfolgreich war.	-

## 2.7 User

Enthält Methoden die für User benötigt werden.

### 2.7.1 User.get()

Ein Nutzer wird durch seine ID angefragt

### Request

Name	Typ	Beschreibung	Beispiel
id	<a href="#">Integer</a>	Die ID des Benutzers	45544

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist - das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">User</a>	Die Daten des Nutzers	-

### 2.7.2 User.getBills()

Man kann Daten für seine Rechnungen anfragen.

### Request

Name	Typ	Beschreibung	Beispiel
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

## Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">Bill[]</a>	Die Rechnung für den Nutzer	-

## 2.7.3 User.getUsername()

Ein Nutzer wird durch seinen Benutzernamen angefragt

### Request

Name	Typ	Beschreibung	Beispiel
username	<a href="#">String</a>	Der Benutzername des Benutzers	<i>KlausKleber//</i>

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">User</a>	Die Daten des Nutzers	-

## 2.7.4 User.login()

Ein User, der sich einloggen möchte, wird vom Server geprüft. Bei erfolgreichem Login wird an den Client die SessionID und der Benutzername zurückgesendet.

### Request

Name	Typ	Beschreibung	Beispiel
password	<a href="#">String</a>	Das Passwort des Benutzers	<i>xxxx</i>
username	<a href="#">String</a>	Der Benutzername des Benutzers	<i>KlausKleber//</i>

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">User</a>	Das Profil des Benutzers	-
session	<a href="#">String</a>	Die Session-ID des	<i>yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC</i>

Benutzers

## 2.7.5 User.logout()

Die Session wird beendet.

### Request

Name	Typ	Beschreibung	Beispiel
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob der Logout erfolgreich war.	-

## 2.7.6 User.update()

Jeder User kann seine Profildaten ändern.

### Request

Name	Typ	Beschreibung	Beispiel
User	<a href="#">User</a>	Das Profil des Nutzers	-
session	<a href="#">String</a>	Die Session-ID des Benutzers	yvwH8WrBWqz3mabw2TDDTYDUSwDC54HC

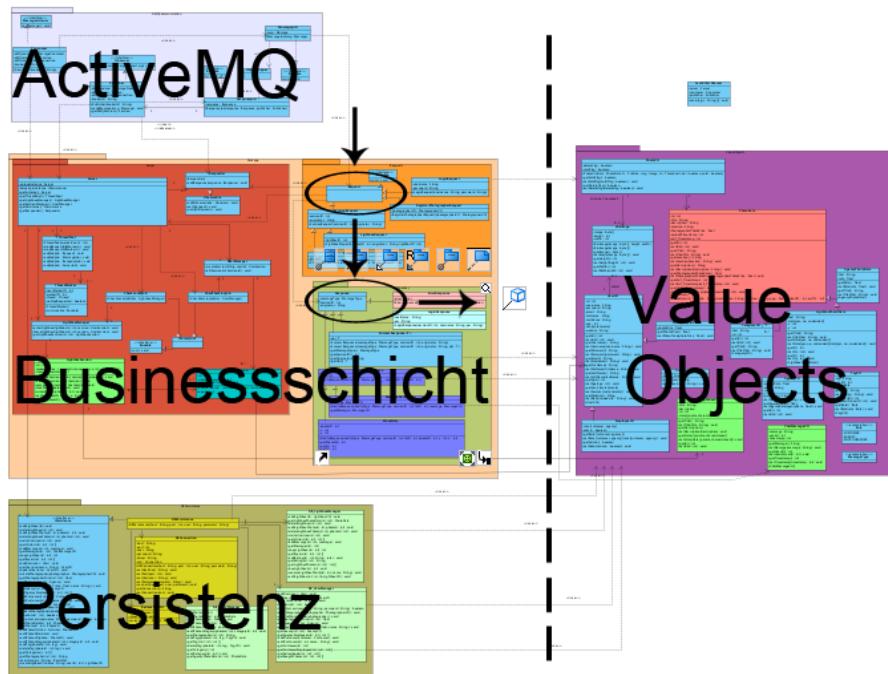
### Response

Name	Typ	Beschreibung	Beispiel
error	<a href="#">ErrorMessage</a>	Beschreibt den aufgetretenen Fehler, wenn einer aufgetreten ist. Ansonsten ist das Attribut null.	-
iserror	<a href="#">Boolean</a>	Gibt an, ob es sich um einen Fehler handelt	<i>False</i>
result	<a href="#">SucceedMessage</a>	Liefert Aussage, ob das Ändern erfolgreich war.	-

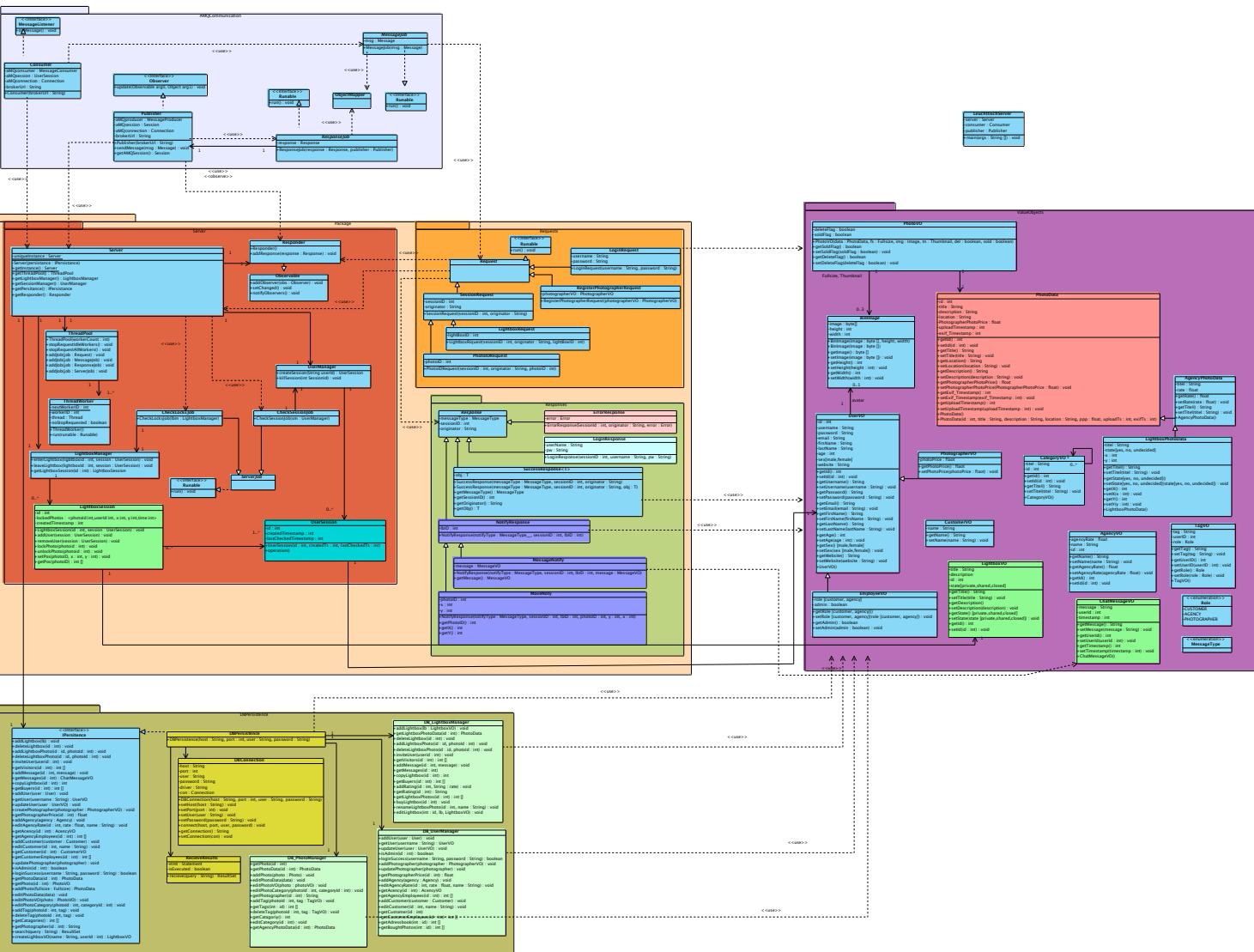


## Aufbau Server

siehe separater Ausdruck



Visual Paradigm for UML Community Edition [not for commercial use]





# **Architekturbeschreibung Server**

Anhang zum Leuchttisch Designdokument

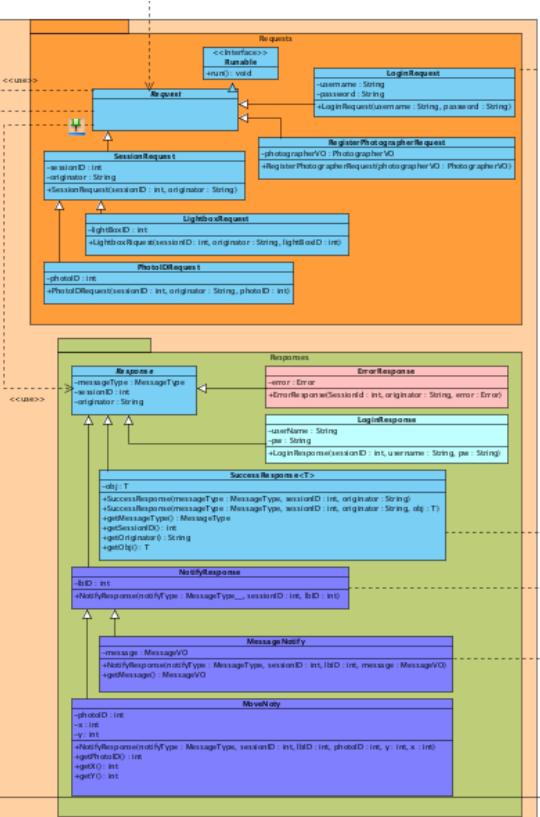
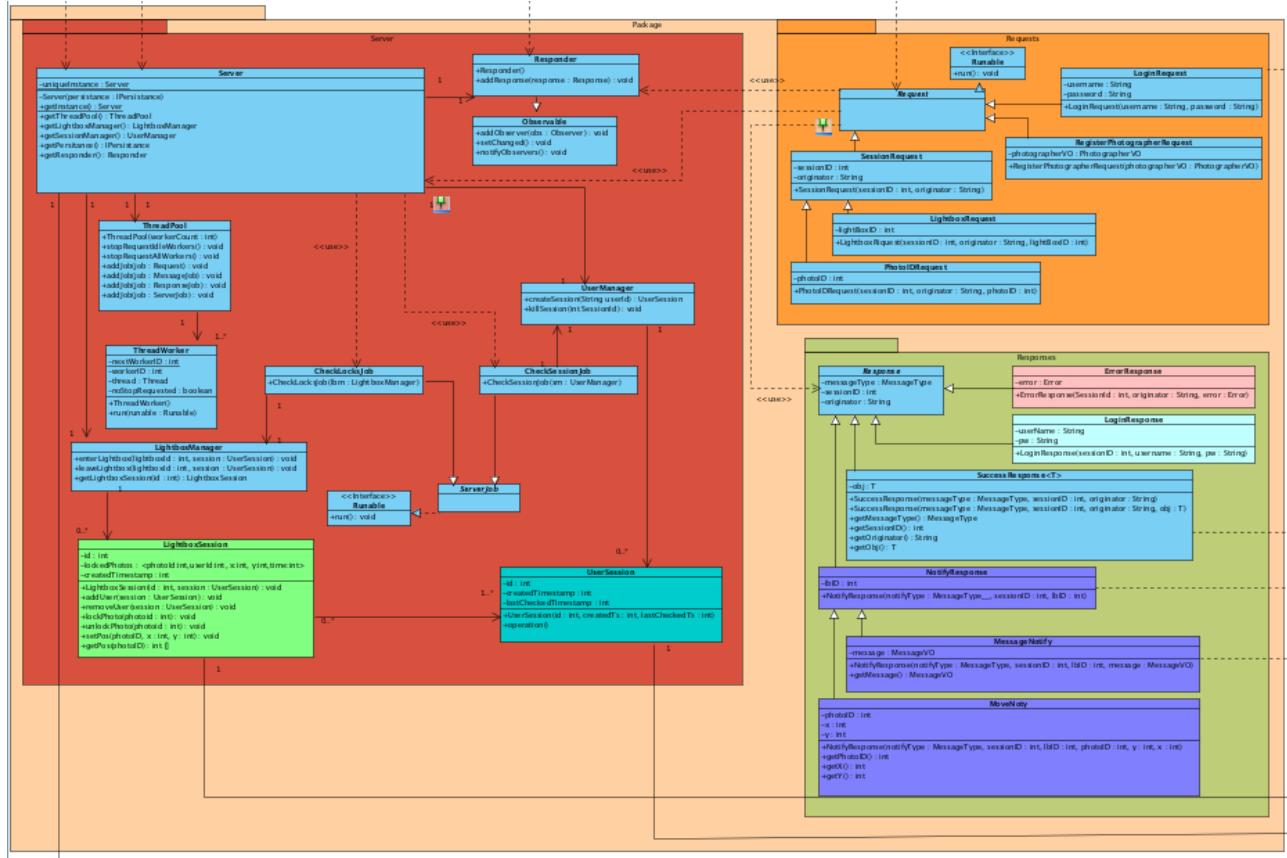
Dieser Angang beschreibt den Aufbau der Architektur des Servers.

## Inhaltsverzeichnis

1 Business.....	3
1.1 Server.....	3
1.2 ThreadPool.....	3
1.3 ThreadWorker.....	4
1.4 Responder.....	4
1.5 LightboxManager.....	4
1.6 UserManager.....	4
1.7 Usersession.....	4
1.8 LightboxSession.....	4
1.9 ServerJob.....	5
1.9.1 CheckLockJob.....	5
1.9.2 CheckSessionJob.....	5
2 Requests.....	6
2.1 SessionRequest extends Request.....	6
2.2 LightboxRequest extends SessionRequest.....	6
2.3 PhotoIDRequest extends SessionRequest.....	6
2.4 LoginRequest extends Request.....	6
2.5 ReqRegisterPhotographer.....	6
2.6 ReqGetStatistics.....	7
2.7 ReqGetAdressbook.....	7
2.8 ReqGetBill.....	7
2.9 ReqBoughtPhotos.....	7
2.10 ReqLogout.....	7
2.11 ReqGetCategories.....	7
2.12 Req GetUser.....	7
2.13 ReqGetPhotoVO.....	7
2.14 ReqGetPhotoData.....	7
2.15 ReGetFullsize.....	8
2.16 ReqGetImage.....	8
2.17 ReqGetThumbnail.....	8
2.18 ReqAgencyPhotoData.....	8
2.19 ReqDeletePhotoData.....	8
2.20 ReqLightboxPhotoData.....	8
2.21 ReqEditPhotoCategory.....	8
2.22 ReqGetTags.....	8
2.23 ReqAddTagVO.....	9
2.24 ReqDeleteTag.....	9
2.25 ReqGetLightboxVO.....	9
2.26 ReqGetLightboxImages.....	9
2.27 ReqCopyLB.....	9
2.28 ReqCreateLightboxVO.....	9
2.29 ReqUploadPhoto.....	9
2.30 ReqEdit3.....	9
2.31 ReqGetImagesFromCategory.....	10

2.32 ReqSetAgencyRate.....	10
2.33 ReqSetPhotoRate.....	10
2.34 ReqCreateAgency.....	10
2.35 ReqCreateCustomer.....	10
2.36 ReqSearch.....	10
2.37 ReqAlive.....	10
2.38 ReqEnterLightbox.....	10
2.39 ReqLeaveLightbox.....	11
2.40 ReqDeleteLightboxVO.....	11
2.41 ReqBuyLightbox.....	11
2.42 ReqSendLBMessage.....	11
2.43 ReqMoveLightboxPhoto.....	11
2.44 ReqRenameLightboxPhoto.....	11
2.45 ReqAddTagLightboxPhoto.....	11
2.46 ReqSetLBPhotoState.....	12
2.47 ReqRenameLB.....	12
2.48 ReqLockLBPhoto.....	12
2.49 ReqUnlockLBPhoto.....	12
2.50 ReqDeleteLightboxPhoto.....	12
2.51 ReqAddLightboxPhoto.....	12
2.52 ReqInviteLBEmployee.....	12
2.53 ReqGetLBzipped.....	12
2.54 ReqGetAllinvited.....	13
3 Responses.....	13
3.1 SuccessResponse.....	13
3.2 NotfyResponse.....	13
3.3 ErrorResponse.....	13
3.4 LoginResponse.....	13

# 1 Business



## 1.1 Server

Der Server hat einen ThreadPool, LightboxManager, SessionManager und Responder. Er wird mit dem Entwurfsmuster Singleton erstellt und erzeugt bei der einmaligen Initialisierung einen CheckLockJob und CheckSessionJob welche er dem ThreadPool hinzufügt. Der Server kennt außerdem einen Umsetzung des Interface IPersistence.

## 1.2 ThreadPool

Der ThreadPool besitzt eine Zahl an ThreadWorkern, welche zur Verarbeitung von verschiedenen Jobs bereitgestellt werden. Dabei unterscheiden wir verschiedene Jobs welche darin abgearbeitet werden:

- MessageJob
- ResponseJob
- ServerJob
- Request

## 1.3 ThreadWorker

Der ThreadWorker erhält dabei den auszuführenden und zuvor konstruierten Job und arbeitet diesen ab.

## 1.4 Responder

Der Responder erbt von Observable und stellt die Methode addResponse() zur Verfügung. Mit ihr werden Responses zur weiteren Verarbeitung hinterlegt. Damit wird in erster Linie die Delegation von fertig verarbeiteten Anfragen des Servers, an die Kommunikationsschicht bereitgestellt.

## 1.5 LightboxManager

Der LightboxManager hält alle bereits initialisierten Lightbox-Sessions und stellt grundlegende Funktionalitäten zur Interaktion mit Lightboxen bereit. Dabei handelt es sich um die Methoden:

- enterLightbox: Wird aufgerufen wenn ein Benutzer in eine Lightbox beitritt. Dabei muss er überprüfen ob eine Lightboxsession bereits existiert und diese gegebenenfalls erzeugen.
- leaveLightbox: Wird aufgerufen wenn ein Benutzer die Lightbox verlässt und überprüft falls nötig, ob eine Lightboxsession geschlossen werden kann, wenn keine User-Session diese mehr nutzt.

## 1.6 UserManager

Der UserManager hält alle bereits initialisierten User-Session und stellt grundlegende Funktionalitäten bereit.

- createSession: Erzeugt eine Usersession und hält diese.
- killSession: Zerstörte eine existierende Usersession.

## 1.7 Usersession

Die Usersession identifiziert die Session eines Benutzers eindeutig. Es hält eine Referenz auf ein spezifisches UserVO. Außerdem hat jede Session, welche durch eine eigene ID gekennzeichnet wird, eine createdTimestamp und lastCheckedTimestamp. Der lastCheckedTimestamp wird dazu verwendet um regelmäßige ReqAlive-Anfragen des Clients zu hinterlegen.

## 1.8 LightboxSession

Die LightboxSession ist anhand einer ID eindeutig identifizierbar und hält alle kollaborativen Status, der innerhalb einer Lightbox befindlichen Photos. Dabei hält die LightboxSession alle "gespererten" Photos, welche momentan verwendet werden, mit ihrer nicht-persistenten Position innerhalb der aktiven Lightbox. Sie besitzt folgende Methoden:

addUser / removeUser: Hinzufügen oder Entfernen von aktiven Benutzern zu einer LightboxSession.

lockPhoto / unlockPhoto: Sperren oder Entsperren eines Photos innerhalb einer LightboxSession.

setPos / getPos: Setzt und liefert die unpersistennten Positionenangaben eines Bildes innerhalb einer Lightbox.

## 1.9 ServerJob

Die ServerJobs sind Jobs welche ein Server zur Kontrolle der Laufzeit bei der Initialisierung des Servers startet und welche permanent laufen. Dabei unterscheiden wir zwei Spezialisierungen:

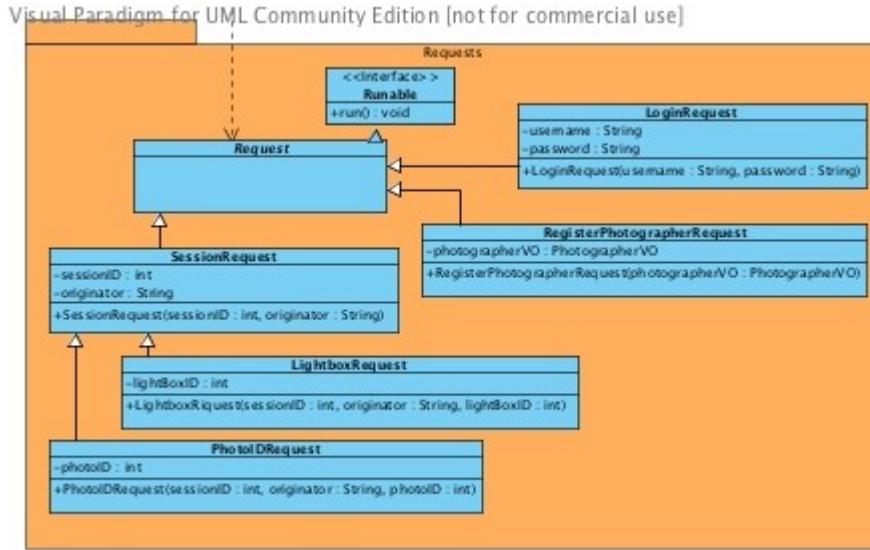
### 1.9.1 CheckLockJob

Der CheckLockJob wurde bei der Initialisierung des Servers in den ThreadPool gegeben und überprüft in bestimmten Intervallen ob die Sperre eines Photos, innerhalb einer Lightbox, zu lange unverändert gesetzt wurde. Damit wollen wir verhindern, dass bei unvorhergesehenen Verbindungsabbrüchen das Photo versehentlich gesperrt bleibt und allen anderen Benutzern die Interaktion mit diesem Objekt verweigert wird.

### 1.9.2 CheckSessionJob

Überprüft regelmäßig ob eine UserSession abgelaufen ist. Dabei wird die Differenz zwischen lastCheckedTimestamp und aktueller Zeit überprüft. Wenn die abgelaufene Zeit einer Session einen vorgegebenen Maximalwert überschreitet, wird die Session des Benutzers vom Server beendet.

## 2 Requests



Requests werden durch MessageJobs erzeugt. Sie implementieren das Interface „Runnable“ und können somit innerhalb eines ThreadPools abgearbeitet werden. Sie tragen somit auch zu einer besseren Skalierung bei. Der Request stellt einen wesentlichen Baustein unserer Architektur dar. In ihm befindet sich die eigentliche Prozesslogik zur Abarbeitung verschiedener Aufgaben und Anforderungen an den Server durch den Client.

Viele der folgenden Ausprägungen und ihre Zusammenhänge werden in der Grafik im **Anhang Request-Übersicht Server** dargestellt.

### 2.1 SessionRequest extends Request

Viele der folgenden Requests besitzen als Standardparameter eine sogenannte SessionID und Originator. Diese SessionID ist in der Laufzeit für jeden Client eindeutig und identifiziert diesen. Der Originator dient dem Client zur eindeutigen Identifizierung einer abgesendeten Anforderung. Sie werden in den folgenden Parameter-Auflistungen nicht mehr erwähnt und werden in der Generalisierung SessionRequest zusammengefasst.

### 2.2 LightboxRequest extends SessionRequest

Ein LightboxRequest ist eine Spezialisierung des SessionRequest. In ihr werden verschiedene Requests zusammengefasst, mit welchen auf die Verarbeitung von Lightboxen eingegangen wird. Sie hält neben den Standardparametern eine zusätzliche lbID.

## 2.3 PhotoIDRequest extends SessionRequest

Ein PhotoIDRequest ist eine Spezialisierung des SessionRequest und hält eine zusätzliche PhotoID. Mit ihr werden Operationen generalisiert welche auf die Verarbeitung von Photos innerhalb des Systems ausgerichtet sind.

## 2.4 LoginRequest extends Request

Der Request hat die Aufgabe einen bis dahin dem Server unbekannten Client zu authentifizieren, seine Angaben zu kontrollieren und gegebenenfalls eine SessionID zu erzeugen.

Parameter: Benutzername, Passwort

## 2.5 ReqRegisterPhotographer

Mit Hilfe dessen haben Fotografen die Möglichkeit sich in dem System zu registrieren. Dabei wird das vom Client übergebene PhotographerVO verarbeitet, überprüft und persistent gemacht.

Parameter: Benutzerdaten des Fotografen

## 2.6 ReqGetStatistics

Request liefert verschiedene statistische Auswertung.

## 2.7 ReqGetAddressbook

Erstellt das Adressbuch des Benutzers anhand der übergebenen Standardparameter und dessen Abhängigkeiten und Rolle innerhalb des Systems.

## 2.8 ReqGetBill

Erzeugt Rechnungsinformation des Benutzers anhand der übergebenen Standardparameter.

## 2.9 ReqBoughtPhotos

Liefert alle jemals gekauften Fotos eines Kunden auf Anfrage eines Angestellten.

## 2.10 ReqLogout

Meldet den aktuellen Benutzer innerhalb des Systems ab und sorgt für das Beenden der laufenden Session und deren Beteiligung an anderen Systemprozessen.

## 2.11 ReqGetCategories

Erzeugt den Kategoriebaum des Systems, innerhalb dessen der Bildbestand angeordnet ist.

## 2.12 Req GetUser

Der Request liefert allgemeine Informationen über einen Benutzer des Systems anhand einer übergebenen UserID und kann somit zum Beispiel zur Darstellung eines Benutzerprofils verwendet werden.

Parameter: UserID

## 2.13 ReqGetPhotoVO

Erzeugt ein ValueObject zu einem bestimmten Bild.

Parameter: PhotoID des Bildes

## 2.14 ReqGetPhotoData

Der Request sorgt für die Bereitstellung von Bildinformationen eines bestimmten Bildes.

Parameter: PhotoID des Bildes

## 2.15 ReGetFullsize

Holt sich ein Bild in voller Größe aus der persistenten Datenhaltung.

Parameter: PhotoID des Bildes

## 2.16 ReqGetImage

Holt sich ein Bild in Standard-Größe aus der persistenten Datenhaltung. Parameter: PhotoID des Bildes

## 2.17 ReqGetThumbnail

Holt sich die verkleinerten Darstellungen aus der persistenten Datenhaltung.

Parameter: PhotoID des Bildes

## 2.18 ReqAgencyPhotoData

Holt sich die AgencyPhotoData eines bestimmten Bildes aus der persistenten Datenhaltung.

Parameter: PhotoID des Bildes

## 2.19 ReqDeletePhotoData

Löscht die hinterlegten PhotoData eines bestimmten Bildes.

Parameter: PhotoID des Bildes

## 2.20 ReqLightboxPhotoData

Holt sich die LightboxPhotoData eines bestimmten Bildes, welches zu einer Lightbox gehört.

Parameter: PhotoID des Bildes, LbID der Lightbox

## 2.21 ReqEditPhotoCategory

Setzt anhand der übergebenen Kategorie-Nummer, die Zugehörigkeit eines Bildes.

Parameter: PhotoID des Bildes, CategoryID einer Kategorie

## 2.22 ReqGetTags

Holt sich alle gesetzten Tags zu einem angegebenen Bild, anhand der Bild-ID und einer „Rollenbezeichnung“.

Parameter: PhotoID des Bildes, Role

## 2.23 ReqAddTagVO

Fügt einem Bild ein bestimmtes Tag hinzu, anhand der übergebenen Bild-ID und eines TagVO-Objektes, welches das eigentliche Tag enthält.

Parameter: PhotoID des Bildes, TagVO

## 2.24 ReqDeleteTag

Löscht ein bereits von einem Benutzer gesetzten Tag wieder.

Parameter: PhotoID des Bildes, TagVO

## 2.25 ReqGetLightboxVO

In dem Request wird ein LightboxVO einer bestimmten Lightbox erzeugt.

Parameter: LbID der Lightbox

## 2.26 ReqGetLightboxImages

Der Request hat die Aufgabe, alle Bilder welche sich innerhalb einer bestimmten Lightbox befinden zu suchen.

Parameter: LbID der Lightbox

## 2.27 ReqCopyLB

Kopiert eine Lightbox mit den gleichen Bildinhalten und Angaben.

Parameter: LbID der Lightbox

## 2.28 ReqCreateLightboxVO

Der Request wird zum erzeugen eines neuen LightboxVO verwendet und wird hauptsächlich für die Erstellung einer neuen Lightbox, anhand eines übergebenen Namens verwendet.

Parameter: Name der neuen Lightbox als String

## 2.29 ReqUploadPhoto

Der Request reagiert auf den Uploadvorgang eines neuen Bildes und sorgt für die Ablage dieses und Registrierung in dem System.

Parameter: Fullsize Bilddaten

## 2.30 ReqEdit3

Überschreibt die PhotoData eines Bildes, anhand den übergebenen Werten innerhalb eines PhotoData.

Parameter: PhotoData eines Bildes

## 2.31 ReqGetImagesFromCategory

Request liefert alle Bilder einer übergebenen Kategorie.

Parameter: CatID einer Kategorie

## 2.32 ReqSetAgencyRate

Setzt den von einer Agentur veranschlagten Wert auf alle vermittelten Bilder.

Parameter: Rate einer Agentur als double

## 2.33 ReqSetPhotoRate

Setzt den von einer Agentur veranschlagten Wert auf ein bestimmtes Bild.

Parameter: AgencyPhotoData eines Bildes

## 2.34 ReqCreateAgency

Erzeugt eine Agentur innerhalb des Systems anhand eines übergebenen Namens.

Parameter: Name als String

## 2.35 ReqCreateCustomer

Erzeugt einen neuen Kunden innerhalb des Systems anhand eines übergebenen CustomerVO.

Parameter: CustomerVO mit allen Informationen eines Kunden

## 2.36 ReqSearch

Request wird zum Durchsuchen des Bildbestandes verwendet und erwartet die Übergabe eines Suchstrings.

Parameter: Search String

## 2.37 ReqAlive

Request welcher dem Server bestätigt, dass ein Client noch mit dem Server verbunden ist. Dies wird anhand der übergebenen Standardparameter verifiziert, um wenn nötig, einen nicht mehr verbundenen Client aus der Sessionverwaltung auszutragen und dies zu publizieren.

## 2.38 ReqEnterLightbox

Der Request wird aufgerufen, wenn ein Benutzer in eine Lightbox-Session eintreten will. Innerhalb des Systems müssen, wenn nötig, eine neue Lightbox-Session erzeugt werden und dies publiziert werden.

Parameter: LbID einer Lightbox

## 2.39 ReqLeaveLightbox

Wenn ein Benutzer einer Lightbox-Session nicht mehr beiwohnen will, wird dieses Request erzeugt und innerhalb des Systems verarbeitet.

Parameter: LbID einer Lightbox

## 2.40 ReqDeleteLightboxVO

Wenn eine Lightbox gelöscht werden soll wird dieses Request erzeugt und abgearbeitet. Dabei müssen auch laufende Lightbox-Sessions behandelt werden können.

Parameter: LbID einer Lightbox

## 2.41 ReqBuyLightbox

Der Request wird benötigt wenn ein Kunde eine bestimmte Lightbox erwerben möchte. Der Request muss die State der gerade erworbenen Lightbox verändern.

Parameter: LbID einer Lightbox

## 2.42 ReqSendLMessage

Der Request ist für die Verarbeitung einer abgesendeten Nachricht innerhalb des Chat-Systems einer Lightbox zuständig. Es verarbeitet das eingehende MessageVO.

Parameter: LbID einer Lightbox, MessageVO mit Nachricht

## 2.43 ReqMoveLightboxPhoto

Der Request berücksichtigt die Veränderung der Bildpositionen eines Bildes, innerhalb einer Lightbox durch den Benutzer.

Parameter: LbID einer Lightbox, PhotoID des Bildes, x Position als int, y Position als int

## 2.44 ReqRenameLightboxPhoto

Der Request verarbeitet die Anfrage ein Bild innerhalb einer Lightbox umzubenennen.

Parameter: LbID einer Lightbox, PhotoID des Bildes, Name als String

## 2.45 ReqAddTagLightboxPhoto

Der Request fügt ein eingegangenes TagVO einem Bild welches innerhalb einer Lightbox bearbeitet wird hinzu.

Parameter: LbID einer Lightbox, PhotoID eines Bildes, TagVO

## 2.46 ReqSetLBPhotoState

Der Request verarbeitet das Setzen eines Status durch einen Kunden, ob er das Bild kaufen will oder nicht.

Parameter: LbID einer Lightbox, PhotoID eines Bildes, State als boolean

## 2.47 ReqRenameLB

Der Request ermöglicht das umbenennen einer Lightbox.

Parameter: LbID einer Lightbox, lbName String

## 2.48 ReqLockLBPhoto

Der Request wird vor jeder Interaktion eines Benutzers mit einem Photo innerhalb der Lightbox aufgerufen und sorgt für die Sperrung des Bildes. Damit wird garantiert das kein anderer Benutzer eine Interaktion mit dem selben Photo eingehen kann.

Parameter: LbID einer Lightbox, PhotoID des Bildes

## 2.49 ReqUnlockLBPhoto

Der Request entsperrt das am Anfang einer Interaktion gesetzte Photo und ermöglicht die

Weiterverarbeitung durch andere.

Parameter: LbID einer Lightbox, PhotoID des Bildes

## 2.50 ReqDeleteLightboxPhoto

Der Request verarbeitet die Anfrage ein Bild innerhalb einer Lightbox zu Löschen.

Parameter: LbID einer Lightbox, PhotoID des Bildes

## 2.51 ReqAddLightboxPhoto

Der Request verarbeitet die Anfrage ein neues Bild einer Lightbox hinzuzufügen.

## 2.52 ReqInviteLBEmployee

Der Request lädt einen Benutzer des Systemes in eine Lightbox ein.

Parameter: LbID einer Lightbox, EmployeeID eines Benutzers

## 2.53 ReqGetLBzipped

Der Request verarbeitet die nach dem Kauf bereitgestellte Funktion eine gesamte Lightbox als ZIP herunter zu laden.

Parameter: LbID einer Lightbox

## 2.54 ReqGetAllInvited

Der Request liefert alle eingeladenen Employee einer Lightbox

Parameter: LbID einer Lightbox

# 3 Responses

Responses sind das Pendant zu den Requests und werden durch diese erzeugt. Jeder Response enthält den MessageType die ID der Session des Benutzers und einen Originator mit welchem der Client die eingehenden Anfragen identifizieren kann. Im Response werden Daten gehalten, die wieder an den Client gesendet werden. Im Allgemeinen sind Responses unabhängig vom Objekttyp, da sie generalisiert sind.

## 3.1 SuccessResponse

Nach der erfolgreichen Verarbeitung eines Requests, wird ein passendes SuccessResponse erzeugt. Bei einem SuccessResponse werden aus allgemeinen Objektdaten unter Verwendung der verschiedenen ValueObjects die genauen Datenobjekte erstellt, die durch den Response erstellten konkreten Datenobjekte können unabhängig von ActiveMQ weiterverwendet werden und werden gehalten.

## 3.2 NotifyResponse

Für Daten, welche interaktiven Änderungen unterliegen, gibt es NotifyResponses. Wir sprechen dann von einem Notify, wenn Nachrichten an einen Client verschickt werden, welche er nicht durch einen zuvor selbst ausgelösten Request erwartet. Auch diese Antworten enthalten auf Basis der Responses wieder den MessageType und die Session-ID, sowie den Originator. Die NotifyResponses bekommen stets die ID der Lightbox übergeben, sodass eine eindeutige Zuordnung möglich ist. Die Klasse NotifyResponse, welche Response implementiert, hat für jede Art eines Notifiers eine eigene Spezialisierung.

Die Trennung von SuccessResponses und NotifyResponses ermöglicht Variabilität bei den Topics von ActiveMQ.

## 3.3 ErrorResponse

Bei Fehlern welche während der Verarbeitung eines Requests auftreten, können ErrorResponses erzeugt werden. Dabei handelt es sich um eine Spezialisierung einer Response, welche einen Fehler enthalten kann.

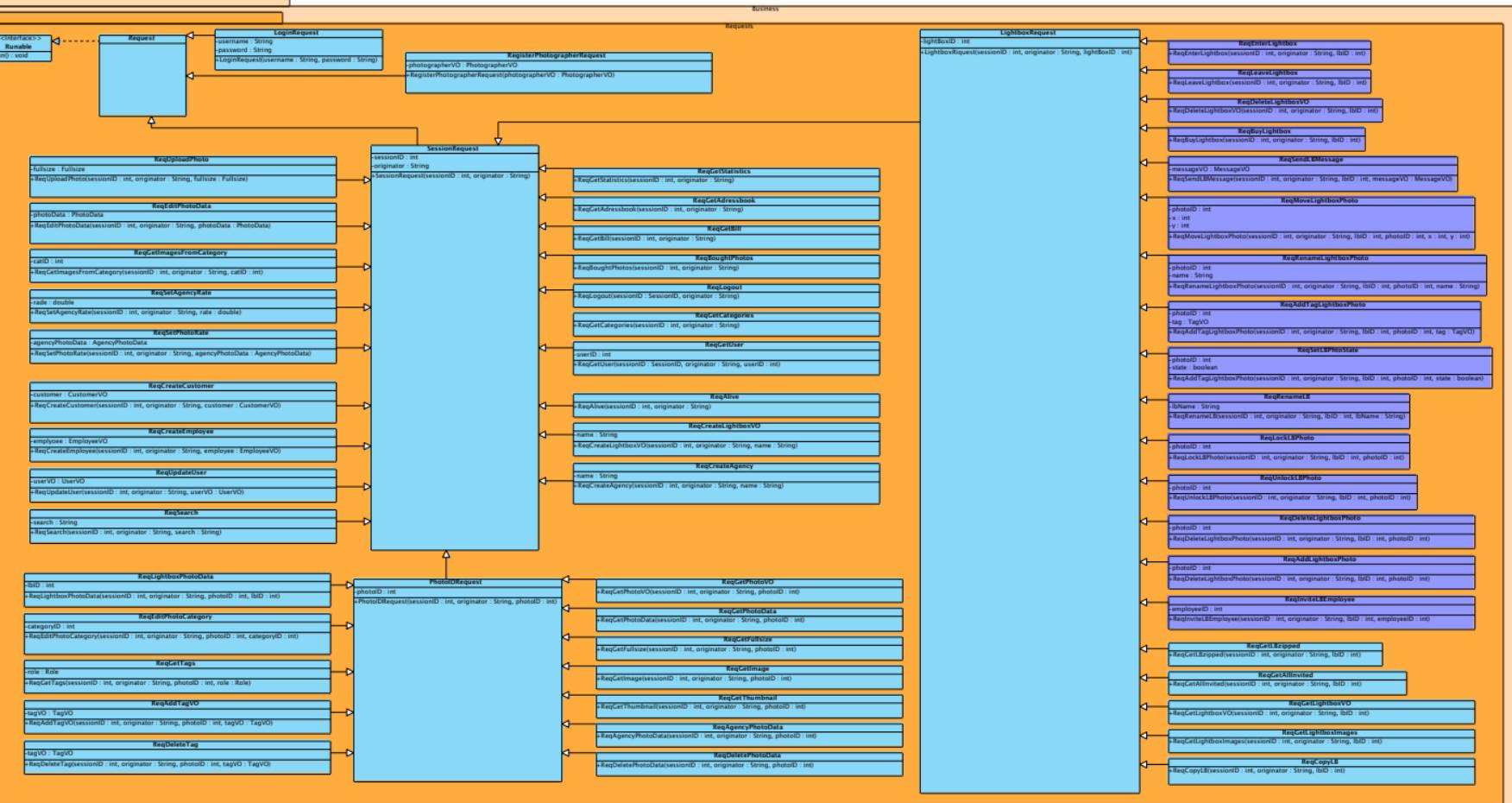
## 3.4 LoginResponse

Die LoginResponse ist eine weitere Spezialisierung der Response. Aufgrund der Tatsache dass zum Zeitpunkt der Anmeldung eines Benutzers, noch keine SessionID existiert mit welcher Nachrichten über den ActiveMQ Service an den Client adressiert werden können, muss dieser Fall besonders behandelt werden.



# **Request-Übersicht Server**

Anhang zum Leuchttisch Designdokument





# Persistenz

Anhang zum Leuchttisch Designdokument

Dieser Angang beschreibt die Persistenz-Schicht des Servers und definiert den Aufbau der relationalen Datenbank.

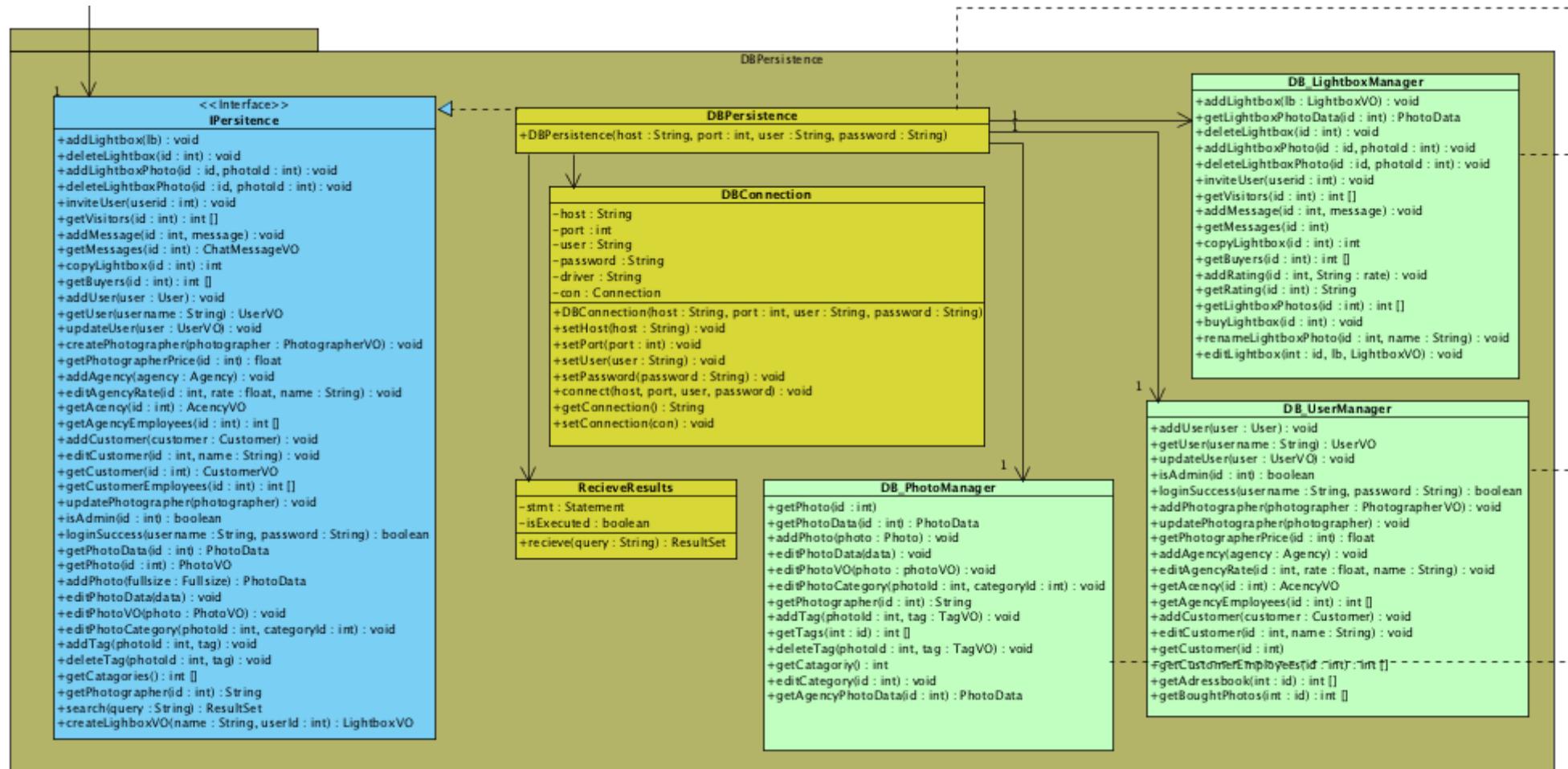
## Inhaltsverzeichnis

1 Persistenzlayer.....	2
2 Datenbank.....	4
2.1 Entity-Relationship-Model.....	4
2.2 Relationales Modell.....	6

# 1 Persistenzlayer

Die Klasse DBPersistence, welche vom Interface IPersistence realisiert wird, baut zunächst die Datenbankverbindung auf, dafür werden Host, Port, User und Password an DBConnection gereicht, deren connect()-Methode die Verbindung zur Datenbank aufbaut. Die Resultat der Datenbankanfragen werden von RecieveResults geliefert, dazu wird der Anfragestring als Query übergeben und ein ResultSet aus java.sql.ResultSet zurückgeliefert. Um dies zu erhalten wird auf der bestehenden Verbindung zunächst ein Statement erzeugt, worauf dann der Query ausgeführt wird.

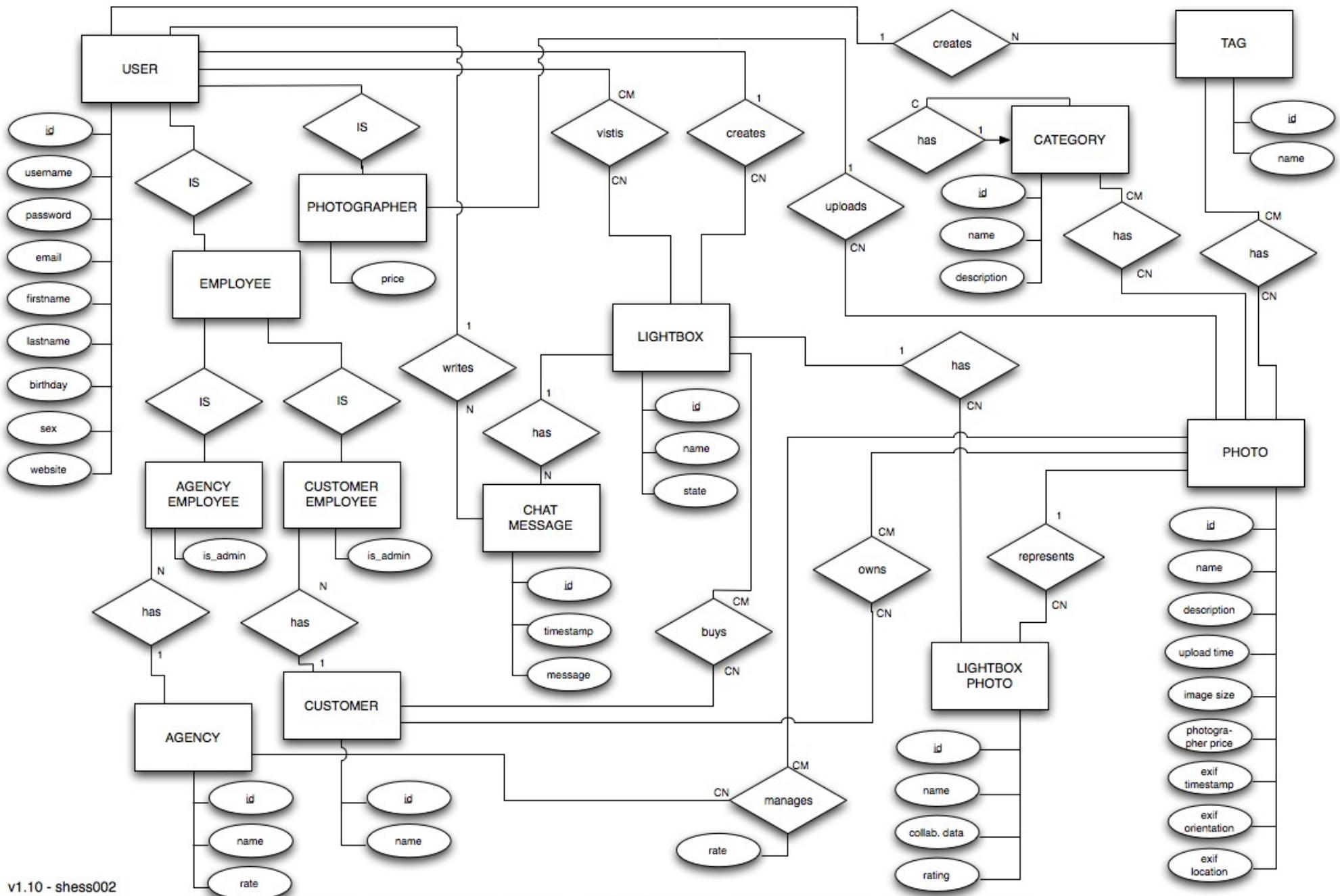
Die einzelnen Methoden der Datenbankanfragen sind extra aufgeteilt in Klassen für User-Querys, Photo-Querys und Lightbox-Querys, die Anfragen werden vom Interface IPersistence weitergereicht. Die Methoden verarbeiten die Results und liefern (je nach Anfrage) fertige ValueObjects zurück.



## 2 Datenbank

### 2.1 Entity-Relationship-Model

Zur Persistenz der Daten dient eine PostgreSQL-Datenbank, welche vom Server angesteuert wird. Auf der Datenbank werden sämtliche Daten gespeichert, die dauerhaft zur Verfügung stehen sollen. Dazu zählen neben den Userdaten, mit Agentur- und Customer-Daten samt Zugehörigkeit auch die Fotos - wobei die Fotodateien selbst auf dem Server gesichert sind, in der Datenbank dann der Speicherort sowie alle zugehörigen Daten, wie Name, Beschreibung oder EXIF-Daten.



## 2.2 Relationales Modell

Die Userdaten bestehen aus einer ID, welche als SERIAL hochgezählt wird und der PRIMARY KEY ist, zudem ist der USERNAME noch UNIQUE. Das Geburtsdatum ist ein DATE, aus welchem das Alter berechnet werden kann. Die Agencys und Customer haben je auch eine ID als SERIAL und einen Name, die Agency zusätzlich ihre Preisrate. Deren Employees werden als Verweis auf Agency-ID/Customer-ID und User-ID je als FOREIGN KEY und in Kombination als PRIMARY KEY in extra Tabellen gesichtet - zudem haben Employees je noch einen BOOLEAN, ob sie Admin sind.

Da jedes Foto eine bestimmte Kategorie und Tags von verschiedenen Usern hat, werden auch diese samt Zugehörigkeit in der Datenbank gesichert. Die Zugehörigkeit von Fotos zu einer bestimmten Lightbox samt den damit verbunden Daten, wie der individuelle Foto-Titel innerhalb einer Lightbox sind eine extra Tabelle. Auch Chat-Nachrichten aus Lightboxen werden persistent gesichert.

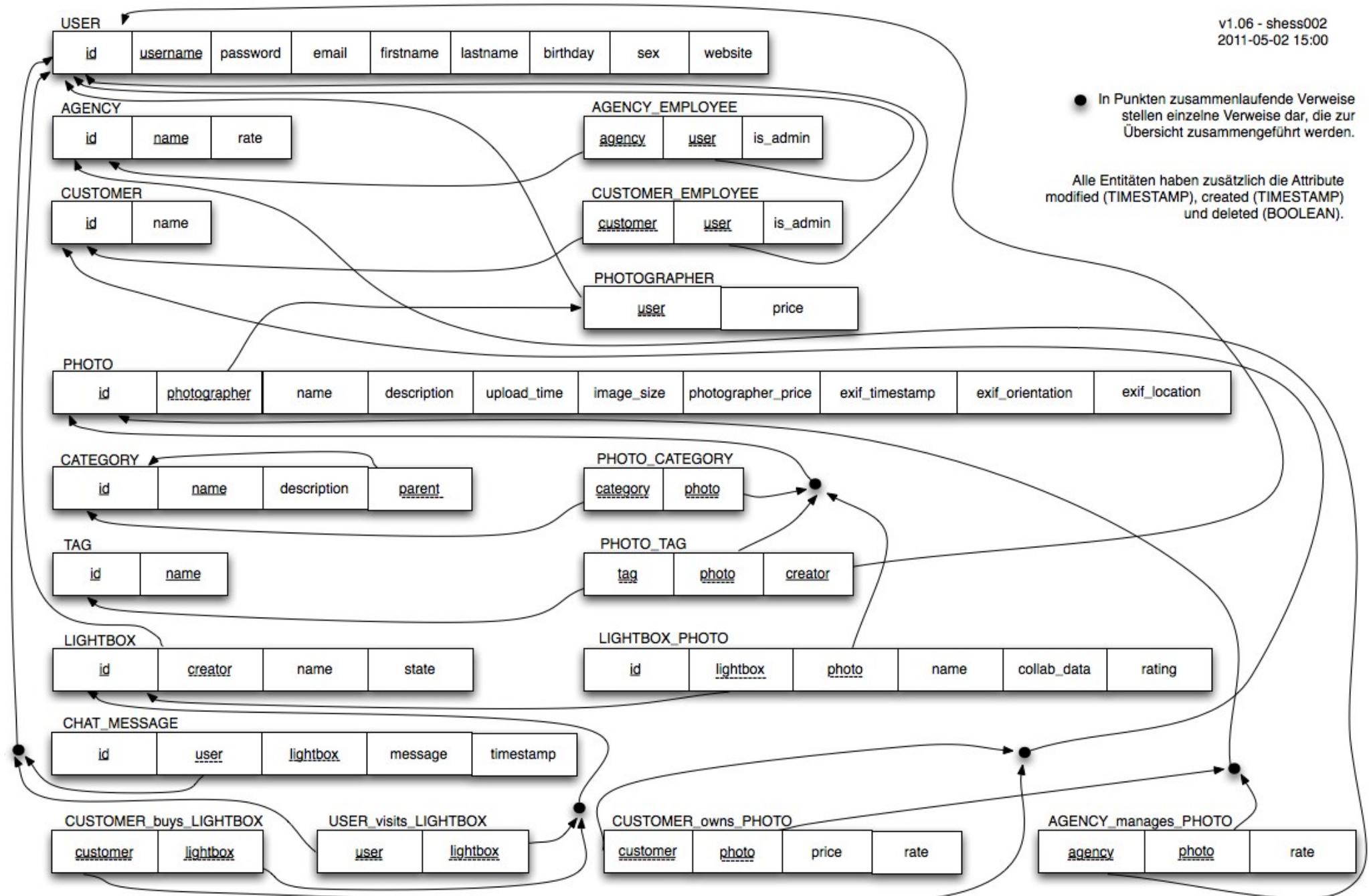
Auch bei Photos ist die ID ein SERIAL, der Photograph wird als FOREIGN KEY zur ID eines Photographer gesichert, wobei dessen ID ein FOREIGN KEY einer User-ID ist. Die Kategorien der Fotos sind haben eine ID als PRIMARY KEY, zudem sind sie über den Kategoriennamen und die Elternkategorie UNIQUE. Die Elternkategorie ist ein KEY auf eine weitere Kategorie-ID, sodass sich ein Kategorienbaum erzeugen lässt. Auf der obersten Kategorieebene ist dieser Eintrag leer. Die Beziehung zwischen Photo und Category gesichtet durch eine Tabelle, welche jeweils die beiden IDs als FOREIGN KEY und in Kombination als PRIMARY KEY enthält.

Eine Lightbox hat eine ID als SERIAL, eine User-ID als FOREIGN KEY für den Ersteller der Lightbox und einen Status, welcher sich durch Besucher oder den Kauf ergibt. Da Photos in Lightboxen andere Eigenschaften haben können als globale Photos, werden Lightbox-Photos mit ihrer eigenen ID versehen und verweisen auf die zugehörige Lightbox und das originale Photo. Innerhalb der Lightbox können sie nun mit einem Rating versehen werden oder umbenannt. An der Lightbox hängt zudem der Chat, dessen Nachrichten durchnummierter werden, auf eine Lightbox und einen User als FOREIGN KEY verweisen und neben der reinen Textnachricht auch den genauen TIMESTAMP des Absendens enthalten.

In weiteren extra Tabellen wird gesichert welche User Zugriff auf bestimmte Lightboxen haben, welche Lightboxen bereits gekauft wurden, welche Customer was für Fotos gekauft haben sowie die speziellen Agency-Photos, welche beispielsweise die bestimmte Preisrate enthält.

Die Tabellen CUSTOMER\_buys\_LIGHTBOX, USER\_visits\_LIGHTBOX, CUSTOMER\_owns\_PHOTO und AGENCY\_manages\_PHOTO stellen die Bezüge stets durch Verweise auf die IDs als FOREIGN KEY sicher und in Kombination sind beide IDs PRIMARY KEY. CUSTOMER\_owns\_PHOTO enthält zusätzlich die Attribute Price und Rating, welche den Kaufpreis und die Rate der Agency enthalten - diese sind keine Fremdschlüssel, da sich die Preise der Fotos und Raten der Agentur ändern können.

Jede Entität hat zudem die Attribute modified (TIMESTAMP), created (TIMESTAMP) und deleted (BOOLEAN) um diese zu sortieren und kategorisieren und als gelöscht markierte Daten nicht mehr auszuliefern. Daten werden nicht komplett aus der Datenbank entfernt, sondern mit einem Flag versehen, ob sie bereits "gelöscht" wurden - ist diese Flag gesetzt, werden die Daten bei Result-Anfragen ignoriert.





# **Exemplarische Architektur- Durchstiche**

Anhang zum Leuchttisch Designdokument  
große Version der Sequenzdiagramme siehe separater Ausdruck

Dieses Dokument beinhaltet Sequenzdiagramme, die exemplarische Architektur-Durchstiche beschreiben.

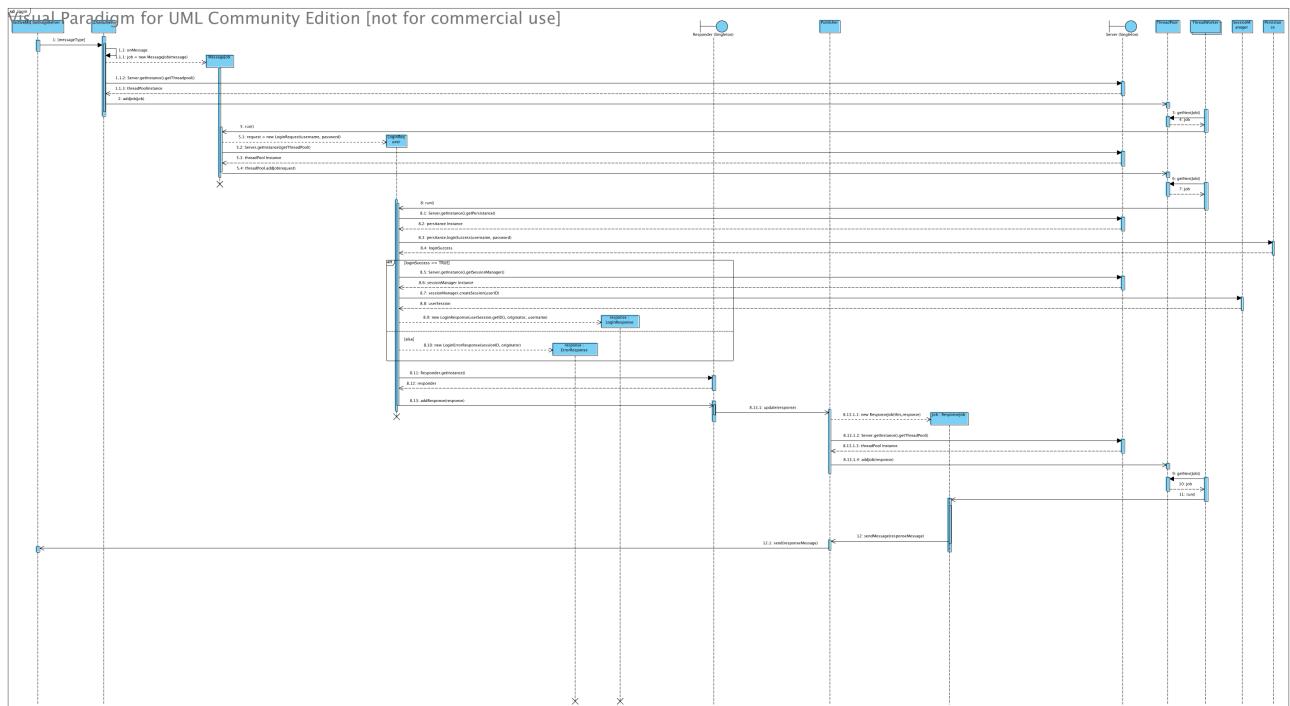
## Inhaltsverzeichnis

1 Server.....	2
1.1 Login.....	2
1.2 Benutzerdaten anfragen.....	3
1.3 Foto-Daten anfragen.....	3
1.4 Lightbox erstellen.....	4
1.5 Fotograf registriert sich.....	4
1.6 Hochladen eines Bildes.....	5
2 Client.....	6
2.1 View am Consumer als Kommunikationsteilnehmer anmelden.....	6
2.1.1 Nachricht senden.....	7
2.1.2 Nachricht empfangen.....	8

# 1 Server

Bei den folgenden Sequenzdiagrammen gibt es Standardabläufe, die sich von Sequenzdiagramm zu Sequenzdiagramm nicht unterscheiden. Wenn eine Nachricht beim Consumer des Servers eingeht, wird für diese direkt ein MessageJob erstellt, welcher die Message als Instanz hält. Dieser wird dann in den ThreadPool gelegt und im Anschluss durch einen freien ThreadWorker über die run()-Methode angestoßen. Daraufhin erstellt der MessageJob anhand des entsprechenden MessageType einen Request, welcher wieder im ThreadPool landet und auf seine Ausführung wartet. In der run()-Methode eines Requests werden dann bestimmte Aktionen auf dem Server ausgeführt (z.B. ein Photo speichern, eine Suchanfrage starten, ...) und im Anschluss wird ein Response erstellt. Der Response hält das Objekt, welches wieder an den Client zurückgesendet werden soll. Dieser Response wird dann zum Responder gesendet, wodurch der Publisher benachrichtigt wird, welcher mit dem Response einen ResponseJob erstellt. Dieser ResponseJob wird dann wieder dem ThreadPool übergeben. Beim Aufruf des ResponseJobs durch einen ThreadWorker, baut dieser anhand des messageTypes und dem Objekt, welches er sich vom Response holt, die responseMessage zusammen. Diese wird dann an den Publisher übergeben: sendMessage(responseMessage), welcher die Message dann abschickt.

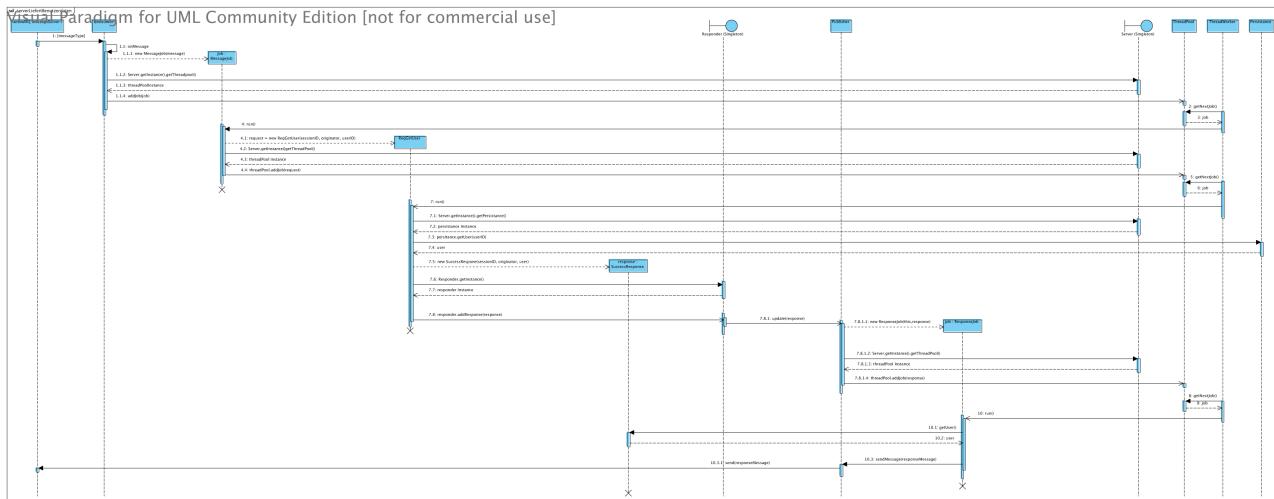
## 1.1 Login



Bei einem Login-Request wird eine Instanz der Persistenz geholt und es werden mit loginSuccess(username,password) die Daten überprüft. Sollte die Login-Anfrage erfolgreich verlaufen, so wird über den SessionManager im Server eine neue Session erstellt. Jetzt kann eine LoginResponse erzeugt werden und dem Responder hinzugefügt werden. Sollte die Login-Anfrage scheitern, so wird eine ErrorResponse erzeugt. Da keine

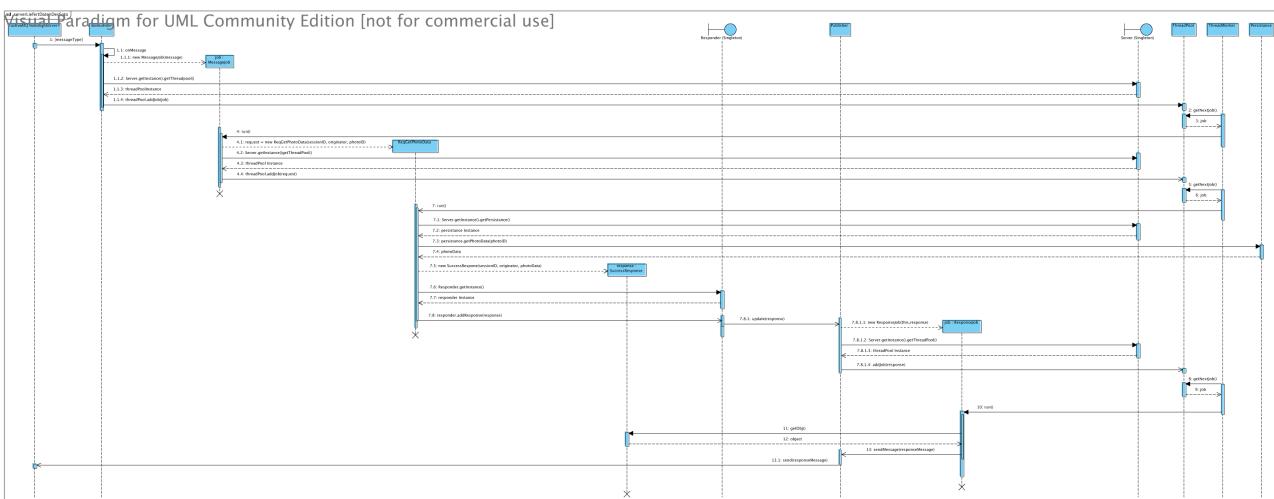
SessionID ermittelt werden konnte wird eine o als SessionID zurückgegeben. Diese ErrorResponse kann nun dem Responder hinzugefügt werden.

## 1.2 Benutzerdaten anfragen



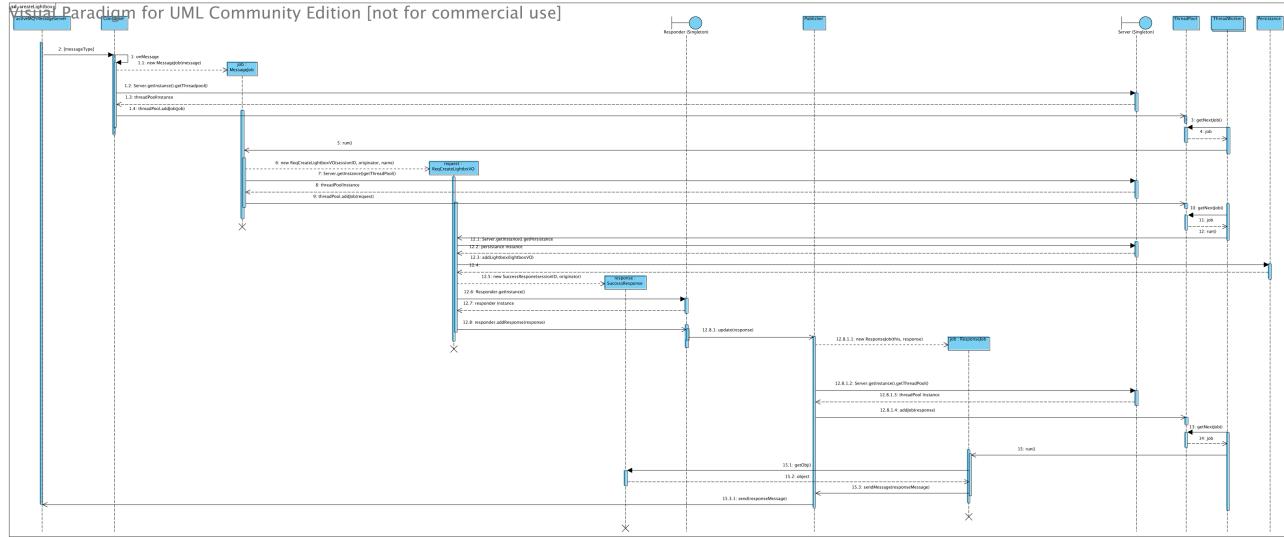
Bei einem GetUser-Request wird eine Instanz der Persistenz geholt und es wird über die userID die Nutzerdaten angefragt. Verläuft dies erfolgreich wird eine SuccessResponse mit einem user-Objekt erzeugt. Diese wird dem Responder hinzugefügt.

## **1.3 Foto-Daten anfragen**



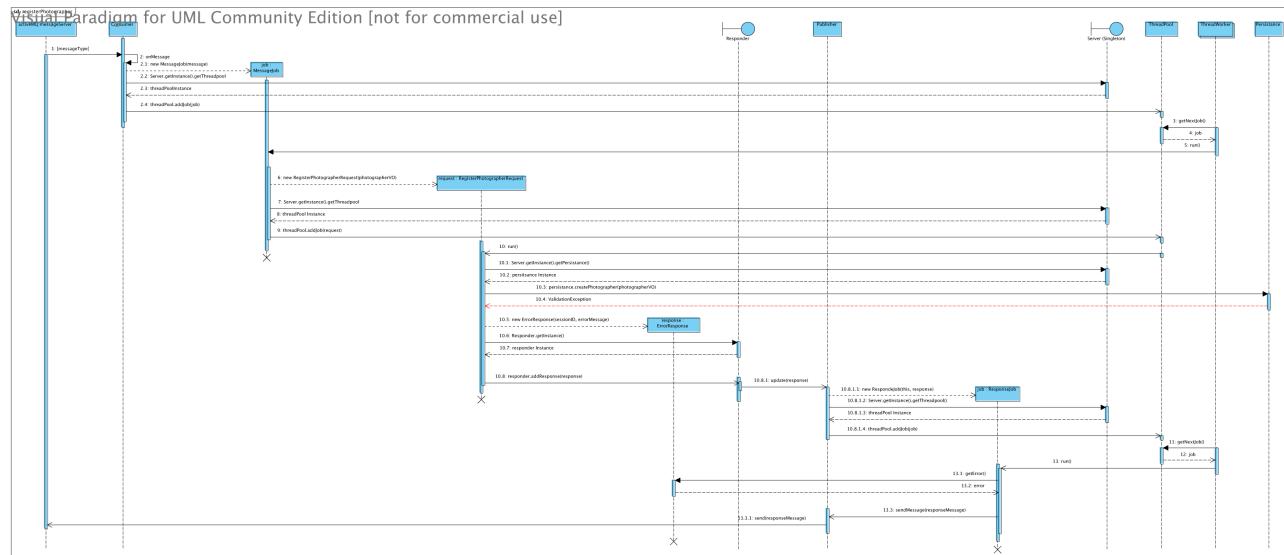
Bei einem GetPhotoData-Request wird eine Instanz der Persistenz geholt und es wird über die photoID die Foto-Daten angefragt. Verläuft dies erfolgreich wird eine SuccessResponse mit einem photoData-Objekt erzeugt. Diese wird dem Responder hinzugefügt.

## 1.4 Lightbox erstellen



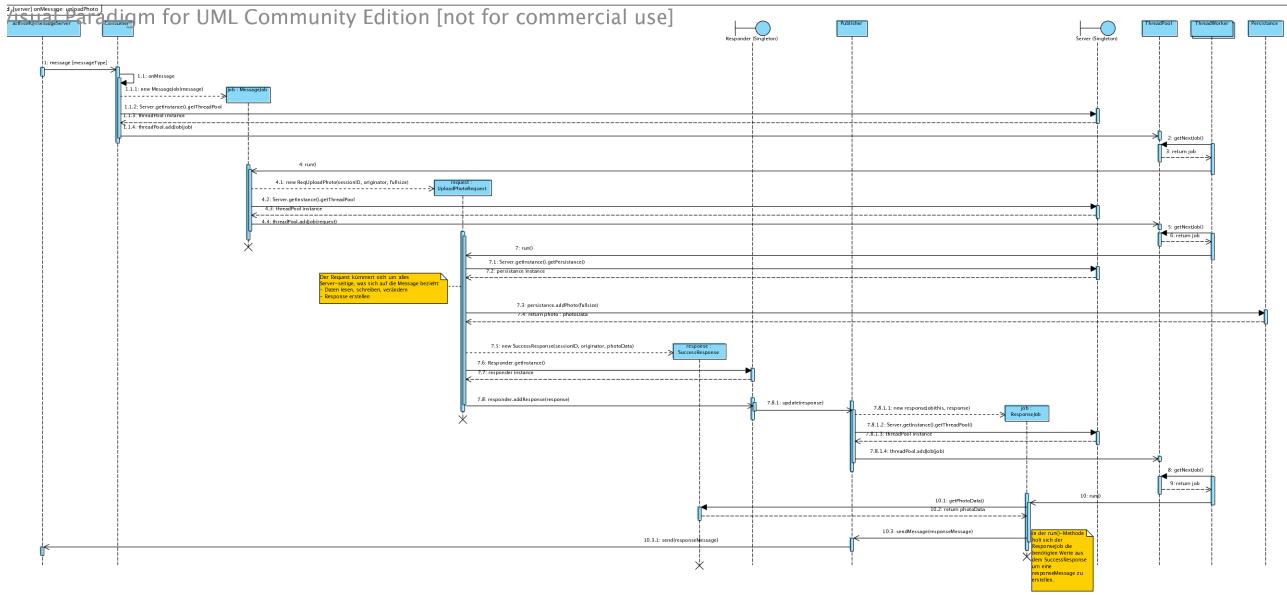
Bei einem CreateLightbox-Request wird eine Instanz der Persistenz geholt und es wird mit dem lightboxVO über addLightbox() eine Lightbox hinzugefügt. Verläuft dies erfolgreich wird eine SuccessResponse erzeugt. Diese wird dem Responder hinzugefügt.

## 1.5 Fotograf registriert sich



Bei einem RegisterPhotographer-Request wird eine Instanz der Persistenz geholt und es wird mit dem photographerVO über createPhotographer() ein Fotograf hinzugefügt. In diesem Sequenzdiagramm wird beim erzeugen des Fotografen ein Fehler in der Datenangabe festgestellt (z.B. Nutzernname existiert schon). Es wird eine ValidationException geworfen, die im Request weiter verarbeitet wird. Es wird eine ErrorResponse mit gefangener Fehlermeldung erzeugt und dem Responder hinzugefügt.

# 1.6 Hochladen eines Bildes

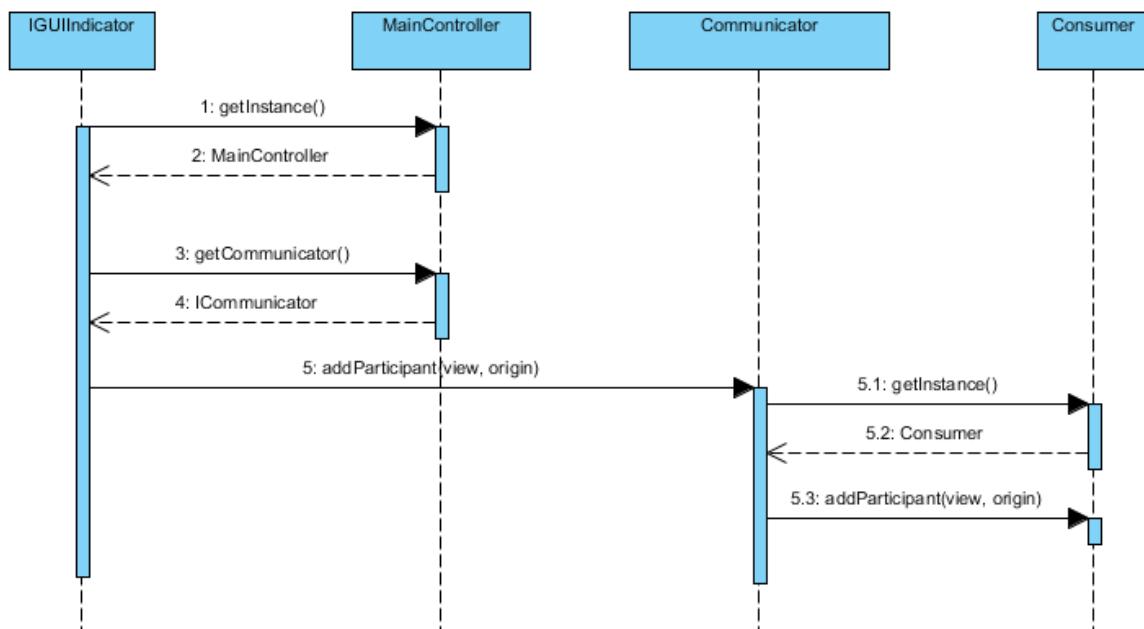


Bei einem UploadPhoto-Request wird eine Instanz der Persistenz geholt und es wird mit dem photo über addPhoto() ein neues PhotoVO erstellt. Die Methode addPhoto() gibt ein Photo-Data zurück. Verläuft dies erfolgreich wird eine SuccessResponse mit der photo-Data erzeugt. Diese wird dem Responder hinzugefügt.

## 2 Client

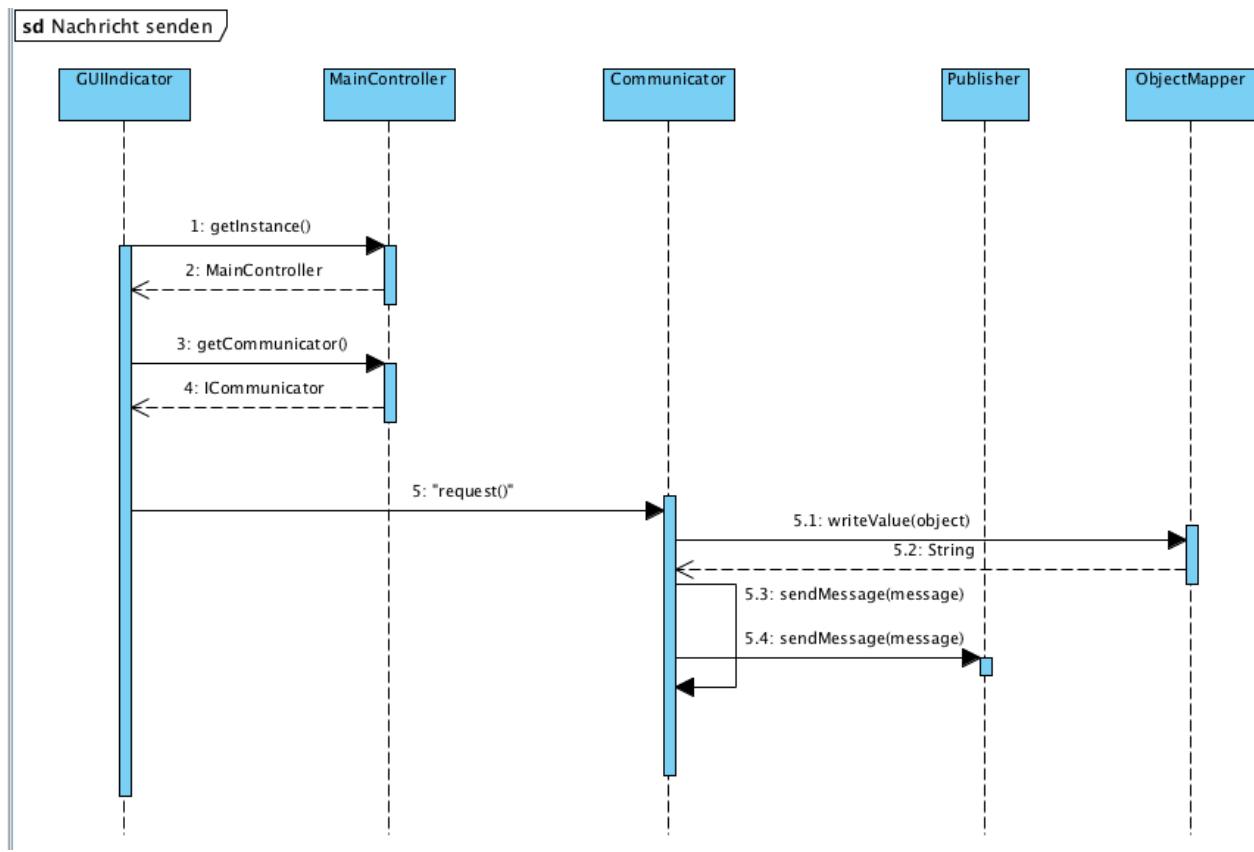
Im folgenden werden drei Sequenzdiagramme gezeigt, welche den Ablauf von der User-Eingabe / Anfrage, bis zur Weiterleitung der Eingabe / Anfrage an den Server beschreiben. Es wurde bewusst kein explizites Beispiel gewählt, da der Ablauf bei nahezu jeder Anfrage der Selbe ist. Der Hauptsächliche Unterschied besteht in den verschiedenen Requests, welche je nach Anwendungsfall variieren. Dieses wird allerdings im folgenden noch näher erklärt.

### 2.1 View am Consumer als Kommunikationsteilnehmer anmelden



Ein Element der View meldet sich als Kommunikationsteilnehmer an, indem es sich eine Instanz von der Singleton Klasse MainController erstellt. Durch den MainController bekommt das Element einen Communicator und meldet sich über diesen als Teilnehmer beim Consumer an. Die Anmeldung muss vor dem Versenden der Anfrage an den Server stattfinden, da sonst der Fall eintreten könnte, dass die Antwort vom Server auf die Anfrage bereits vor der Anmeldung des View - Elements erfolgen könnte. Dadurch würde die Antwort nicht an das GUI-Element weitergeleitet werden können, da es sich nicht in der entsprechenden participants-Map des Consumers befindet.

## 2.1.1 Nachricht senden



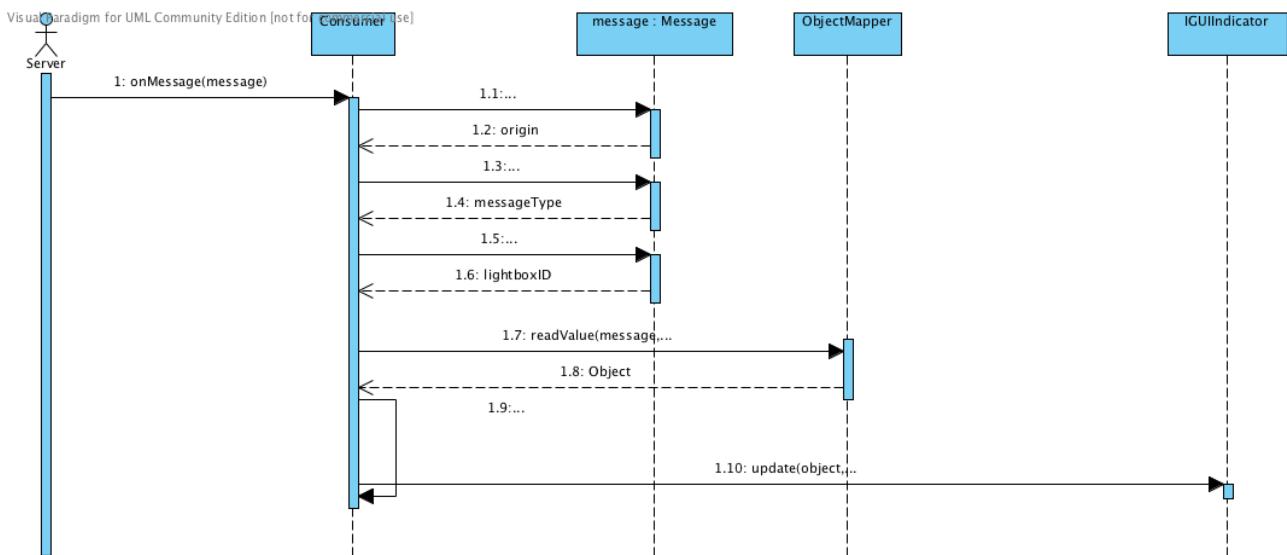
Voraussetzung: Das Senden der Nachrichten setzt voraus, dass sich das jeweilige View-Element (GUIIndicator), über den Communicator bei dem Consumer als Partizipant angemeldet hat (siehe Sequenzdiagramm *View am Consumer als Kommunikationsteilnehmer anmelden*).

Ablauf: Der GUIIndicator holt sich zunächst eine Instanz vom Main-Controller (Singleton), um danach über diese auf den Communicator zugreifen zu können. Wie in allen anderen Fällen auch, dient der Main-Controller als Vermittler zwischen der Präsentationsschicht und der Kommunikationsschicht.

Abhängig von einer aus der GUI-initiierten Aktion wird ein entsprechender \*Request vom Communicator aufgerufen. Der Communicator reicht den Request, welcher die benötigten Parameter (Beispielsweise würden für den Login Username und Password übergeben werden), in den ObjectMapper hinein, welcher einen JSON-String daraus baut und diesen an den Communicator zurückgibt. Über die sendMessage-Methode wird der JSON-String in eine Nachricht verpackt und an den Publisher durchgereicht. Der Publisher ermöglicht als einzige Klasse die Kommunikation zum Server und ist auch für das versenden der Nachrichten an den Server zuständig.

\*Request: Die Art des Requests unterscheidet sich natürlich je nach Anwendungsfall, jedoch ist die Art und vor allem der Weg der Kommunikation in den Meisten Fällen gleich. Die Liste der Requests ist dem Klassendiagramm zu entnehmen.

## 2.1.2 Nachricht empfangen



Der Empfang von Nachrichten wird initial über die “onMessage” Methode des Consumers abgewickelt. Der Consumer liest die “origin” aus, um eine Zuordnung zwischen Nachricht und den einzelnen GUI-Elementen zu erhalten. Enthält eine Nachricht keinen “origin” so wird die Lightbox\_ID aus der Nachricht mit allen in der participants-Map aus der Consumer Klasse gepeicherten abgeglichen. Jede Klasse die sich als Participant anmeldet, muss daher die ICollaborativ Schnittstelle implementieren. ICollaborativ stellt sicher, dass die Klasse eine Methode getLightBoxID() besitzt. Dadurch wird sichergestellt, dass Antworten abgearbeitet werden können, die nicht angefragt wurden, wie beispielweise bei Chatnachrichten. Im Allgemeinen wird die kollaborative Funktion des Client durch dieses Prozedere ermöglicht.

# Visual Paradigm for UML Community Edition [not for commercial use]

