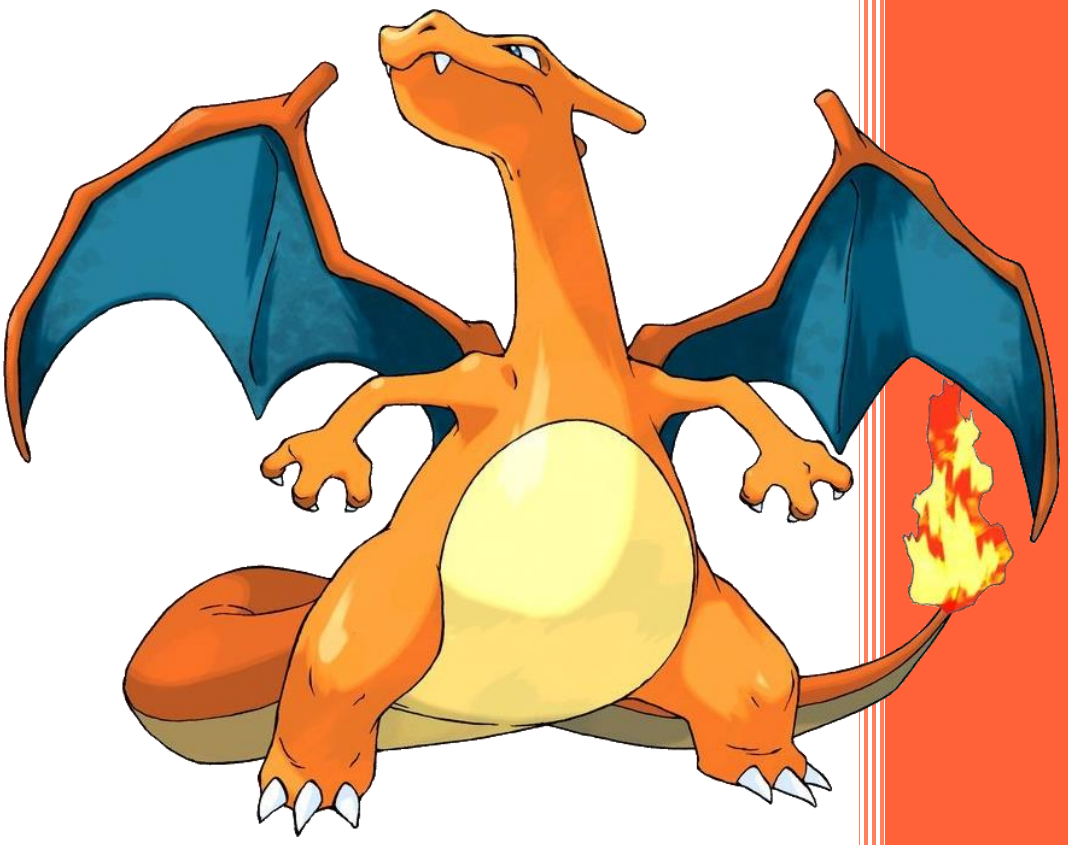
2014

Entwurfsdokument Glurak



Jonas Stadler

Gruppe 4B

5.3.2014

Inhaltsverzeichnis

1. Einleitung

Das hier vorgestellte System hat den Zweck eine Social-Media-Plattform zur Verfügung zu stellen. Auf dieser Plattform soll es dem Benutzer ermöglicht werden Musik von Künstlern zu hören und seine Lieblingsmusik in Wiedergabelisten zu speichern. Um das System zu nutzen erstellt sich jeder Benutzer ein eigenes Profil. Profile von anderen Benutzern können im System angeschaut werden. Dass System stellt eine Follow-Funktion zur Verfügung, über welche der Benutzer über neue Beiträge, wie z.B. neue Ankündigungen, neue Wiedergabelisten oder Alben von Benutzern, Künstlern oder Labels, informiert wird.

Wenn ein Benutzer Musik hochladen will, muss dieser erst zum Künstler ernannt werden. Dazu stellt das System eine Funktion zur Verfügung. Ist er nun Künstler kann er Musik veröffentlichen und für sich werben. Die hochgeladene Musik wird nun im Profil des Künstlers angezeigt. Über die im System vorhandene Datenbank hat jeder Benutzer nun die Möglichkeit sich die Musik anzuhören.

Zusätzlich stellt das System noch Labels zur Verfügung. Diese haben ebenfalls eine eigene Profilseite, auf welcher die im Label zusammengeschlossenen Künstler aufgelistet werden. Ebenfalls werden auf der Labelseite auch die Playlists des Labels angezeigt. Die Label-Profil-Seiten werden von Label-Managern bearbeitet. Nach dem einloggen stellt das System diesen die Funktionen zur Verfügung, welche für das Bearbeiten von Playlisten und Künstlern benötigt werden.

Der Label-Manager bekommt aber nicht nur die Funktionen zum Bearbeiten der Label-Profil-Seiten, er kann auch im Namen eines Künstlers Musik für diesen ins System uploaden und dessen bereits in der Datenbank vorhandenen Musikstücke bearbeiten.

Alle im System vorhandenen Daten werden lokal gespeichert und über eine Datenbank verwaltet.

2. Systementwurf

2.1 Entwurfsziele

Verlässlichkeitskriterien	
Robustheit	Das System soll nach Möglichkeit Abstürze vermeiden. Sollte es doch zu einem Absturz kommen, ist ein Verlust der temporären Daten akzeptabel, bereits gespeicherte Daten sollen aber erhalten bleiben.
Zuverlässigkeit	Es ist ein hohes Maß an Übereinstimmung zwischen erwartetem und beobachtetem Verhalten ge-wünscht.
Verfügbarkeit	Das System soll ausschließlich im Normalbetrieb laufen.
Fehlertoleranz	siehe *Robustheit*
Schutz vor feindlichen Angriffen (security)	Angabe von Benutzername und Passwort ist erforderlich, darüberhinaus im Rahmen des Softwarepraktikums nicht weiter berücksichtigt, da es sich um eine Einzelplatzanwendung ohne Einbindung in öffentliche Netwerke handelt.

Sicherheit (safety)	Es ist in keiner Weise eine Gefährdung menschlichen Lebens durch Funktion oder Fehlfunktion der vorliegenden Software zu erwarten.

Wartungskriterien	
Erweiterbarkeit	Das Hinzufügen neuer Funktionalitäten zum fertigen System ist im Rahmen des Softwarepraktikums nicht beabsichtigt, die Möglichkeit ist prinzipiell aber gegeben.
Modifizierbarkeit	Durch Anwendung des Model-View-Controller Musters wird ein Änderung bzw. Korrektur der Funktionalität erleichtert.
Anpassungsfähigkeit	siehe *Erweiterbarkeit*
Portierbarkeit	Eine Übertragung des Systems auf eine andere Plattform ist nicht vorgesehen.
Lesbarkeit	Durch Anwendung des Model-View-Controller Musters und Verwendung intuitiver Paket-, Klassen-, Attribut- und Methodennamen wird das System durch Lesen des Codes bereits in groben Zügen verständlich.
Rückverfolgbarkeit	siehe *Lesbarkeit*

Leistungskriterien	
Antwortzeit	Die Antwortzeit soll möglichst klein ausfallen, eine besonders hohe Geschwindigkeit ist jedoch nicht gefordert.
Durchsatz	Da es sich hierbei um eine Einzelplatzanwendung in kleinem Rahmen handelt, gibt es hierbei keine speziellen Anforderungen.
Speicherbedarf	Im Rahmen des Softwarepraktikums werden keine besonderen Anforderungen an den Speicherbedarf gestellt.

Kostenkriterien
Entfallen im Softwarepraktikum

Endbenutzerkriterien	
Nützlichkeit	Der Benutzer soll den vollen Funktionsumfang des Systems nutzen können.
Nutzbarkeit	Die Bedienung des Systems soll möglichst intuitiv sein. Es wird lediglich vorausgesetzt, dass der Benutzer über Grundkenntnisse in der Softwareanwendung verfügt.

2.2 Paketverteilung

de.glurak.data:
Datapaket enthält alle Entitätsklassen. Diese erfüllen die Model-Funktion für die übrigen Klassen.

de.glurak.database:
Database-Paket kümmert sich darum, dass die Entitätsklassen persistent in einer Datenbank verwaltet werden können.

de.glurak.feature:
Das Feature-Paket beinhaltet Klassen, die sich um einen speziellen Aufgabenbereich kümmern (z. B. Mp3-Player).

de.glurak.frontend:
Das Frontend-Paket enthält Klassen, die sich um die Interaktion mit dem Anwender mittels Swing von Java kümmert und Daten entsprechend aufbereitet.

2.3 Verwendung existierender Softwarekomponenten

Hibernate/ JPA:
Hibernate/JPA kümmert sich sehr transparent darum, dass die Entitätsklassen in eine Datenbank unserer Wahl gespeichert werden können.

Laut Aufgabenstellung müssen die Daten einen Neustart überleben. Da Hibernate empfohlen wurde, wurde es auch benutzt.

JUnit:
JUnit kümmert sich um automatische Tests unserer Anwendung, d.h. wir brauchen nach Änderungen nicht explizit alle Funktionen zu testen, sondern können dies automatisiert ausführen und Fehler somit aufspüren.

JUnit ist das Standart-Tool und wird bereits von vielen IDE's unterstützt.

Mp3-Player (Library: javazoom):

HSQLDB:
HSQLDB ist eine Datenbank, die gänzlich in Java implementiert worden ist.

Da es in Java geschrieben ist und auch die Speicherung im Dateisystem unterstützt, ist es sehr komfortabel in Java zu benutzen.

2.4 Management persistenter Daten

- ~ Persistente Daten in unserem System sind folgende:
- ~ Bilder (für: Profilbilder (Benutzer, Künstler, Label), Albumbilder)
- ~ Musikstück (Titel, Künstler, Genre, weitere Metainformationen, Musikdatei)
- ~ Alben/Playlisten (Name, weitere Metainformationen, Liste der zugehörigen Musikstücke)
- ~ Userdaten (Username, Passwort, Vorname, Nachname, E-Mailadresse, Geburtsdatum, Herkunftsland)
- ~ Genre (Name, übergeordnetes Genre)
- ~ Nachricht (Absender, Empfänger, Nachrichtentext)

- ~ Ankündigung (Künstler, Text)

Persistenzmechanismus:

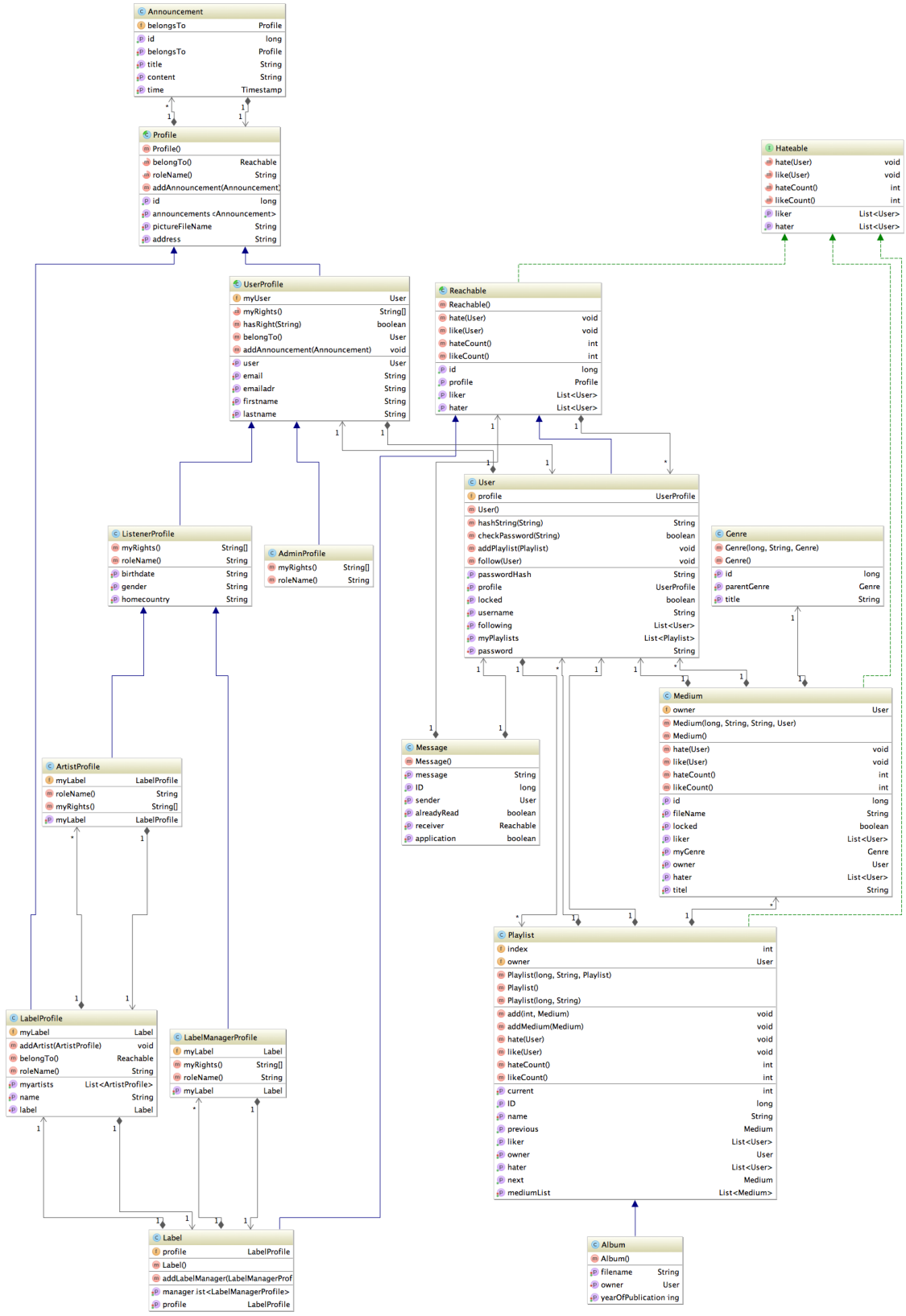
- ~ Bilder und Musikstücke werden lokal gespeichert.
- ~ Alle anderen Daten, sowie die Dateipfade von den Bildern bzw. Musikstücken werden in einer Hibernate Datenbank persistent gehalten.

3. Objektentwurf

3.1. Abwägungen des Objektentwurfs

Textuelle Beschreibung der Überlegungen, die zur Entscheidung für den vorliegenden Objektentwurf geführt haben

3.2. Klassenmodell der Entitätsklassen



3.3. Dokumentation weiterer interessanter Teile des Entwurfsklassenmodells

Verwendete Entwurfsmuster:

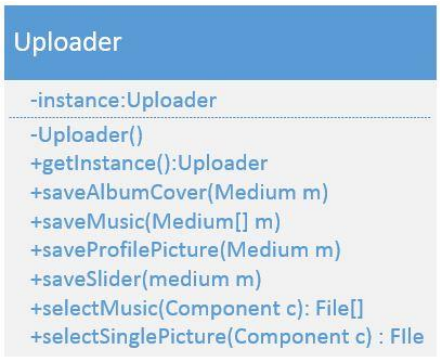


Figure 1 Muster Singleton verwendet in der Klasse "Uploader"

1. Singleton:
   Im obig zu sehenden UML-Klassendiagramm wurde das Singleton Entwurfsmuster verwendet. Das Singleton-Muster wird für den Upload-Vorgang genutzt. Wir benutzen das Singleton-Muster, weil es dafür sorgt, dass von einer Klasse nur genau ein Objekt existiert. Dies ist besonders im Upload wichtig, da große Dateien wie Bilder und Musikstücke auch nur einmal gespeichert werden sollen.
2. Beobachter:
   Das voranstehende UML-Klassendiagramm zeigt das Entwurfsmuster Beobachter. Das Beobachter-Muster gehört zu den Verhaltensmustern und dient dazu, Änderungen an Objekten weiterzugeben. Dies ist wichtig, weil sich häufig Daten ändern können und man nicht mit den alten Daten weiterarbeiten möchte.
3. Kompositum:
   Der Ausschnitt des UML-Klassendiagramms oben ist ein Beispiel für das Entwurfsmuster Kompositum. In unserem Fall wurde das Entwurfsmuster Kompositum für die Genres verwendet. Das Es gibt in unserem Fall ein Wurzelknoten (Standard-Genre) und von diesen zweigen alle weiteren Untergenres ab. Jedes Untergenre hat also einen Mutterknoten, der wiederum einen Mutterknoten hat usw., bis man letztendlich beim Standard-Genre ankommt.

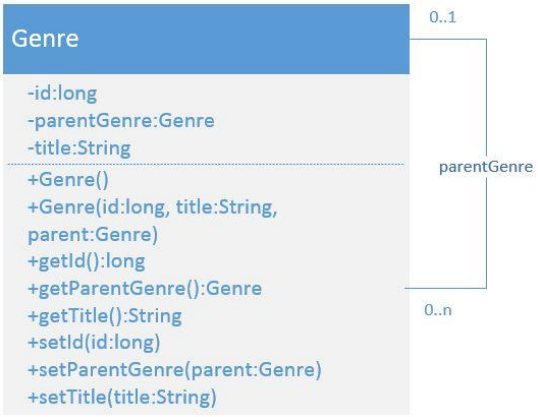


Figure 2Muster Kompositum verwendet in der Klasse “Genre”

4. Glossar

~ User:
- User ist Oberklasse von allen Benutzer. Und besitzt die Grundfunktion (Suchen/Kommunizieren/Musik zum Abspiel verwalten).

~ Listener:
- Listener repräsentiert einen typischen Benutzer. Listener ist ein User. Er wird als einzige Auswahl bei einer Registrierung erstellt. Er kann auch Playlisten erstellen.

~ Artist

- Artist stellt ein Künstler dar. Im System ist er eine Rechteerweiterung des Listener in der Hinsicht, dass er Musik hoch lädt. Außerdem kann er zu einem Label gehören. Er kann auch Genres erstellen.

~ Label-Manager
- Ein Label-Manager stellt ein Label-Manager da!!!! Er kann mehrere Artisten verwalten und in deren Namen Medien verwalten. Diese müssen sich zuvor bei ihm Bewerben, bzw. der Label-Manager bei den Artisten. Zudem erbt er die Rechte vom Artist.

~ The Label
- Dies ist kein eigenständiges Profil/Listener. Er wird von einen bzw. mehreren Label-Manager verwaltet. The Label ist eine Seite die das Label/mehrere Artist repräsentiert.

~ Admin
- Admin ist der Verwalter des Systems. Er kann alles machen, was der Listener kann. Er kann außerdem Medien und Nutzer (en-)sperren. Außerdem kann er Artistanträge annehmen/ablehnen, welcher ein Listener zum Artist macht. Er erstellt die Initial-Genres.

~ Medium:
- Die Musik, die im Programm abgespielt.

~ Playlist:
- Ansammlung von Musikstücken (kann leer sein)

~ Album:
- Erweiterung von Playlist um Metadaten. Kann nur von Artist/Label-Manager erstellt werden.

~ Upload
- Eine Audio-Datei wird von einem Benutzer hochgeladen und im System hinterlegt.

5. Anhang

Javadoc des Codes inkl. Spezifikation nicht-trivialer Verträge und Invarianten (textuell; Formulierung in OCL nicht notwendig); Anhang nur elektronisch abgeben