2014

Entwurfsdokument HateTunes

Gruppe 4B

Entscheider - Robin Rexeisen

Dadomadi - Darko Dermadi

Zian92 - Jonas Stadler

Zengo - Daniel Papoutzis

Christopher Distelkämper

Cimoe - Simon Wolter

Marco Mühlenjost

MxBox - Maxim Balaganskij

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. Das 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 vorhandene Musikstücke bearbeiten.

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

2. Systementwurf

2.1 Entwurfsziele

Die von uns in jeder Kategorie priorisierten Kriterien sind fett geschrieben.

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 gewünscht.**
Verfügbarkeit	Das System soll ausschließlich im Normalbetrieb laufen.
Fehlertoleranz	siehe *Robustheit*
Schutz vor feindlichen Angriffen (security)	Angabe von Benutzername und Passwort ist erforderlich, darüber hinaus im Rahmen des Softwarepraktikums nicht weiter berücksichtigt, da es sich um eine Einzelplatzanwendung ohne Einbindung in öffentliche Netzwerke 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	Das System soll auf Linux, Mac und Windows lauffähig sein.
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	**Es werden keine besonderen Anforderungen bzgl. der Antwortzeit gestellt, solange diese die Benutzerfreundlichkeit bzw. die Funktionalität nicht wesentlich einschränkt. Lediglich der Startvorgang der Anwendung nimmt eine gewisse Zeit in Anspruch, diese wird durch den Splashscreen überbrückt.**
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 eine positive Veränderung im Gebrauch von Social-Media-Plattformen erhalten.
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

2.2.1 Paketbeschreibung

de.glurak.data:

Data-Paket enthält alle Entitätsklassen. Diese erfüllen die Modell-Funktion für die übrigen Klassen.

de.glurak.database:

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

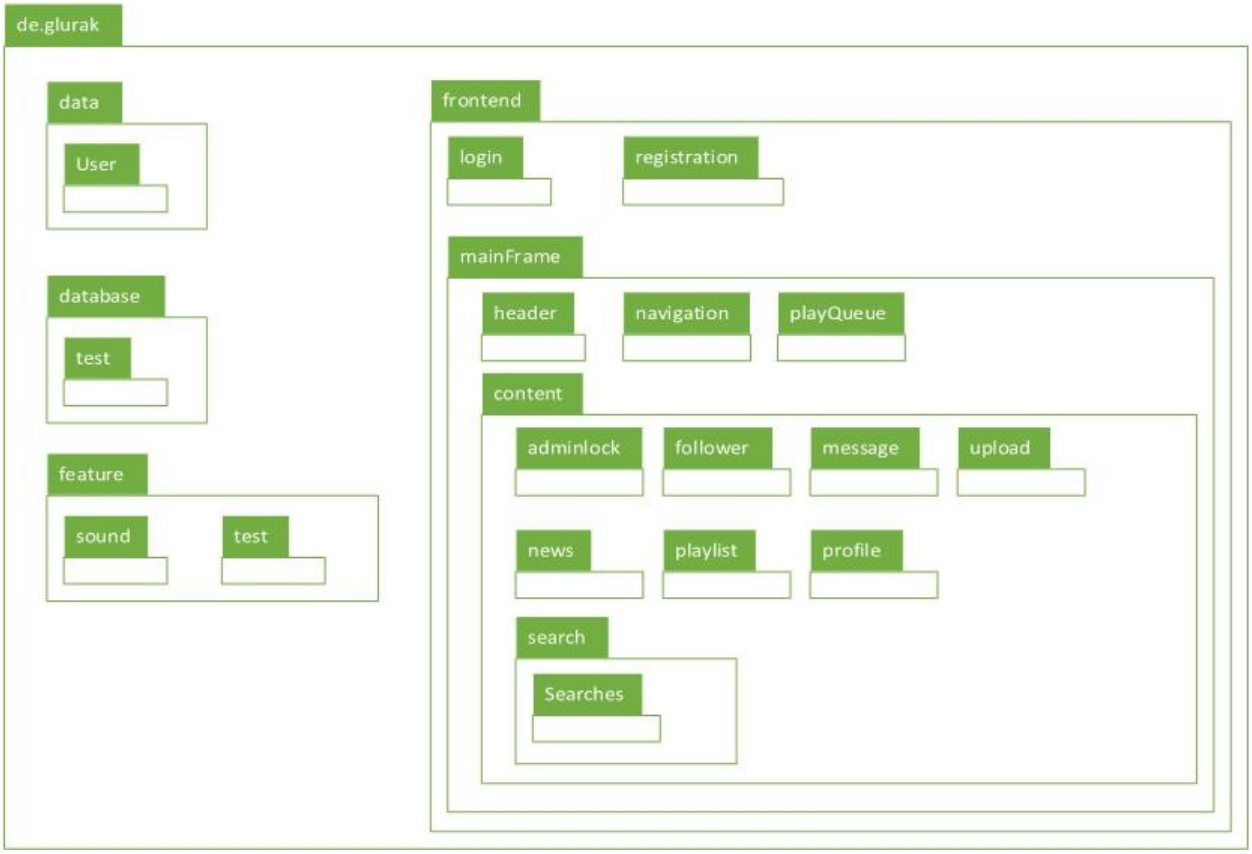
de.glurak.feature:

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.

2.2.2 Paketdiagramm



2.3 Verwendung existierender Softwarekomponenten

2.3.1 Externe 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):

JavaZoom ist eine library welche MP3 Dateien in Echtzeit dekodiert, konvertiert und abspielt.

Mp3plugin kann aus Mp3 Dateien Eigenschaften wie Bitrate, Länge und Samplerate auslesen.

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.

2.3.2 Interne Softwarekomponenten

Splash-Screen und Loginframe

Der Splash-Screen taucht in der Zeit auf, in welcher die Datenbank vom Programm geladen wird. Er verdeutlicht dem Nutzer, dass das System aktiv ist und verkürzt die Wartezeit. Nachdem die Datenbank geladen wurde verschwindet der Splash-Screen und der Login-Screen wird geladen.

Der Login ermöglicht es dem Nutzer sich im System anzumelden oder zu registrieren. Loggt sich ein Nutzer erfolgreich ein, wird das Hauptprogramm geladen und der Nutzer kann dies in vollem Umfang nutzen.

Package *mainFrame*

Das mainFrame Package ist für das Kontrollieren der Hauptansicht verantwortlich.

Es besteht aus vier Klassen die im Folgenden beschrieben werden.

- ~ Der MainFrameVController enthält die Logik um die einzelnen Komponenten der MainFrameView zu erstellen und mit den nötigen Daten zu versorgen. Er initialisiert die MainFrameView, HeaderVController, NavigationVController, ContentController und PlayQueueViewController. Mithilfe des Observer/Observable Patterns fängt er Befehle zum ändern des Hauptbereiches ab und lädt die View des entsprechenden ContentController in den Hauptbereich.
- ~ Der HeaderVController ist für die obere Leiste verantwortlich, welche die Suche, Logoutbutton und das Logo enthält.
- ~ Der NavigationVController ist für die Navigationsleiste links verantwortlich. Er enthält die Logik für die Navigationsbuttons, welche den neuen Content für den ContentController bestimmen. Wenn ein Navigationselement ein Event hervorruft gibt er den dadurch bestimmten Content an den MainFrameVController, mithilfe des Observer/Observable Patterns weiter.
- ~ Der ContentController ist ein Interface welches alle Controller implementieren, dessen Views im Hauptbereich Bereich angezeigt werden.
  Ein Beispiel für einen ContentController ist der ProfileVController. Dieser stellt alle nötigen Daten und View für das Anzeigen von Profilen zu Verfügung.
- ~ Der PlayQueueViewController ist für den Musikplayer verantwortlich.
  Er lädt den Player und versorgt ihn mit der Playlist die abgespielt werden soll und organisiert Änderungen an der Playlist im Player.

Die Klasse *Rights*:

Die Klasse *Rights* deklariert sämtliche Rechte, die ein Userprofil haben kann. Diese können über die Methode *myRights* im korrespondierenden Userprofil abgefragt werden.

Grundsätzlich werden Rechte benötigt, um verschiedene Funktionalitäten nutzen zu können. So kann z. B. ein User, der über ein Listenerprofil verfügt, u.a. Playlists erstellen und Musik hören aber nicht hochladen. Ein User mit Artistprofil hingegen kann auch Musik hochladen.

Das Interface *Hateable*:

Das Interface *Hateable* stellt Methoden zur Bewertung anderer Benutzer (außer dem Admin) sowie Medien bzw. Playlists zur Verfügung. Außerdem kann die Anzahl der positiven/ negativen Bewertungen sowie die Liste der positiven/ negativen Bewerter zurückgegeben werden.

Das Package *database*:

Das Package beinhaltet die Klassen HibernateDB und DBSearch. Erstere stellt grundlegende Funktionen zur Verfügung, um die Entitätsklassen mittels Hibernate in die Datenbank zu speichern bzw. auszulesen. DBSearch stellt Funktionen zur Suche innerhalb der Datenbank nach den Entitätsklassen zur Verfügung.

Die Klasse *Uploader* und das Filesystem:

Die Klasse Uploader ermöglicht es den Usern Dateien, wie z.B. Musik- oder Bilddateien hochzuladen. Dazu wird ein Fenster geöffnet, in dem man die Dateien auf dem System auswählen kann. Die hochgeladenen Dateien werden in einer bestimmten Datei-Struktur gespeichert. Jeder Benutzer hat seinen eigenen Ordner, in welchem die Dateien gespeichert werden.

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)

2.4.1 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.

Begründung für Persistenzmechanismus:

Die Verwendung einer Datenbank bietet einige für die vorliegende Anwendung entscheidende Vorteile gegenüber dem Konzept der Serialisierung. Diese sind vor allem:

- ~ Bessere Skalierung
- ~ Unterstützung komplexer Anfragen an Datenbestand, wie Selektierung aus gespeicherten Medien
- ~ Persistente Datenspeicherung
- ~ Schnelle Reaktionszeit

2.5 Konfiguration/ Installation

Die fertige Software wird als ZIP-Datei übergeben, die eine lauffähige JAR-Datei und die benötigten Ordnerstrukturen enthält. Außerdem enthält das Zip-Archiv eine readme-Datei.

3. Objektentwurf

3.1. Abwägungen des Objektentwurfs

Der vorliegende Systementwurf gliedert das zu erstellende System in kohärente Pakete, die sowohl das Modell der Daten als auch die GUI für den Nutzer und den dazwischen geschalteten Controller enthalten. Über diesen Paketen stehen die Grundelemente des Systems, die Entitätsklassen. Sowohl die Abhängigkeiten dieser Klassen zu den Paketen, als auch die Abhängigkeiten zwischen den Paketen untereinander galt es in der Implementierung aufzulösen.

Jedes Paket im System erhielt daher Zugriff auf die von ihm benötigten Entitätsklassen.

Eine Haupt-Controllerklasse soll im System zwischen den Paketen navigieren. Dieser *MainFrameController* erhält daher eine Instanz vom Typ *ContentContoller,* welcher das Oberinterface aller anderen Controller ist. Wird ein Ereignis ausgelöst, das die Anzeige, und damit die Interaktionsmöglichkeit des Nutzer ändern soll, wird dieses Ereignis von einer Instanz der Kind-Klassen von *ContentContoller* gefangen und an den *MainFrameController* propagiert. Dieser erzeugt nun den angeforderten Controller um damit die nötigen Interaktionsmöglichkeiten zur Verfügung zu stellen.

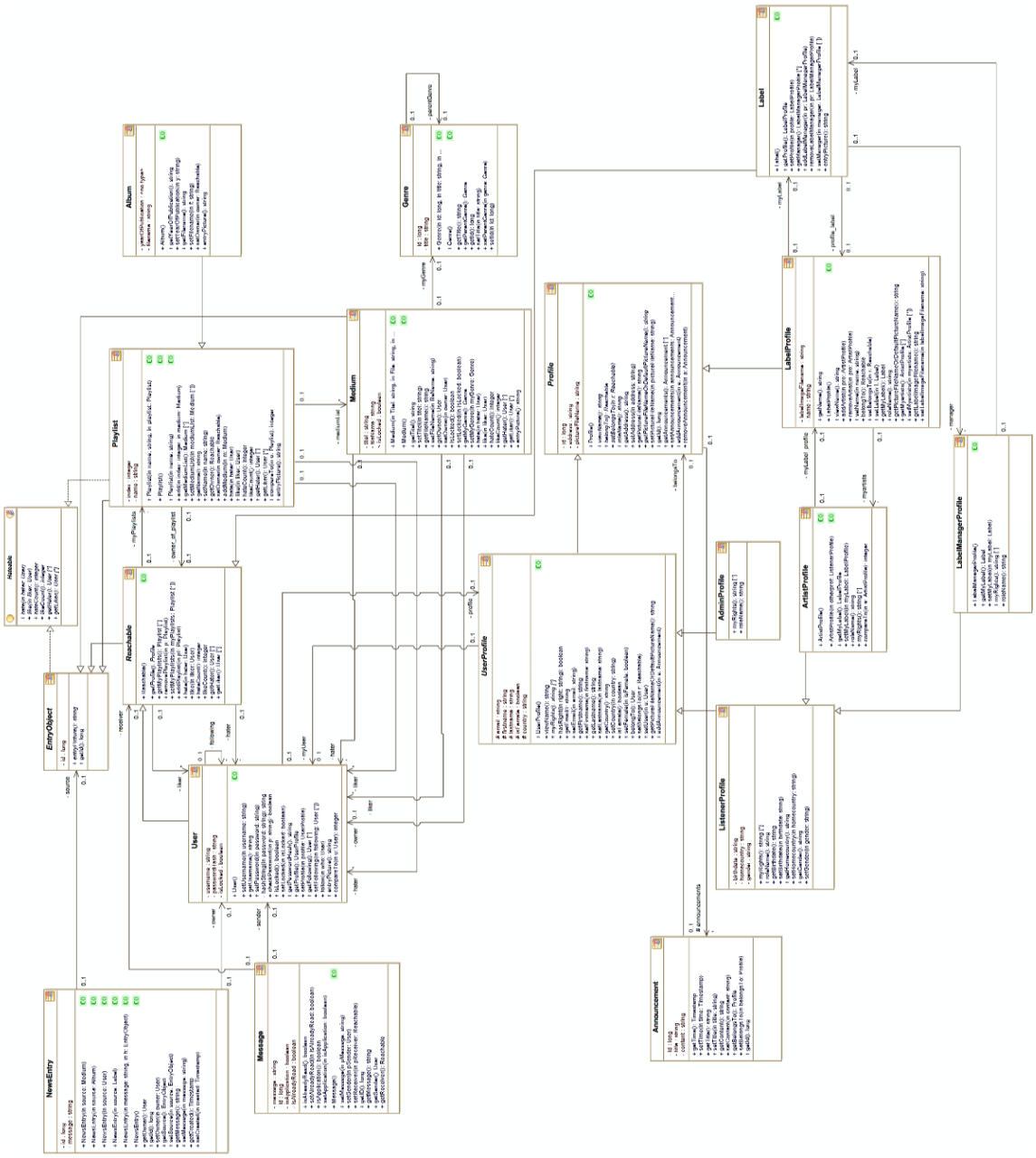
Das Propagieren von Ereignissen wird mithilfe des *Observer/Observable* Musters bewerkstelligt. Dadurch ist sichergestellt, dass jedes Paket sich zunächst selbst um auftretende Ereignisse kümmern muss, was eine stärkere Kohäsion innerhalb von Paketen zur Folge hat.

Die modellierten Entitätsklassen wurden bei der Implementierung überarbeitet. So fasst nun eine Oberklasse *Reachable* Funktionalität zusammen die sich Nutzer und Labels teilen. Dieses beinhaltet in wesentliches das Attribut für das zugehörige Profil und das implementieren des Hateable Interfaces. Dieses Interface ist dazu da um die wesentlichen Methoden zum Liken oder Haten bereitzustellen.

Instanzen der Klasse *User* repräsentieren den Benutzer innerhalb des Systems. Daten des Benutzers werden jedoch nun in Instanzen der Klasse *UserProfile* gesichert. Wird ein Benutzer nun mit den Rechten eines *Artists* ausgestattet, bleibt seine Repräsentation im System gleich, lediglich sein *UserProfil* wird geändert.

3.2. Klassenmodell der Entitätsklassen

Diagramm extern noch einmal als PDF im BSCW im selben Ordner und im **Anhang dieses Dokuments**.



3.3. Dokumentation weiterer interessanter Teile des Entwurfsklassenmodells

Verwendete Entwurfsmuster:

Singleton:

Das in Figur 1 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.

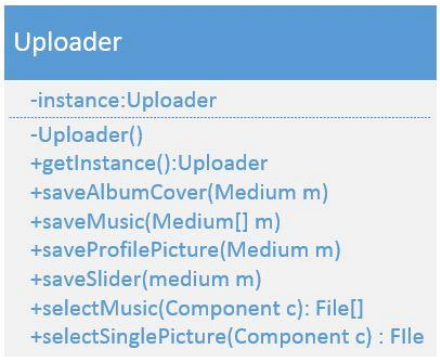


Figure 1 Uploader

Der PlayQueueViewController benutzt ebenfalls das Singleton-Pattern, da dieser aus verschiedensten ControllerKlassen aus angesprochen werden können muss, ohne einen neuen MediaPlayer zu initialisieren, und den aktuellen PlayQueueViewController als Objekt zu kennen. So kann dem aktuellen MediaPlayer von überall aus neue Playlisten oder Medien hinzugefügt werden. Da die Datenbank und SessionThing global verfügbar für jede Klasse seien müssen arbeiten auch diese als Singleton.

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.

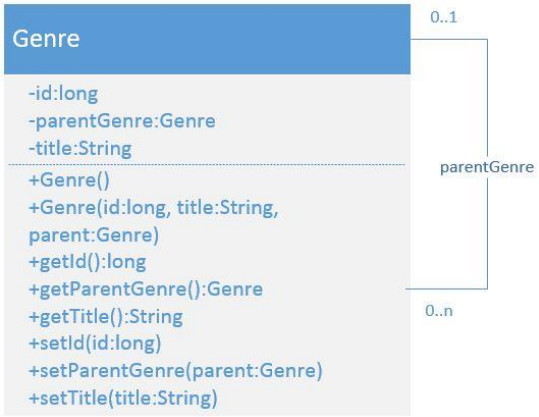


Figure 2 Genre

Entwurfsmuster Observer/Observable

Das Entwurfsmuster Observer/Observable haben wir verwendet um auf Änderungen von Objekten zu reagieren. So implementiert der MainFrameVController das Interface Observer und observiert alle Controller welche für Änderungen in der Hauptview zuständig sind. So erweitert z.B. der NavigationVController das Observable und wird vom MainFrameVController beobachtet. Wenn nun auf der NavigationView ein Button aktiviert wird, gibt der NavigationVController Bescheid, dass er sich geändert hat und alle Beobachter, in diesem Fall der MainFrameVController, rufen ihre Update-Methode auf um auf die entsprechende Änderung zu reagieren.

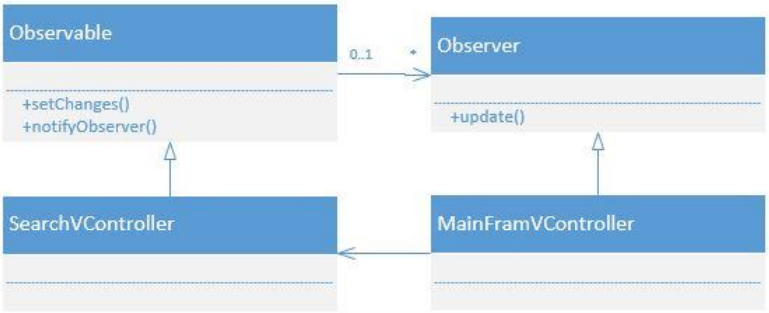


Figure 3 Observer

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

Java-Doc, Benutzerhandbuch, Quellcode und Zip-Datei sind im BSCW hochgeladen.