Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Angewandte Informatik
Systementwicklung und Frameworks

Konzeption und Entwicklung einer CD-Tauschbörse mit Ruby on Rails

Christian Bunk
Alexander Miller
Christian Sandvoß
Antonia Ziegler

Abgabedatum: 08.01.2012

Prof. Dr. Albrecht Fortenbacher

Inhaltsverzeichnis

1. Einleitung

In der Softwaretechnik wird unter dem Begriff Framework ein Ordnungsrahmen verstanden, innerhalb dessen die Entwickler eine Anwendung erstellen. Das Ziel ist die Definition einer einheitlichen Struktur mittels komponentenbasierten Entwicklungsansätzen sowie die Vereinfachung der Entwicklung durch die Verwendung von zur Verfügung stehenden Frameworks-Bestandteilen. Im Laufe der Lehrveranstaltung "Systementwicklung und Frameworks" wurden die Prinzipien der unterschiedlichen Frameworks wie zum Beispiel EJB und .NET vorgestellt und diskutiert. Als Prüfungsleistung muss ein Projekt erfolgreich realisiert werden, wobei jede Gruppe von vier bis fünf Personen die Anwendung mit einem ausgewählten Framework entwickeln müssen.

In der vorliegenden Ausarbeitung handelt es sich um eine Dokumentation zum Projekt, wobei eine CD-Tauschbörse konzipiert und implementiert wurde. Wie in der Aufgabenstellung bzw. im Pflichtenheft definiert (Anhang A.1) wurde eine Rich-Applikation (RIA) modelliert und entwickelt, wobei die meisten Funktionalitäten nicht von Hand, sondern durch die Verwendung von Framework-Komponenten realisiert wurden. Die Gruppe besteht aus vier Personen (Christian Bunk, Alexander Miller, Christian Sandvoß, Antonia Ziegler) und entwickelt das Projekt mit dem Framework "Ruby on Rails".

Ruby on Rails ist ein Framework für Webapplikationen, welches in der Programmiersprache Ruby entwickelt wurde. Model-View-Controller (MVC) Architektur ermöglicht eine Isolation der Businesslogik von der graphischen Benutzeroberfläche und gewährleistet das "don't repeat yourself" (DRY) Prinzip.

Der SourceCode wird mit Git (https://github.com/christianb/SE-FW) verwalten. Hierzu wird die online Plattform GitHub als zentrales Repository verwendet, die sowohl die aktuellen Aufgaben (Issues) als auch die Meilensteine verwaltet. Um eine Arbeitsgrundlage zu erschaffen, haben alle Gruppenmitglieder sich das grundlegende Wissen über das ausgewählte Framework angeeignet. Wie die geforderten Funktionalitäten im Einzelnen implementiert sind, kann den folgenden Kapiteln entnommen werden. An dieser Stelle muss noch ergänzt werden, dass alle gestellten Anforderungen erfüllt wurden.

2. Ruby-on-Rails Framework

Ruby On Rails (Rails) ist ein Web-Framework welches auf der Programmiersprache Ruby basiert. Ruby ist eine Objektorientierte Programmiersprache. Anweisungen werden nicht durch ein Semikolon abgeschlossen sondern durch einen Zeilenumbruch. Jede Rails Anwendung hat eine feste Ordnerstruktur und verfolgt das Prinzip "Convention Over Configuration". Was bedeutet das der Konfigurationsaufwand durch einhalten von Konventionen so gering wie möglich gehalten werden soll. Zusätzlich zu allen erstellten Controllern wird auch ein so genannter Application-Controller erstellt. Alle weiteren Controller sind Unterklassen von diesem und erben daher auch alle Methoden die darin deklariert wurden. Im nächsten Abschnitt sind die wichtigsten Konventionen näher erläutert.

2.1. Rails-Konventionen

Wie oben erwähnt verfolgt Rails das Konzept "Convention Over Configuration". Darunter fallen z.B. die Namenskonventionen welche besagen, dass die Namen der Controller möglichst im Plural benannt werden und Models im Singular. Des Weiteren wird die Schreibweise von Variablen- und Klassennamen geregelt. Durch diese Konventionen weiss Rails welcher Controller zu welchen Model gehört, sowie welche Views zu einem Controller. Dadurch entfällt die explizite Zuweisung zwischen den einzelnen Komponenten. Die Methoden im Controller werden Actions genannt. Zu jeder Action gibt es meist eine View welches Äquivalent zur Bezeichnung der Action benannt ist. Sobald einen View und eine Action den gleichen Namen haben, wird beim Aufruf dieser automatisch die entsprechende View geladen.

2.2. Model-View-Controller (MVC)

Das Rails-Framework ist nach dem Model-View-Controller Konzept aufgebaut. Dabei dient das Model zur Kommunikation mit der Datenbank. Außerdem sorgt es durch Validierung dafür, dass Daten im korrekten Format in die Datenbank geschrieben werden. Die Views sind die HTML-Seiten, welche dem Nutzer angezeigt werden. Sie dienen zur Anzeige der Daten aus der Datenbank oder dem Erfassen von Nutzereingaben mit Hilfe von Formularen bzw.

Eingabefeldern. Der Controller ist der Vermittler zwischen der View und dem Model. Er empfängt Daten von der View und sendet sie an das Model weiter. Umgekehrt werden Daten vom Model über den Controller der View zur Verfügung gestellt.

2.3. Don't Repeat Yourself (DRY)

DRY beschreibt das "Don't Repeat Yourself" Konzept. Dies bedeutet das so wenig wie möglich Code dupliziert wird. Zur Einhaltung dessen gibt es in Rails einigen Vorkehrungen. Als erstes wären da die Helper-Methoden zu nennen. Dies sind Methoden welche oft dazu genutzt werden, häufig genutzten Code durch Schlüsselwörter zu ersetzen. Die Helper-Methoden werden meist in den Views verwendet um dort HTML-Code zu generieren. Ein oft verwendeter Helper ist beispielsweise "link-to", welcher den aus HTML bekannten Link-Tag generiert.

Ein weiterer Punkt ist die Definition des Seitenlayouts. Das Layout legt die Darstellung bzw. das Aussehen der HTML-Seiten fest. Dabei kann ein globales, für alle Seiten gültiges Layout definiert werden oder auch Layout-Vorlagen für spezielle Seiten definiert werden. Das Layout welches mit "application.html.erb" benannt wird, ist für alle Seiten gültig. Falls ein anderes Layout verwendet werden soll, muss diese explizit eingebunden werden. Ansonsten wird immer das in der Datei "application.html.erb" definierte genutzt. Das gleiche gilt auch für den Application-Controller. Hier definierte Methoden sind in allen anderen Controllern verfügbar und müssen dadurch nicht in jedem Controller separat implementiert werden.

Seit der Version 3.1 von Rails, wird standardmäßig jQuery als Javascript Framework verwendet. Auch hier gibt es eine globale Datei (application.js) auf die aus allen Views zugegriffen werden kann. Javascript Code wird in der Regel nicht in den Views eingefügt, sondern es wird auf die zu verändernden HTML Elemente durch dessen IDs oder Klassen zugegriffen. Dies dient den Prinzip des Unobtrusive (unaufdringlich) Javascript und erhöht die Lesbarkeit des Codes der Views. Eine weitere Neuerung der Version 3.1, ist die Asset-Pipeline. Darin werden die CSS und JavaScript Dateien gespeichert. Diese werden dann, vor Auslieferung an den Browser komprimiert, um die zu übertragene Datenmenge zu verringern und dadurch die Zugriffszeit zu erhöhen.

2.4. Object-Relation-Mapper (ORM)

Um bei Rails Objekte in einer Datenbank zu speichern, kommt der Object-Relation-Mapper ActiveRecord zum Einsatz. Dabei werden die Tabellen als Klasse gesehen, die Zeilen als Objekt und die Spalten sind die Objekt-Attribute.

Durch den ORM müssen keine SQL-Statements mehr geschrieben werden, sondern für den Zugriff auf die Datenbank werden Methoden genutzt. Beim erstellen einer Tabelle werden von Rails automatisch Methoden generiert. Eine Klasse hat unter andren die Methoden ”new” und ”find”. Diese dienen zum erstellen oder finden eines Objekts. Das Objekt besitzt die Methoden ”save”, ”update” und ”delete”. Diese werden zum Speichern, Aktualisieren und Löschen genutzt.

3. Anwendung

3.1. Projektmanagement

Eine der wichtigsten Aufgaben bei der Realisierung eines informationstechnischen Projektes ist das Projektmanagement. Es ist äußerst wichtig eine genaue Planung sowie die Konzeption durchzuführen, damit alle Anforderungen, Termine und der Kostenrahmen eingehalten werden. In den nächsten Kapiteln wird es beschrieben wie das Projektmanagement erfolgte, wobei intensiv auf die Meilensteinplanung sowie die Versionsverwaltung und Deployment eingegangen wird.

3.1.1. Meilensteinplanung

Um das Zeitmanagement zu organisieren, wurde die Meilensteinplanung durchgeführt. Es wurde vereinbart, dass bei der Entwicklung der meisten Funktionalitäten immer zwei Gruppenmitglieder beteiligt sein müssen, um bei krankheitsbedingten Ausfällen eines Gruppenmitglieds die Entwicklung fortsetzen zu können.

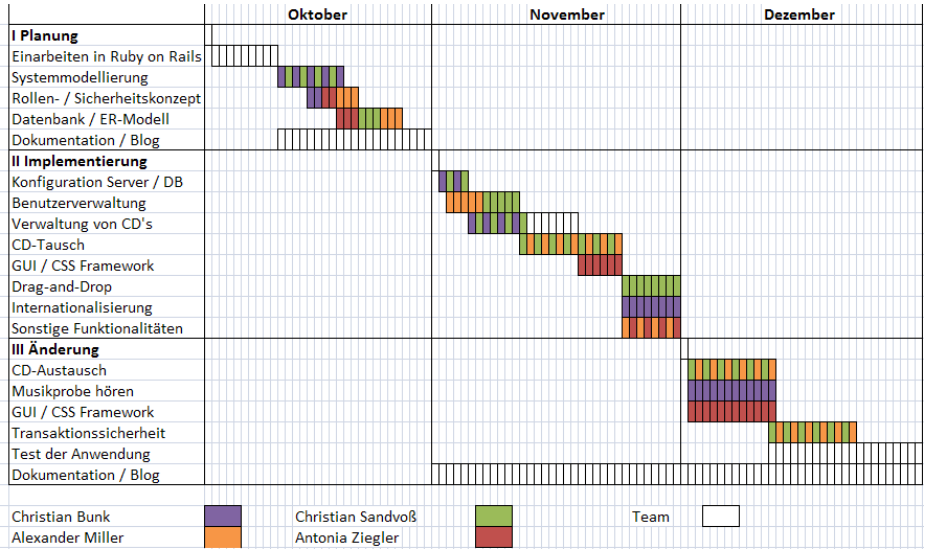

Abbildung 3.1.: Meilensteinplanung

Darüber hinaus wurde es bestrebt, die Konzeption sowie die Systemmodellierung als Gruppenarbeit durchzuführen. Für diese Zwecke wurden wöchentliche Termine (dienstags 12:00-14:00 und donnerstags 11:15 - 12:30) vereinbart. An diesen Terminen wurden in Rahmen einer Gruppenarbeit sowohl die aufgetretenen Schwierigkeiten gelöst, die wöchentliche Präsentation vorbereitet als auch der weitere Projektverlauf diskutiert.

3.1.2. Versionsverwaltung von Quellcode

Für die Quellcode-Verwaltung wurde ein GitHub-Repository verwendet. Dabei wurde an das Branching-Modell orientiert, die in (http://nvie.com/posts/a-successful-git-branching-model/) vorgestellt wird.

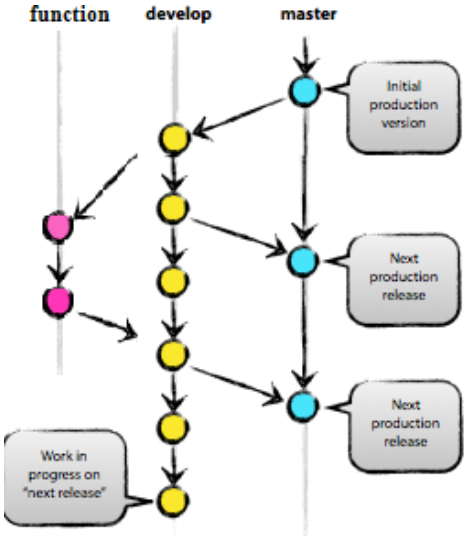

Abbildung 3.2.: Git - Function, Develop und Master Branches

Dabei wird folgende Strategie verfolgt: Als erstes werden zwei Branches angelegt *develop* und *master*. Die eigentliche Entwicklung geschieht auf dem *develop* Branch und lediglich die Zwischenversionen, die eine neue und vor allem fehlerfreie Implementierung eine zusätzlichen Funktionalität beinhalten, werden auf den *master* Branch gelegt. Durch das Erstellen von weiteren Branches für die jeweilige Funktionalität kann weiterer Komfort bei der Entwicklung erreicht werden. Jede zusätzliche Funktionalität wird dabei separat entwickelt, so dass eine Aufteilung der Aufgaben sowie die anschließende Zusammenführung der Quelltexte ohne Einschränkungen und Probleme erfolgen können. Wenn eine Funktion fertig gestellt worden ist, wird diese ins *develop* Branch kopiert. Falls mehrere Funktionalitäten im *develop* Branch getestet und die identifizierten Fehler behoben wurden, wird die Anwendung ins *master* Branch gemergt.

Bei der Einhaltung des vorgestellten Models sieht die Quellcode-Verwaltung wie folgt aus:

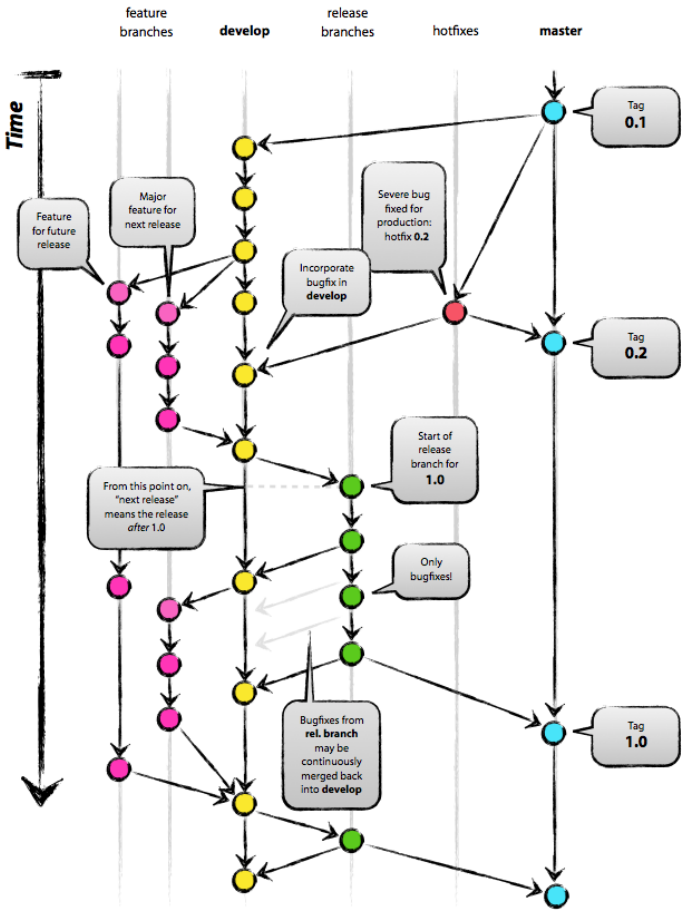

Abbildung 3.3.: Git - Feature und Develop Branches

3.1.3. Deployment

Da die Anwendung auf einem Webserver zur Verfügung gestellt werden soll, wurde dieser eingerichtet[1]. Dafür wurde auf den uns zur Verfügung stehenden Server, als erstes die benötigte Software installiert. Um überhaupt Rails Anwendungen ausführen zu können, wurde Ruby in der Version 1.9.2 und Rails 3.1.1 installiert. Anschließend wurde der Apache Webserver installiert. Dieser wurde entsprechend konfiguriert sowie eine PostGreSQL Datenbank installiert. Desweiteren wurde, Capistrano ein OpenSource Deployment-Tool für Rails-Anwendungen, installiert. Dabei handelt es sich um eine Software für das automatisierte Ausführen von Aufgaben auf einem oder mehreren entfernten

[1] http://kallisto.f4.htw-berlin.de

Servern. Das zentrale Prinzip des Tools ist die vollständige Automatisierung des Verteilungsprozesses. Somit sind die einzelnen Schritte in einem zusammengefasst, wie: Auschecken der Software aus der Versionskontrolle, Ausführen der Unit-Tests, Übertragung der Software auf die Ziel-Server, Aktualisierung der Datenbanken und Neustart des Webservers. Diese Schritte wären einzeln ausgeführt sehr zeitintensiv, fehleranfällig und zu dem auf Dauer langweilig, vor allem da in einem agilen Entwicklungsprozesses dies relativ häufig ausgeführt werden muss.

3.2. File Upload (Paperclip)

In der Webanwendung sollen zwei Arten von Datein hochgeladen werden kÃ¶nnen: Bilder und Audio. Um Dateien zum Server zu laden stehen fÃ¼r Rails zwei Plugins in Rails zur VerfÃ¼gung Attachment-fu und Paperclip[2]. Wir haben uns fÃ¼r Paperclip entschieden das etwas flexibler als Attachment-fu ist. MÃ¶chte man Bilder mit Paperclip zum Server hochladen wird zusÃ¤tzlich noch das Tool ImageMagick benÃ¶tigt, welches in der Lage ist verschiedene Versionen eines Bildes zu erstellen sowie die Meta-Informationen auszulesen. Im Model wird definiert welche Art von Datei hochgeladen werden kann. So sollen in fÃ¼r den User als auch fÃ¼r die CD's Bilder hochgeladen werden kÃ¶nnen. Dazu mÃ¼ssen in der Datenbank zusÃ¤tzliche Spalten angelegt werden. Paperclip speichert die hochgeladenen Daten nicht in der Datenbank sondern auf dem Dateisystem. In der Datenbank wird lediglich die URL zu der Datei gespeichert. Das macht den ganzen Prozess wesentlich performanter, da die Datenbank nicht so vollgestopft wird. Das hochladen von Audio funktioniert fast genauso, nur das man hier auf andere Mime-Types prÃ¼ft. Es hat sich gezeigt dsa verschiedene Browser und auch verschiedene Plattformen unterschiedlich mit den Mime-Types umgehen. Mit Paperclip kann sehr schnell ein Datei Upload eingerichtet werden.

[2]https://github.com/thoughtbot/paperclip

3.3. MusicBrainz

MusicBrainz[3] ist ein Webservice ueber den Daten zu Alben und Kuenstlern abgerufen werden kÃ¶nnen. Die API erlaubt Abfragen. Dabei kann man entweder allgemein suchen dann bekommt man eine Ergebnisse liste die nach einen Score sortiert ist. Man kann aber auch speziell nach einer eindeutigen ID der MBID suchen. Unsere Vorstellung war es MusicBrainz zur Erstellung einer CD zu verwenden. Der Benutzer muss dazu mindestens das Album und den Kuenstler angeben. Diese Daten sind fest und werden von MusicBrainz nicht verÃ¤ndern. Anhand dieser Daten wird versucht den KÃ¼nstler und das Album zu finden. War die Suche erfolgreich wird nach dem Erscheinungsjahr und Tracks geschaut. Diese werden in die Form eingetragen und kÃ¶nnen vom Benutzer noch bearbeitet, ergaenzt oder geloescht werden bevor das Formular abgeschickt wird und die CD erstellt wird. Gleichzeitig wird eine Amazon Standard Identifikation Number (asin) gespeichert welche die CD eindeutig bei Amazon identifiziert. DarÃ¼ber kann eine URL erstellt werden, Ã¼ber welche ein Cover heruntergeladen werden kann.Diese hat die Form http://images.amazon.com/images/P/<ASIN>.jpg, wobei ASIN irgendeine eindeutige Nummer darstellt. Ist ein Cover vorhanden wird das Ã¼ber die URL heruntergeladen, so das der Benutzer selbst kein Cover hochladen muss.

[3] http://musicbrainz.org/

3.4. Tests

Wir haben das Standard Testframework von Rails verwendet um umfangreiche Unit Tests zu schreiben. Dies basiert auf der Idee des Test-Driven-DEvelopment. Also zuerst Tests schrieben, schauen wie sie fehlschlagen und dann die FunktionalitÃ¤t implementieren und prÃ¼fen ob der Test erfolgreich ist. ie Unit Tests sichern unsere GrundfunktionalitÃ¤t. Des weiteren haben wir mit Cucumber Tests geschrieben. Mit Cucumber (Behavior-Driven-Development) ist es mÃ¶glich Tests in Prosaform zu schrieben. Das macht die Tests besser lesbarer auch fÃ¼r nicht Informatiker. Die Tests werden hÃ¤ufig als User Stories geschrieben in der Form: Äls <Rolle>, mÃ¶chte ich <Ziel/Wunsch>, um <Nutzen>". Umfangreiche Tests wie sie in einem realen Projekt gemacht werden sollten konnten wir aufgrund der beschrÃ¤nkten Ressourcen jedoch nicht durchfÃ¼hren. Rails eignet jedoch wunderbar fÃ¼r ein Test-Driven-Development da es ein Standard Testframwork integriert.

4. Zusamenfassung

...

A. Anhang

A.1. Aufgabenstellung

Pflichtenheft

CD-Tauschbörse

Systementwicklung und Frameworks
Wintersemester 2011/12

geändert: 2011-12-01
geändert: 2011-12-15

1 Zielbestimmung

Die Anwendung realisiert eine Tauschbörse für Audio-CDs. Registrierte Mitglieder können Kontaktprofile erstellen, CDs anbieten und mit anderen Mitgliedern tauschen. Gäste und Mitglieder können Profile und CDs ansehen und CDs bewerten. Gäste können sich auch registrieren, um selbst Mitglieder zu werden.

~~Jeder Benutzer kann eine Tauschanfrage auf eine fremde CD stellen. Der Besitzer kann ablehnen oder ein Tauschangebot machen, welches der erste Benutzer annehmen oder ablehnen kann. Tauschanfragen und Bestätigung/Ablehnung werden über einen internen Nachrichtendienst realisiert. Bei erfolgtem Tausch werden die CDs der Sammlung des jeweiligen Benutzers hinzugefügt.~~

Jeder Benutzer kann ein Angebot abgeben, welches mindestens eine Wunsch-CD enthält. Das Angebot kann abgelehnt oder angenommen werden. Alternativ kann auf das Angebot mit einem modifizierten Angebot (Gegenangebot) geantwortet werden.

Die Webanwendung realisiert ein Rollen- und Sicherheitskonzept. Der Tausch von CDs ist als Transaktion zu realisieren. Die Anwendung soll eine Rich Application sein, welche auch verschiedene Medien (Audio) integriert.

2 Produktfunktionen

2.1 Benutzer

/PF01/ Am System registrieren
Akteur Gast[1]

/PF02/ Am System anmelden
Akteur Gast

/PF03/ Eigene Profildaten ändern
Akteur Benutzer[2]
Zusatz nur PD02, PD03, PD04, PD05, PD06, PD7

/PF04/ Eigenes Benutzerkonto löschen
Akteur Benutzer

1 Gast = Ein am System aktuell nicht angemeldeter Benutzer.
2 Benutzer = Ein am System aktuell angemeldeter Benutzer.

/PF05/ Beliebiges Benutzerprofil einsehen
Akteur Benutzer
Zusatz PD02, PD03, PD04, PD06, PD07

/PF06/ Beliebiges Benutzerprofil einsehen
Akteur Administrator
Zusatz nur PD01, PD02, PD03, PD04, PD06, PD07

/PF07/ Beliebiges Benutzerkonto löschen
Akteur Administrator

/PF08/ Benutzer suchen
Akteur Benutzer, Administrator
Zusatz Suche über Teiltexte (Substring)
Zusatz nur PD02, PD03, PD04, PD06

/PF09/ Passwort vergessen?
Akteur Gast
Zusatz neues Passwort wird an Mailadresse verschickt

2.2 CDs

/PF10/ beliebigen Showcase ansehen
Akteur Gast, Benutzer, Administrator

/PF11/ zu jeder CD weitere Infos ansehen
Akteur Gast, Benutzer, Administrator

/PF12/ zu jeder CD Musikprobe hören (falls vorhanden)
Akteur Benutzer

/PF13/ CD in eigenen Showcase hochladen
Akteur Benutzer

/PF14/ CD aus eigenem Showcase entfernen
Akteur Benutzer

/PF15/ nach CDs suchen
Akteur Benutzer
Zusatz Suche über Teiltexte (Substring)
Zusatz nur PD10, PD11, PD13, PD14

2.3 Tauschangebote

~~**/PF16/** Anfrage erzeugen~~
~~**Akteur** Benutzer~~

~~**/PF17/** Anfrage ablehnen~~
~~**Akteur** Benutzer~~

/PF18/ Angebot ~~zu Anfrage~~ erzeugen
Akteur Benutzer

/PF19/ Angebot annehmen
Akteur Benutzer

/PF19/ Angebot ablehnen
Akteur Benutzer

/PF20/ Angebot modifizieren und zurückschicken (Gegenangebot)
Akteur Benutzer

~~**/PF20/** erhaltene Anfragen anschauen~~
~~**Akteur** Benutzer~~

~~**/PF21/** meine Anfragen anschauen~~
~~**Akteur** Benutzer~~

/PF20/ erhaltene Angebote anschauen
Akteur Benutzer

/PF21/ ~~meine~~abgegebene Angebote anschauen
Akteur Benutzer

3 Produktdaten

3.1 Benutzerdaten

/PD02/ E-Mail Adresse

/PD03/ Vorname

/PD04/ Nachname

/PD05/ Passwort
Zusatz verschlüsselt dargestellt

/PD06/ Pseudonym (optional)

/PD07/ Benutzerbild (optional)

/PD08/ Zeitpunkt der Registrierung

3.2 CD-Daten

/PD10/ Titel

/PD11/ Interpret

/PD12/ Cover

/PD13/ Genre (Freitext, optional)

/PD14/ Erscheinungsjahr (optional)

/PD15/ Beschreibung (optional)

~~Anfrage-Daten~~

~~**/PD16/** Wunsch-CD~~

~~anfragender Nutzer~~

~~angefragter Nutzer~~

3.3 Angebots-Daten

/PD2016/ Wunsch-CDs (mindestens eine)

/PD2117/ Tausch-CDs (beliebig)

/PD22/ anfragender Nutzer

/PD23/ angefragter Nutzer

/PD24/ Text

4 sonstige Anforderungen

4.1 Usability

/PS01/ Angebotserstellung über Drag & Drop

/PS02/ Darstellung von CD im Angebot durch Cover

~~**/PS03/** Beschreibung von CD im Angebot durch Tooltip~~

4.2 System

/PS04/ Angebotsworkflow über internes Nachrichtensystem

/PS05/ Transaktionssicherheit

/PS06/ Rollenkonzept mit Authentifizierung