Karlsruhe Institute of Technology

Praktikum Software Quality Engineering mit Eclipse
WinterSemester 2014/15

Dokumentation
zum Projekt „MediaStore“

Autoren:
Andreas Dillmann (okunev1983@gmail.com)
Anastasia Osintseva (osintseva.a@gmail.com

Betreuer:
Misha Strittmatter (strittmatter@kit.edu)

Inhalt

Aufgabenstellung

Im Rahmen des Praktikums müssen die folgenden Aufgaben erfüllt werden.

1. Komponentenarchitekturdiagramm muss überarbeitet werden.

2. Funktionalität implementieren/updaten:
   - Benutzermanagement (Registrierung, Login, Logout, Session Management)
   - Musik-Dateimanagement (Hochladen, Herunterladen)
   - Datenbankanbindung
   - Watermarking
   - MP3 (Re-)Encoding in verschiedenen Bitrates
   - Passworthashing mit Salz

3. Flexible Architektur
   - Komponentenbasiert
   - Komponenten austauschbar
   - Komponenten remotefähig

4. Schätzung der Performanz von remote Interfaces, die lokal verwendet werden.

Komponentenarchitekturdiagramm

Komponentenarchitekturdiagramm beschreibt die Struktur von der Applikation „Mediastore“ und Schnittstellen, die die Verbindung zwischen Komponenten ermöglichen. Es wurde keine Schnittstellen zwischen **MediaStore** und der Datenbank und dem Filesystem beschrieben, da die Datenbank und das Filesystem keine richtigen Komponenten von **MediaStore** sind. Es wurde vier alternativen Varianten von Komponentenarchitekturdiagramm vorgeschlagen.

Die erste Variante hat getrennte Watermarking- und Reencoding-Komponenten, wie es ab Anfang vorgeschlagen wurde. In diesem Fall wird Tagwatermarking wie in der früheren Implementierung von **MediaStore** umgesetzt. Das heißt, dass es einfach ein „Tag“ in die .mp3 Datei reingeschrieben wird. Reencoding macht die Umkodierung von einer .mp3 Datei, damit die Datei mit eingestellter Bitrate weitergeleitet werden kann.

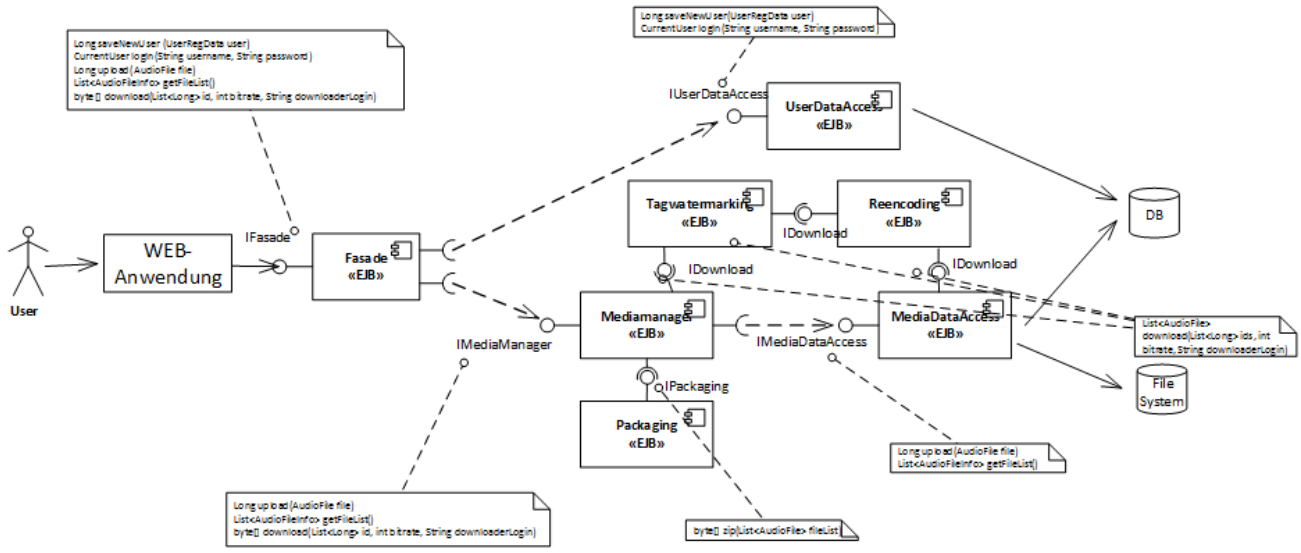


Abbildung 1. Variante #1: Getrennte Watermarking- und Reencoding-Komponenten

Die zweite Variante hat nur Watermarking-Komponente, aber Watermarking selbst wird als zusätzliche Information in .mp3 Datei auf nicht hörbare Frequenz geschrieben. Die Addition vom Information zu .mp3 Datei erfolgt zusammen mit der Umkodierung, deswegen ist es auch von der Performanz nicht sinnvoll, noch die Reencoding-Komponente zu verwenden.

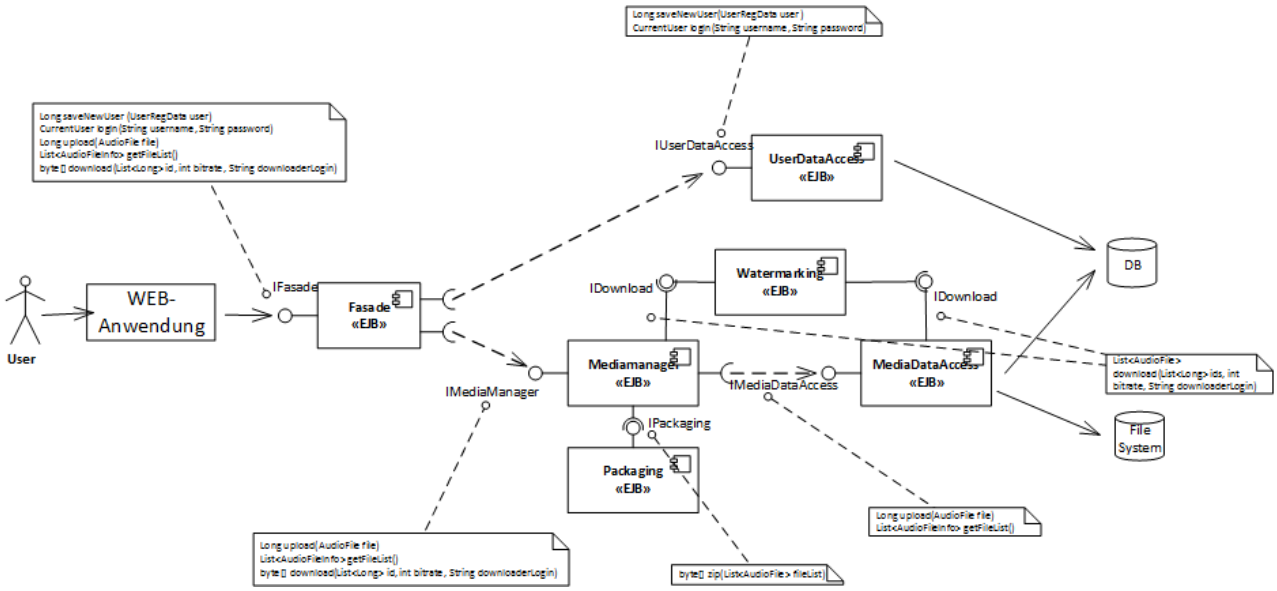


Abbildung 2. Variante #2: Watermarking-Komponente, die die Funktion von Reencoding übernimmt

Die dritte Variante hat nur Reencoding-Komponente. Das ist eine Variante, wenn die Watermarking von Endbenutzer nicht gebraucht ist.

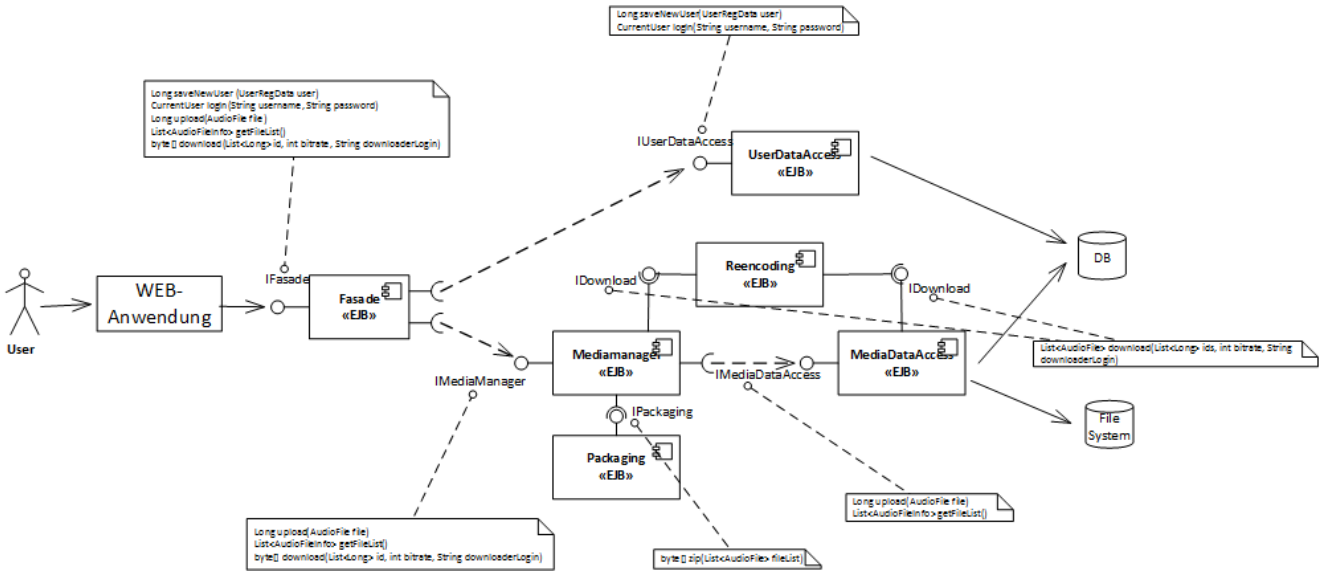


Abbildung 3. Variante #3: keine Watermarking, nur Reencoding

Die vierte Variante hat weder Watermarking- noch Reencoding-Komponenten. Das ist eine Variante, wenn die Umkodierung und Markierung von Endbenutzer gar nicht gebraucht sind.

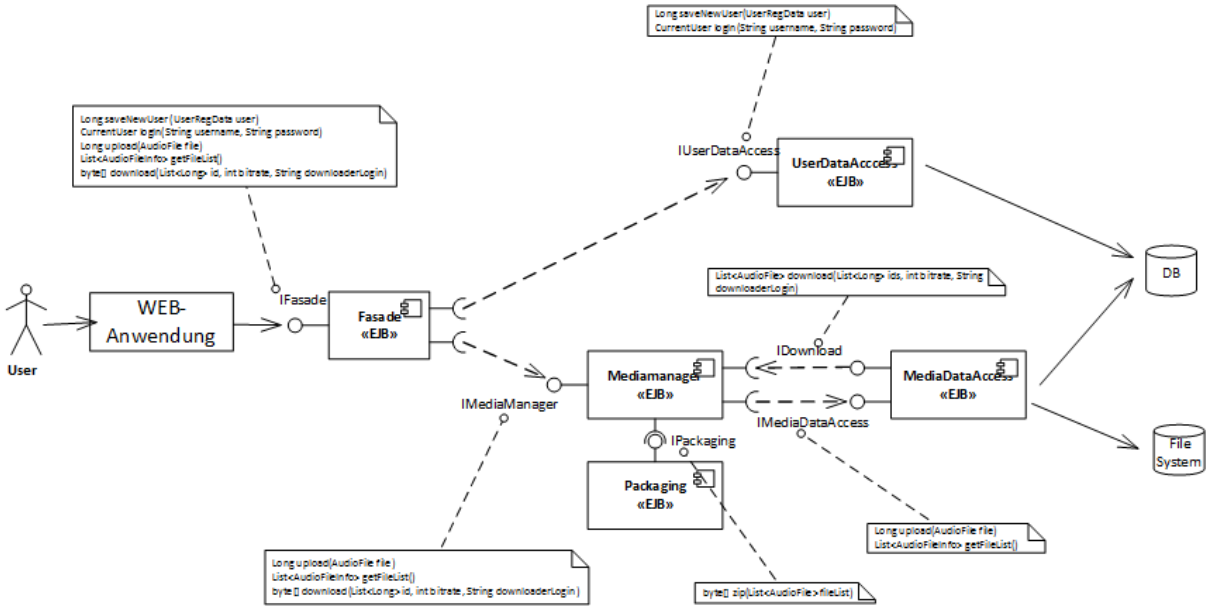


Abbildung 4. Variante #4: Keine Watermarking- und Reencoding-Komponenten

Technische Voraussetzungen

Um das Projekt erfolgreich einzurichten, sind die folgenden Technologien und Werkzeuge erforderlich:

- Eclipse Luna für Java EE Entwicklung
- Java EE 7 (dafür JDK 1.7 oder höher)
- GlassFish 4.1

Server Einrichtung

Für jede remote verteilte Komponente muss eine GlassFish-Server-Instanz konfiguriert werden.
Da die MediaDataAccess- und UserDataAccess-Komponenten zum Speichern, Organisieren und Abrufen von Daten die relationale Datenbank MySQL nutzen, muss die Verbindung mit der Datenbank ermöglicht werden.

In diesem Kapitel wird kurz dargestellt, wie man GlassFish installiert und eine JDBC-Ressource für die Datenbankverbindung erstellt.

GlassFish Installation

Zuerst lädt man ein .zip File mit Glassfish herunter und danach packt das .zip File in ausgewählte Ordner aus.
Glassfish kann von der Konsole gestartet und gestoppt werden:

- Zum Starten: *glassfish4\bin asadmin start-domain*
- Zum Stoppen: *glassfish4\bin asadmin stop-domain*

MySQL Installation

Es gibt verschiedene Möglichkeiten MySQL zu installieren. Im Rahmen dieser Ausarbeitung wird dafür XAMPP verwendet.
XAMPP ermöglicht die einfache Installation und Konfiguration des Webservers Apache mit der Datenbank MySQL. XAMPP enthält zusätzlich ein Werkzeug phpMyAdmin so, dass die Administration über HTTP mit einem Browser erfolgt.

Konfiguration der JDBC Ressource in Glassfish

1. Glassfish-Server muss gestoppt werden.
2. Es soll MySQL JDBC Driver heruntergeladen werden (mysqol-connector-java-5.1.34-bin.jar).
3. MySQL JDBC Driver muss in Ordner *$glassfish_install_folder\glassfish\lib* kopiert werden.
4. Jetzt kann Glassfish gestartet werden und die Admin-Konsole im Browser aufgerufen werden. Bei Default ist die Admin-Konsole unter http://localhost:4848 erreichbar.
5. Es soll *Resources\JDBC\JDBC Connection Pools* geöffnet werden. Jetzt kann Connection Pool mit folgende Eigenschaften erstellt werden (Abbildung 4):

   Pool name: NewPool
   Resource type: java.sql.DataSource
   Database Driver Vendor: MySQL.

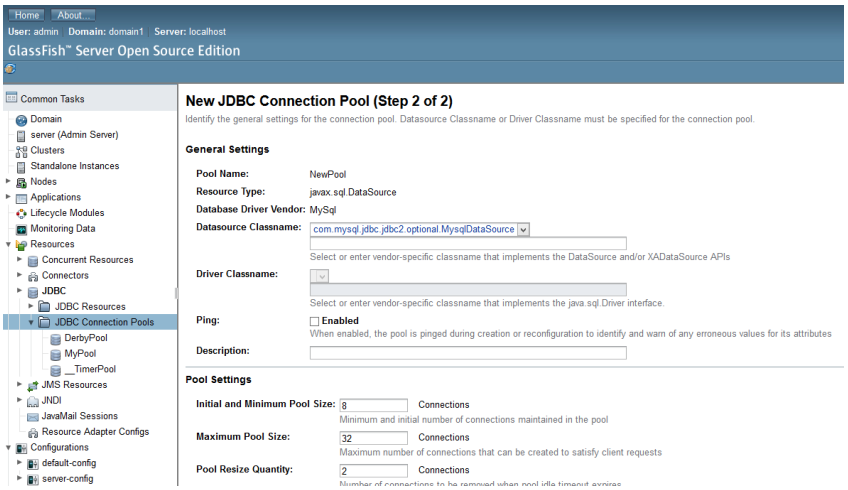


Abbildung 5. Neues Connection Pool

6. Jetzt kann rechts oben auf Next geklickt werden. Als weiteres sollen die nächsten zusätzlichen Eigenschaften festgelegt werden (Abbildung 5):

   port: 3306

   URL: jdbc:mysql://localhost:3306/mediastore

   ServerName: localhost

   DatabaseName: mediastore

   user: hier soll der Name des Benutzers, den auf die Datenbank zugreifen wird, geschrieben werden.

   password: hier soll das Passwort des Benutzers geschrieben werden. Es muss beachtet werden, dass das Passwort unverschlüsselt gespeichert wird.

Es soll rechts unten auf Finish geklickt werden. Nun ist ein Connection Pool mit dem Name NewPool erstellt.

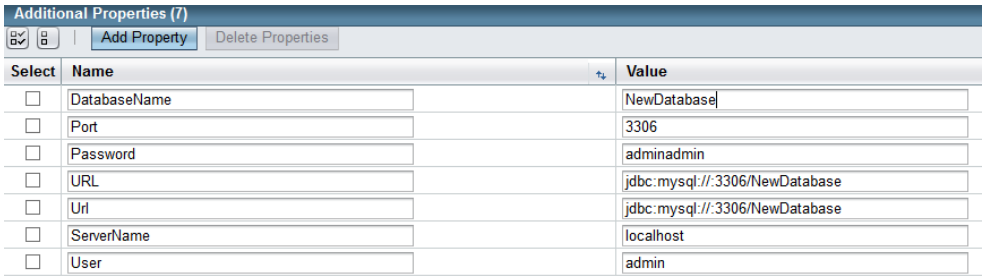


Abbildung 6. Zusätzliche Eigenschaften

7. Es kann jetzt überprüft werden, ob ein neues Connection Pool richtig erzeugt wurde. Dafür unter Resources\JDBC\JDBC Connection Pools soll NewPool ausgewähl werden und auf Ping geklickt. Wenn es eine Meldung „Ping Succesededˮ kommt, dann wurde alles richtig gemacht.
8. Jetzt kann die JDBC-Ressource geschafft werden, die den Zugriff auf NewPool aus der Java-Programme ermöglicht. Dafür unter Resources\JDBC\JDBC Resources auf New geklickt werden soll und ein JDBC-Ressource mit folgende Eigenschaften erzeugt:

   JNDI Name: Mediastore

   Pool Name: NewPool

Projektstruktur

Das Projekt „MediaStore“ besteht aus:

- einem einfachen Java Projekt als Hilfsprojekt(in Form jar-Datei)
- sieben JEE–Modulen (in Form einer jar–Datei) als Backend
- einer Web–Applikation als Frontend (in Form einer war–Datei)

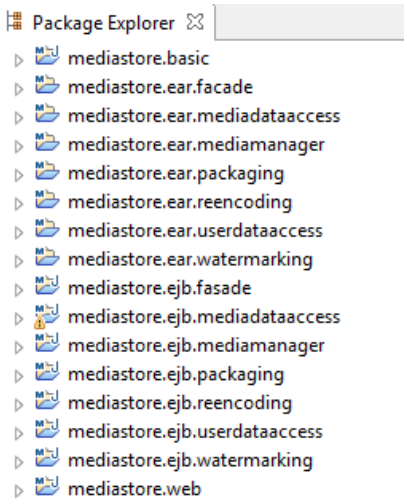


Abbildung 7: Grobstruktur des Projektes

Jedes EAR Projekt stellt zusammen mit dem entsprechenden EJB Projekt eine Komponente aus dem Komponentenarchitekturdiagramm dar. Nachdem eine **.jar-Datei** aus einem EJB Projekt erstellt wird, wird sie von dem EAR Projekt benutzt. GlassFish-Server kann jetzt das EAR Projekt deployen.

Die Abhängigkeiten zwischen Projekten wurden mittels Framework Maven eingerichtet. Alle EJB und EAR Projekte sind Maven-Projekte.

Beispielsweise wurde die Dependency zum Projekt **mediastore.basic** folgendes erstellt:

```
<dependencies>
    <dependency>
        <groupId>mediastore.basic</groupId>
        <artifactId>mediastore.basic</artifactId>
        <version>1.0</version>
    </dependency>
</dependencies>
```

Um ein JEE Projekt aufzubauen, wird auch **maven** verwendet. Für das automatische Deployment des Projektes wird **pom.xml** in EAR Projekt folgendes konfiguriert:

- Fügen ein **EAR Plugin** ein

```
<project>
 [...]
 <build>
  [...]
  <plugins>
   [...]
   <plugin>
    <artifactId>maven-ear-plugin</artifactId>
    <version>2.10</version>
    <configuration>
     <!-- configuration elements goes here -->
    </configuration>
   </plugin>
 [...]
</project>
```

Hier ist ein Beispiel aus dem mediastore.ear.UserDataAcces:

```
<plugin>
	<artifactId>maven-ear-plugin</artifactId>
	<version>2.9</version>
	<configuration>
		<finalName>mediastore</finalName>
		<earSourceDirectory>EarContent</earSourceDirectory>
		<version>7</version>
		<defaultLibBundleDir>lib</defaultLibBundleDir>
		<skinnyWars>true</skinnyWars>
		<modules>
			<ejbModule>
				<groupId>mediastore.ejb.UserDataAcces</groupId>
				<artifactId>mediastore.ejb.UserDataAcces</artifactId>
			</ejbModule>
		</modules>
	</configuration>
</plugin>
```

- Jede EAR-Modul muss weiter folgendes angepasst werden:
  - bundleDir: das Verzeichnis, in der EAR-Struktur, bei der das Artefakt gespeichert werden
  - bundleFileName: der Name des Artefakts in der EAR-Struktur
  - uri: der vollständige Pfad in der EAR-Struktur für das Artefakt
  - excluded: schließt das Artefakt aus der generierten EAR
  - unpack: Packen das Artefakt in der generierten EAR

Hilfsprojekt **mediastore.basic**

mediastore.basic enthält verschiedene Hilfsklassen, die von allen Komponenten des Projekts benutzt werden, und alle nötigen Schnittstellen, die in entsprechenden EJBs implementiert werden.

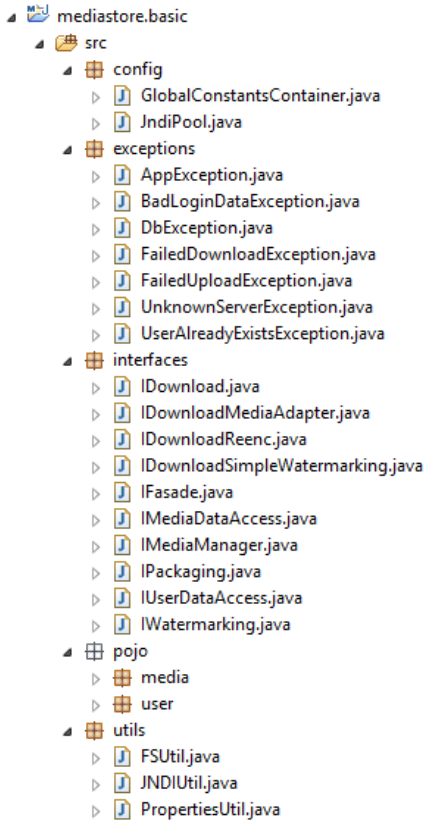


Abbildung 8: Projekt **mediastore.basic**

Das Enum-Objekt **JndiPool** (mediastore.basic.src.config.JndiPool.java) beinhaltet die vollständige Information über die remoteverteilten Komponenten, nämlich:

- der globale JNDI Name der Komponente
- der Hostname und der Port des Servers, auf dem die Komponente installiert wurde
- vollständiger Pfad zu einer konkreten ContextFactory des jeweiligen EJBs.

Beispiel für das „UserDataAcces“ Bean:

```
USER_ADAPTER(
        "java:global/mediastore.ear.UserDataAcces/mediastore.ejb.UserDataAcces-1.0/UserDataAccesImpl", //JNDI Name
        "localhost", // Hostname
        "3700",  //Port
        "com.sun.enterprise.naming.SerialInitContextFactory" // Context Factory für GlasssFish
)
```

mediastore.basic.src.pojo enthält die komplexen Typen für das Media bzw. den Benutzer.

mediastore.basic.src.utils enthält Hilfsklassen. Beispielweise vereinfacht die Klasse „JNDIUtil“ das Lookup, indem das gewünschte Interface an die Methode **JNDIUtil#find** übergeben wird.

mediastore.basic.src.exceptions enthält alle möglichen Exceptions, die im Lauf der Applikation auftauchen könnten.

Backend: EJB/EAR–Projekte

Backend besteht aus 7 Modulen (7 EARs und 7 EJBs):

- **mediastore.ear.fasade**
- **mediastore.ejb.fasade**(fasade ist eine Schnittstelle zwischen Weboberfläche und Businneslogik der Applikation)
- **mediastore.ear.userdataAcces**
- **mediastore.ejb.userdataAcces**(UserDataAcces kümmert sich um die Benutzerdatenmanagement)
- **mediastore.ear.mediamanager**
- **mediastore.ejb.mediamanager**(mediamanager ist eine Schnittstelle zwischen Komponenten, die die Dateiübertragung, die Umkodierung und die Verpackung implementieren)
- **mediastore.ear.mediadataacces**
- **mediastore.ejb.mediadataacces (**mediadapter kümmert sich um die Mediadateien. Er speichert .mp3 Datei und die zugehörige Informationen und holt das ganze von der Datenbank und Festplatte bei der Anfrage zurück)
- **mediastore.ear.reencoding**
- **mediastore.ejb.reencoding**(reencoding kodiert die .mp3 Datei mit angefragter Bitrate um)
- **mediastore.ear.watermarking**
- **mediastore.ejb.watermarking**(watermarking markiert die herunterladbaren .mp3 File)
- **mediastore.ear.packaging**
- **mediastore.ejb.packaging**(packaging packt die herunterladbaren .mp3 Files in .zip File)

Anmerkungen:
1. Da die .mp3 Dateien für die Umkodierung und Watermarking irgendwie zwischengespeichert werden sollen, wird für diesen Zweck ein RamDisk verwendet. Es wird RmDisk von SoftPerfect gewählt. (RamDisk ist Freeware für private und nicht kommerzielle Benutzung)
2. Da die EJBs mittels RMI nur die serialisierbare Datentypen zwischen einander übertragen könne, werden die Files als byte[] übertragen.

Weiter werden nur die wichtigsten Komponenten von Backend detailliert betrachtet.

Komponente Fasade (**mediastore.ear.fasade + mediastore.ejb.fasade**)

Die **Fasade** Komponente bietet eine vereinheitliche Schnittstelle für einem Satz von Schnittstellen des Systems. Das Modul besteht nur aus einer Klasse(**FasadeImpl**), die ganze Menge komplexerer Klassen, die zu dem Projekt gehören, vereinfacht und vereinheitlicht.

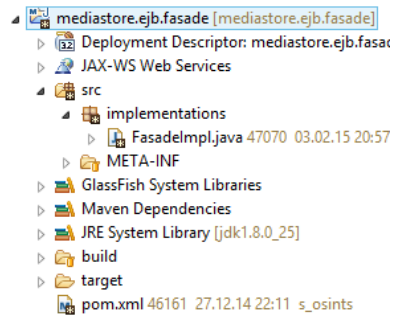


Abbildung 9: Projekt **mediastore.ejb.fasade**

Komponente UserDataAcces (**mediastore.ear.userdataacces + mediastore.ejb.userdataacces**)

Die **UserDataAcces** Komponente übernimmt die Aufgabe von Userdatenmanagement: Registration und Anmeldung.

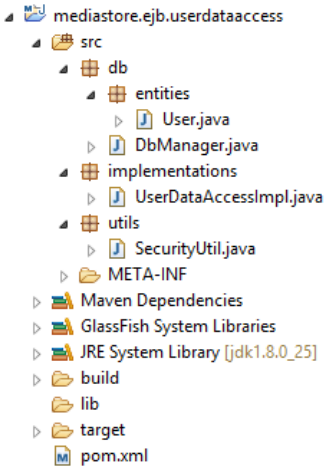


Abbildung 10: Projekt **mediastore.ejb.userdataacces**

mediastore.ejb.userdataacces ist ein JPA Projekt. Um die Datenbankzugriffe und das objektrelationale Mapping zu ermöglichen, wird EclipseLink 2.5 verwendet.

Damit JPA verwendet werden kann, sind die folgenden Konfigurationen notwendig:

- **persistence.xml** - definiert *persistence units*

```
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
                <persistence-unit name="" transaction-type="JTA">
                        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
                        <jta-data-source>jdbc/Mediastore</jta-data-source>
                        <class>db.entities.User</class>
                        <properties>
                            <property name="eclipselink.ddl-generation" value="create-tables"/>
                        </properties>
                </persistence-unit>
</persistence>
```

Hier ist wichtig jta-data-source zu setzen. Wie die Datenquelle in GlassFish konfiguriert wurde, wurde im Kapitel *"Konfiguration der JDBC Ressource in Glassfish"* betrachtet.

Eclipselink kann auch verwendet werden, um automatisch die Tabellen und Datenbankschema aus Java-Klassen zu erzeugen. Dies wird durch die persistence unit Property **eclipselink.ddl-generation** ermöglicht, die entweder auf *create-tables* oder *drop-and-create-tables* gesetzt sein muss.

create-tables – wenn die Tabelle bereits existiert, so wird sie nicht gelöscht oder ersetzt. Es wird die vorhandene Tabelle verwendet.

drop-and-create-tables – wird zuerst die vorhandene Tabelle gelöscht und dann die neue Tabelle erstellt.

- **Entity**

Es wird die Annotation @Entity verwendet, um anzugeben, dass eine Klasse ist eine Entity. In UserDataAccess ist die Klasse **User** eine Entity:

```
@Entity
@NamedQueries({ @NamedQuery(name = "findAll", query = "SELECT e FROM User e"), @NamedQuery(name =
"findByEmail", query = "SELECT e FROM User e WHERE e.email = :email") })
public class User implements Serializable {
private static final long serialVersionUID = -7332870302408416956L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
@Column(nullable = false)
private String firstname;
@Column(nullable = false)
private String lastname;
@Column(nullable = false)
private String email;
@Column(nullable = false)
private String password;
```

Anmerkungen:

- @NamedQueries *ist eine statisch definierte Anfrage mit einer vordefinierten unveränderlichen Query-String*
- *@ Id-Annotation bestimmt Primary Key*
- *@GeneratedValue spezifiziert einen Primärschlüsselgenerator vom Typ TABLE*
- *@Column legt für den Spalten bestimmte Eigenschaften(z.B nullable) fest*

db.DbManager enthält die Methode: **saveUser** und **findUser.** Diese Methoden wurden bei Registration bzw. bei Anmeldung aufgerufen. Die Methoden verwenden die Hilfsklasse **SecurityUtil,** die aus dem eingegebenen Passwort eine Hashwert berechnet. Dafür wird der folgende Algorithmus realisiert:

1. Generieren eine Zufallsfolge: Salz
2. Hashen das Passwort zusammen mit dem Salz (salted hash): SHA-512
3. Fügen das Salz zu dem Ergebnis hinzu(die letzten 10 Bytes)
4. Speichern das gehashte Passwort in Datenbank

Komponente MediaDataAcces (**mediastore.ear.mediadataacces** + **mediastore.ejb. mediadataacces**)

Die **MediaDataAcces** Komponente übernimmt die Aufgabe von Mediadatenmanagement: Speichern der Infodaten über die Medien in die Datenbank, Speichen die .mp3 Dateien in das Filesystem, Lieferung der Metadaten und Audiodateien.

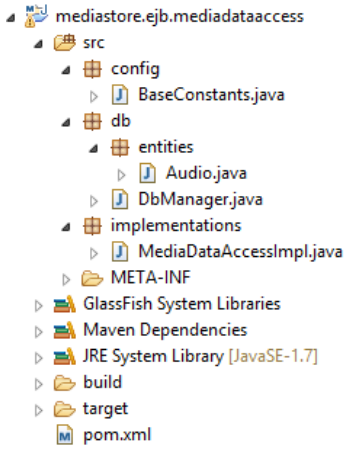


Abbildung 11: Projekt **mediastore.ejb.mediadataacces**

mediastore.ejb.MediaDataAcces ist auch ein JPA Projekt. **JPA** wurde ähnlich dem UserDataAccess konfiguriert. In MadiaDataAccess ist die Java-Klasse **Audio** eine Entity:

```
@Entity
@TableGenerator(name = "audio")
@NamedQueries({ @NamedQuery(name = "findAll", query = "SELECT e FROM Audio e"), @NamedQuery(name =
"findByTitle", query = "SELECT e FROM Audio e WHERE e.title = :title"),
@NamedQuery(name = "findByID", query = "SELECT e FROM Audio e WHERE e.id = :id")})
public class Audio implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.TABLE)
private Long id;
private String album;
@Column(nullable = false)
private String artist;
@Column(nullable = false)
private Integer bitrate;
private String genre;
private Integer releaseyear;
@Column(nullable = false)
private String title;
@Column(nullable = false)
private Long userId;
```

MadiaDataAccess enthält ebenfalls die Klasse **DbManager,** um den Datenbankzugriff zu ermöglichen.
Beispielweise speichert die Methode **DbManager#saveAudioFile** die an sie übergebene Information über eine .mp3 Datei in die Datenbank.
Die Methode **DbManager# getAudioByID** gibt diese Information bei der Angabe einer Identifikationsnummer zurück.

MediaDataAccesImpl ermöglicht das Herunterladen und Laden. Die Methode **MediaDataAccesImpl #download** wird aufgerufen, wenn der Benutzer eine oder mehrere Mediadateien herunterladen will und die Methode **MediaDataAccesImpl #upload –** entsprechend bei dem Hochladen der Datei auf den Server. Der Ordner, in dem die Dateien speichern werden, ist in der Klasse **BaseConstants** gesetzt. Der vollständige Pfad wird aus dem Albumname, Artistname und dem Filename aufgebaut.

Frontend : Web–Applikation

Das Projekt **mediastore.web** implementiert die Clientseite von **MediaStore** aus dem Komponentenarchitekturdiagramm und hat die folgende Struktur:

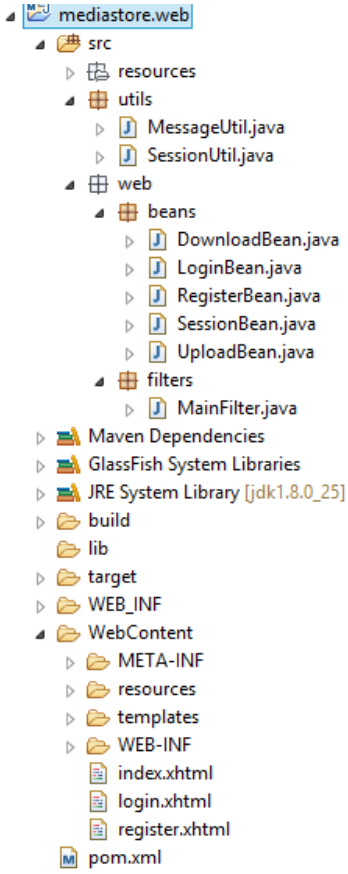


Abbildung 12: Projekt **mediastore.web**

Zur Entwicklung von grafischen Benutzeroberfläche wurde das Framework **JavaServer Faces** verwendet. Die JSF-Implementierung wurde mit dem Komponentenframework **PrimeFaces** erweitert.

In dem Projekt wurde drei Komponenten implementiert:

- **login.xhtml** stellt die Anmeldungsseite dar
- **register.xhtml** stellt die Registrationsseite dar
- **index.xhtml** stellt die Hauptseite von **MediaStore** dar, die das Laden und Herunterladen von .mp3 Dateien für registrierte Endbenutzer anbietet.

Hinter den Komponenten liegen Managed-Beans, die die eigentlichen Werte zum Befüllen der Komponenten liefern:

- **DownloadBean** initiiert Download-Funktion von **MediaStore**
- **LoginBean** ermöglicht das Einloggen von den bereits registrierten Benutzer
- **RegisterBean** ermöglicht die Anmeldung vom neuen Benutzern
- **SessionBean** wird benutzt als ein Container für das Speichern der für die Session relevanten Daten.
- **UploadBean.java** initiiert die Upload-Funktion von **MediaStore**

JavaResources.src.web.filters enthält den Hauptfilter der Webanwendung. Der Filter beschränkt den Benutzern den Zugriff auf die bestimmten Seiten der Website in JSF und leitet sie zu den für sie erlaubten Seiten. Zum Beispiel, wenn der Benutzer nicht angemeldet ist, kann er nicht die Hauptseite mit den Mediadateien zugreifen, deswegen wird er zur Anmeldungsseite umgeleitet.

JavaResources.src.utils enthält zwei Hilfsklassen:

- **MessageUtil** übernimmt die Benachrichtigung der Benutzer auf der Weboberfläche. (Zum Beispiel Error- oder Successmessage)
- **SessionUtil** hat bisher nur eine Methode **isLoggedIn.** Sie gibt true zurück, falls der Benutzer eingeloggt ist.

Funktionalität

Weiter im Text wird beschrieben, wie die im Abschnitt „Komponentenarchitekturdiagramm“ beschriebenen Funktionen implementiert sind.

- Register

Nach der Eingabe von Benutzerdaten wird die Methode **mediastore.web.JavaResources.src.web.beans.RegisterBean#doRegister** aufgerufen. Dann wird die Komponente **Fasade** mittels Hilfsmethode find(JndiPool) der Klasse **mediastore.basic.src.utils.JNDIUtil** gefunden. Weiter werden die Benutzerdaten über **Fasade** nach **UserDataAcces** zum Speichern übergegeben. **UserDataAcces** speichert die Benutzerdaten in der Datenbank.

- Einloggen

Nach der Eingabe von Email und Passwort wird die Methode **mediastore.web.javaResources.src.web.beans.LoginBean#doLogin** aufgerufen. Die eingegebenen Daten werden über **Fasade** nach **UserDataAcces** weitergeleitet und überprüft, ob ein Benutzer mit den angegebenen Anmeldungsdaten existiert. Falls Ja, wird der Benutzer eingeloggt.

- Hochladen (Upload)

Nachdem der Benutzer die erforderlichen Daten und der Pfad zu .mp3 Datei eingegeben hat, wird die Methode **mediastore.web.JavaResources.src.web.beans.UploadBean#upload** ausgeführt.
Das .mp3 File und die zugehörigen Daten werden als Datentyp **mediastore.basic#AudioFile** nach **Fasade**, von **Fasade** nach **Mediamanager** und von **Mediamanager** nach **MediaDataAcces** übertragen. **Mediamanager** speichert das .mp3 File auf die Festplatte und die zugehörigen Metadaten in die Datenbank.

- Herunterladen (Download)

Wenn der Benutzer die .mp3 Files (das .mp3 File) ausgewählt und die gewünschte Bitrate eingegeben hat, wird die Methode **mediastore.web.JavaResources.src.web.beans.DownloadBean#download** ausgeführt.
MediaDataAcces holt die .mp3 Files von der Festplatte und die zugehörigen Metadaten aus der Datenbank und übergibt die weiter nach:

1. Variante: Reencoding
   Die .mp3 Files mit der gegebenen Bitrate werden umkodiert. **Reencoding** übergibt die Daten nach **Tagwatermarking**, wo die Watermarking-Tags zu .mp3 Files reingeschrieben werden.
2. Variante: Watermarking
   Die .mp3 Files mit der gegebenen Bitrate werden umkodiert und die zugehörige Markierung wird gemacht.
3. Variante: Reencoding
   Die .mp3 Files mit der gegebenen Bitrate werden umkodiert.
4. Variante: Mediamanager

Weiter werden die Daten mittels der Methode **packaging.JavaResources.src.implementations.beans.PackagingImpl#zip** von **Packaging** in .zip File verpackt. P**ackaging** liefert .zip File über **MediaManager** und **Fasade** nach Methode **mediastore.web.JavaResources.src.web.beans.DownloadBean.#download** zurück. Das .zip File wird auf die Festplatte vom Benutzer gespeichert.

Zusammenfassung

In Rahmen des Praktikums wurden die folgenden Ziele erreicht:

- Die Architektur von MediaStore wurde überarbeitet und in Java EE umgesetzt.
- Alle Komponenten der Architektur wurden remotefähig gemacht, es heißt, dass alle Beans remote verfügbar sind und frei deployed werden können.
- Dazu wurde die Performanz von remote Interfaces, die innerhalb einer GlassFish-Instanz verwendet werden, mit der Performanz von lokalen Interfaces vergliechen.
- Die Weboberfläche wurde neue implementiert.
- Das Passwort wurde beim Speichern durch Hashing mit Salz gesichert.
- Die Implementierung wurde auf den verschiedenen Rechnern und Operationssystemen getestet. Beim Testen wurden die Komponente remote verteilt.

Arbeitsteilung

- Überarbeitung der Komponentenarchitekturdiagramm (Andreas)
- Packaging (Andreas)
- MP3 (Re-)Encoding in verschiedenen Bitrates (Andreas)
- Neue Weboberfläche (Anastasia)
- Benutzermanagement (Registrierung, Login, Logout, Session Management) (Anastasia)
- Datenbankanbindung (Anastasia)
- Passworthashing mit Salz (Anastasia)
- Testen des Projekts auf verschiedenen Rechnern mit verschiedenen Komponentenverteilungen (Anastasia, Andreas)

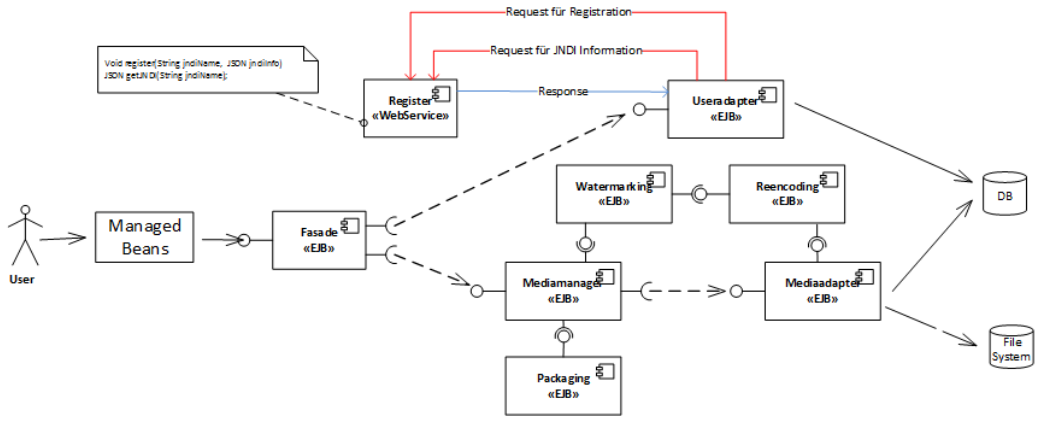
Ausblick

Problemstellung:

- Adressinformationen der Komponenten sind hartcodiert bzw. definiert in einer Konfigurationsdatei.
- Problem: Keine Flexibilität. Komponenten müssen bei einer Änderung neukompiliert bzw. neugestartet werden.

Mögliche Lösung:

- Eine zentrale Stelle, die Adressen verwaltet.



Literaturverzeichnis

EJB 3.0 Specification Final Release . [Online] http://www.oracle.com/technetwork/java/javaee/ejb/index.html.

Administration Guide. [Online] 2013. https://glassfish.java.net/docs/4.0/administration-guide.pdf.

Application Development Guide. [Online] https://glassfish.java.net/docs/4.0/application-development-guide.pdf.

Enterprise JavaBeans classes and interfaces. [Online] http://docs.oracle.com/javaee/6/api/javax/ejb/package-summary.html.

Java Platform, Enterprise Edition (Java EE) 7: The Java EE Tutorial. [Online] https://docs.oracle.com/javaee/7/tutorial/index.html.

Maven in 5 Minutes. [Online] http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html.

Quick Start Guide. [Online] https://glassfish.java.net/docs/4.0/quick-start-guide.pdf.

Skript zur Stammvorlesung "Sicherheit". Arbeitsgruppe für Kryptographie und Sicherheit: 2014.

Stackoverflow. [Online] http://stackoverflow.com.