HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Fakultät **Mathematik/Naturwissenschaften/Informatik**

# BELEGARBEIT

## Verteilte Systeme

**Entwickelung eines lauffähigen RESTful Webservice mit freiwählbaren Thema unter Zuhilfenahme von JAX-RS**

Autor:
**Patrick Reichelt**

Studiengang:
Medieninformatik und Interaktives Entertainment

Seminargruppe:
MI12W2-B

Begleitender Professor:
Prof. Dr.-Ing. Andreas Ittner

Begleitender Dozent
M.Sc. Maik Benndorf

Mittweida, 2015

# I. Inhaltsverzeichnis

# II. Abbildungsverzeichnis

# 1 Aufgabenstellung & Motivation

Im Rahmen des Faches Verteilte Systeme ist ein Beleg zu erbringen, in welchem ein lauffähiger RESTful Webservice mit frei wählbarem Thema unter Verwendung von JAX-RS zu entwickeln ist. Die Funktionen dieser Services sollen später in einer Präsentation mit Hilfe eines Test-Clients nachgewiesen werden.Die Funktionen Anzeigen, Anlegen, ändern und Löschen von Ressourcen sollen dabei möglich sein. Als Rückgabetypen werden sowohl XML, als auch JSON mittels JAXB erwartet. Zudem soll der Service mittels JUnit-Tests erfolgreich getestet werden.

Für die Umsetzung einen RESTful Webservice wurde das Thema Lernkarten Management gewählt. In dieser können Kartensammlungen angelegt und abgerufen werden. Hierfür wurden drei Resourcen (Kartenstapel, Karte, Autor) zur Vefügung gestellt. Ein Kartenstapel muss einen Autor und eine oder mehrere Karten beinhalten. Für dieses Thema wurde sich entschieden, da bereits ähnliche Umsetzung mit WCF gemacht wurden. Für die Datenhaltung wurde sich hierfür MongoDB entschieden, da noch keine Erfahrungen mit NoSQL Datenbanken vorhanden sind und sich hier die Gelegenheit bietet diese zu sammeln.

# 2 Entwurf

## 2.1 Komponenten

### 2.1.1 easylearnServices.Core

Das Modul ”easylearnServices.Core” beinhaltet drei Packages, über die Funktionen zur Verfügung gestellt werden, wie die Kommunikation mit dem Datenbank Server, das Mappen der Daten auf die jeweiligen Objekt Modele und die die Model Klassen selber. Im Package ”easylearnServices.Core.Common” findet Man das Interface und die Implementierung die später vom REST Service aufgerufen werden, um Daten aus der Datenbank abzurufen oder zu ihr zu senden. Das Package ”easylearnServices.Core.Models” beinhaltet die oben genannte Model Klassen, die Interface Methoden Implementieren um die Daten aus der MongoDB auf ihre Datenfelder zu mappen oder für das Speichern in dieser vorzubereiten. Um die Vorbereitung und Konfiguration aller Tests kümmert sich das Package ”easylearnServices.Core.Test”, in der die Verbindung zum Test Datenbank Server aufgebaut wird und diese auch mit Testdaten befüllt.

### 2.1.2 easylearnServices.Rest

Den eigentlichen REST Service findet man im Modul ”easylearnServices.Rest”, in dem alle Endpoints definiert sind. Für diese wurden zwei Klassen angelegt, CardstackEndpoint und AuthorEndpoint, in denen die Methoden die im Interface des Core Moduls definiert sind aufgerufen werden. Die Methoden der Klassen werden dabei mit Pfad angaben, Parametern und HTTP Methoden gekennzeichnet. Diese geben an, auf welche HTTP Mehtode (GET, POST, PUT, DELETE) die jeweilige Funktion hören soll, unter welchem URL Pfad sie zu erreichen ist und welche Parameter dabei übergeben werden und welcher Art diese sind.

Für den Test Methoden wurden Integrationstests geschrieben die im Test Verzeichnis zu finden sind.

## 2.2 Datenpersistierung

Für die Persistierung der Daten wurde MongoDB gewählt. MongoDB ist eine Dokumentenbasierende NoSQL Datenbank. Für dieses Projekt wurden drei sogenannte Collections angelegt, jeweils für Cardstack, Card und Author. In der Cardstack Collection wer-

den sämtliche Daten über den Kartenstapel gespeichert so auch die ReferenzID zum Autor Datensatz. Ein Dokument in der Card Collection beinhaltet die Daten zu einer einzelnen Karte und die ReferenzID zum zugehörigen Kartenstapel. Die Author Collection beinhaltet Dokumente die Informationen über den Author geben. Für die Kommunikation wurden die offiziellen Treiber von MongoDB.org in der Version 3.0 Beta genutzt.

# 3 Implementierung

## 3.1 Endpoint Definition: Cardstack

**Endpoint /cardstack**

| | |
|---|---|
| **POST** | Erstellt einen neuen Kartenstapel<br>200 OK bei erfolgreichem Erstellen<br>400 BAD REQUEST bei fehlerhaftem Aufruf |
| **GET** | Gibt eine Liste aller eingetragenen Kartenstapel<br>200 OK bei erfolgreichem Erstellen<br>400 BAD REQUEST bei fehlerhaftem Aufruf |

**Endpoint /cardstack/<id>**

| | |
|---|---|
| **PUT** | Aktualisiert die Daten eines Autors<br>200 OK bei erfolgreichen Aktualisierung<br>400 BAD REQUEST bei fehlerhaftem Parameter<br>404 NOT FOUND wenn der zu Aktualisierende Datensatz nicht gefunden wurde |
| **GET** | Ruft die Daten eines Autors ab<br>200 OK bei erfolgreichem Abrufen<br>404 NOT FOUND wenn kein Autor gefunden wurde |
| **DELETE** | Löscht einen Autor<br>200 OK bei erfolgreichem Löschen<br>404 NOT FOUND Wenn kein Autor gelöscht werden konnte |

**Parameter für die Operationen PUT und POST**

| | |
|---|---|
| **name** | Name des Kartenstapels |
| **description** | Beschreibung über den Inhalt |
| **allowsharing** | Gibt an ob der Kartenstapel mit andern geteilt werden kann |
| **topic** | Das Thema zu einen Kartenstapel |
| **author_refid** | Die Id des Autors |

## 3.2 Endpoint Definition: Card

**Endpoint /cardstack/<id>/card**

| | |
|---|---|
| **POST** | Erstellt eine neue Karte zum zugehörigen Kartenstapel<br>201 CREATED bei erfolgreichem Erstellen<br>400 BAD REQUEST bei fehlerhaftem Aufruf |
| **GET** | Ruft eine Liste von Karten ab die zu einem Kartenstapel gehören<br>200 OK bei erfolgreichem Erstellen<br>404 NOT FOUND bei fehlerhaftem Anruf |

**Endpoint /cardstack/<id>/card/<cardId>**

| | |
|---|---|
| **PUT** | Aktualisiert die Daten einer Karte<br>200 OK bei erfolgreichen Aktualisierung<br>400 BAD REQUEST bei fehlerhaftem Parameter<br>404 NOT FOUND wenn der zu Aktualisierende Datensatz nicht gefunden wurde |
| **GET** | Ruft die Daten einer Karte ab<br>200 OK bei erfolgreichem Abrufen<br>404 NOT FOUND wenn kein Autor gefunden wurde |
| **DELETE** | Löscht eine Karte<br>200 OK bei erfolgreichem Löschen<br>404 NOT FOUND Wenn kein Autor gelöscht werden konnte |

**Parameter für die Operationen PUT und POST**

| | |
|---|---|
| **name** | Name der Karte |
| **definition** | Definition die auf der Karte stehen kann |
| **term** | Begriff der auf der Karte stehen kann |
| **shortcut** | Abkürzung die auf der Karte stehen kann |
| **cardstack_refid** | Referenz ID des Kartenstapels zu dem die Karte gehört |

## 3.3 Endpoint Definition: Author

**Endpoint /author**

| | |
|---|---|
| **POST** | Erstellt eine neuen Autor<br>201 CREATED bei erfolgreichem Erstellen<br>400 BAD REQUEST bei fehlerhaftem Aufruf |
| **GET** | Ruft eine Liste von Autoren ab<br>200 OK bei erfolgreichem Abruf<br>404 NOT FOUND bei fehlerhaftem Anruf |

**Endpoint /author/<id>**

| | |
|---|---|
| **PUT** | Aktualisiert die Daten eines Autors<br>200 OK bei erfolgreichen Aktualisierung<br>400 BAD REQUEST bei fehlerhaftem Parameter<br>404 NOT FOUND wenn der zu Aktualisierende Datensatz nicht gefunden wurde |
| **GET** | Ruft die Daten eines Autors ab<br>200 OK bei erfolgreichem Abrufen<br>404 NOT FOUND wenn kein Autor gefunden wurde |
| **DELETE** | Löscht einen Autor<br>200 OK bei erfolgreichem Löschen<br>404 NOT FOUND Wenn kein Autor gelöscht werden konnte |

**Parameter für die Operationen PUT und POST**

| | |
|---|---|
| **firstName** | Vorname des Autors |
| **lastName** | Nachname des Autors |
| **userName** | Benutzername des Autors |
| **eMail** | E-Mail Adresse des Autors |
| **password** | Passwort des Autors |

# 4 Zusammenfassung

Nach anfänglichen Schwierigkeiten erwies sich die Entwicklung der Service Komponente als sehr angenehm, da Maven alle Funktionen zur verfügung stellt die gebraucht werden. Die Paketverwaltung mit Maven wurde trotzdem als eher umständlich empfunden. Letztendlich war es die Anbindung der Datenbank, die die größten Probleme bereitete da die Dokumentation nicht sehr Aussagekräftig war und das Angebot der Code Beispiele auch sehr gering. Da sich für die Benutzung der Beta Version des Datenbanktreibers entschieden wurde, kam es zu Problemen beim Aktualisieren eines Datensatzes, die dann über, eine umständlichere Implementierung ersetzt werden musste. Letztlich konnten die Erfahrungen in der Implementierung eines Web Service vertieft werden und auch erste Erfahrungen mit NoSQL Datenbanken konnten gesammelt werden. Es war interessant zu sehen an welchen Stellen sich die Implementierung der Verschieden Technologien wie der mir bekannten Windows Communication Foundation und der hier angewendeten Unterscheiden.

# 5 Installationshinweise

Für die lokale Ausführung auf einem PC ist die Installation von Maven und eine aktuelle Java-Version vorrausgesetzt. Zum Testen wurde der integrierte RestClient der IDE IntelliJ genutzt. Der Webservice kann mit Hilfe des Kommandos mvn clean install im Verzeichnis des Hauptprojekts (easylearn-services) kompiliert werden. Anschließend ist es möglich diesen mit dem Kommando mvn jetty:run im Verzeichnis des REST-services (easylearn-services.rest) auszuführen. Der Webservice ist dann über die Adresse http://localhost:8080 erreichbar.

# Anhang A: Klassendiagramme

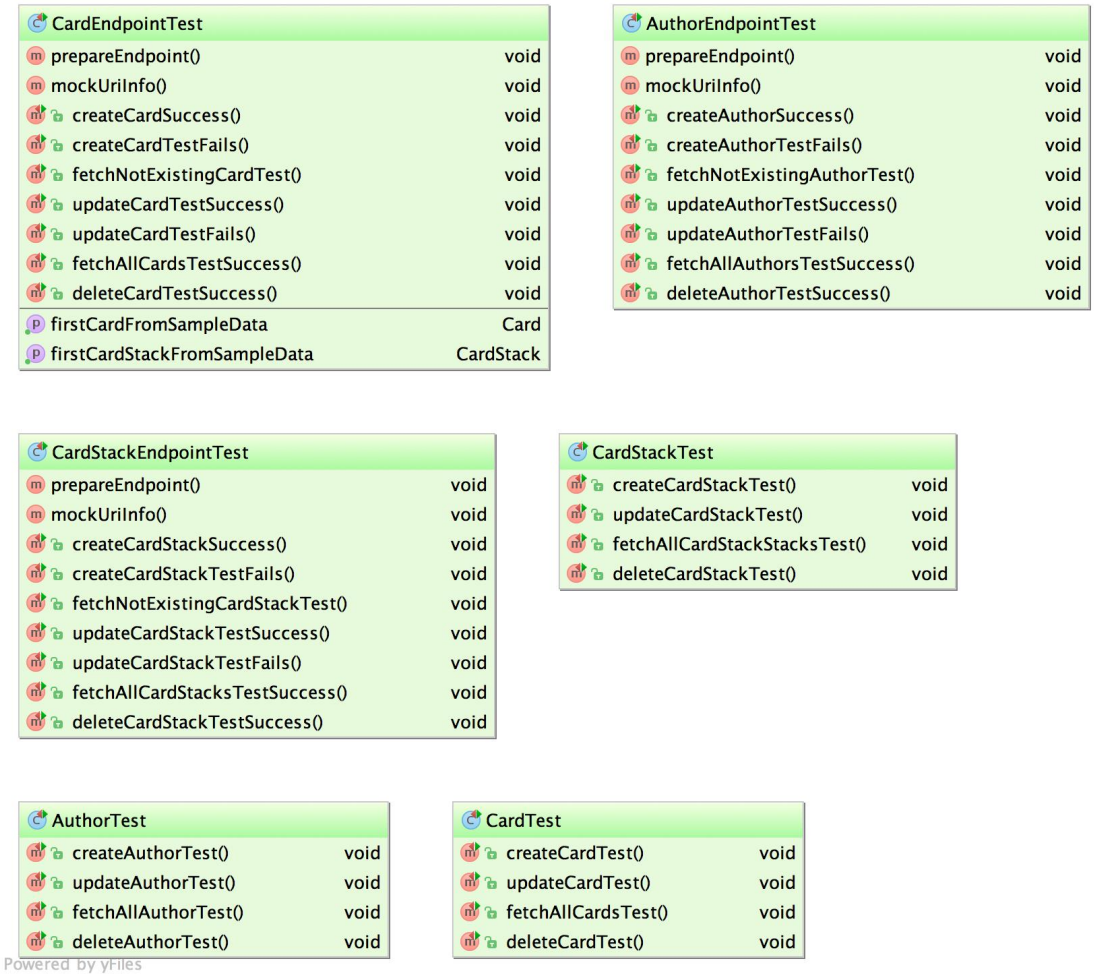


Abbildung A.1: Klassendiagramm - Package easylearnServices.Test

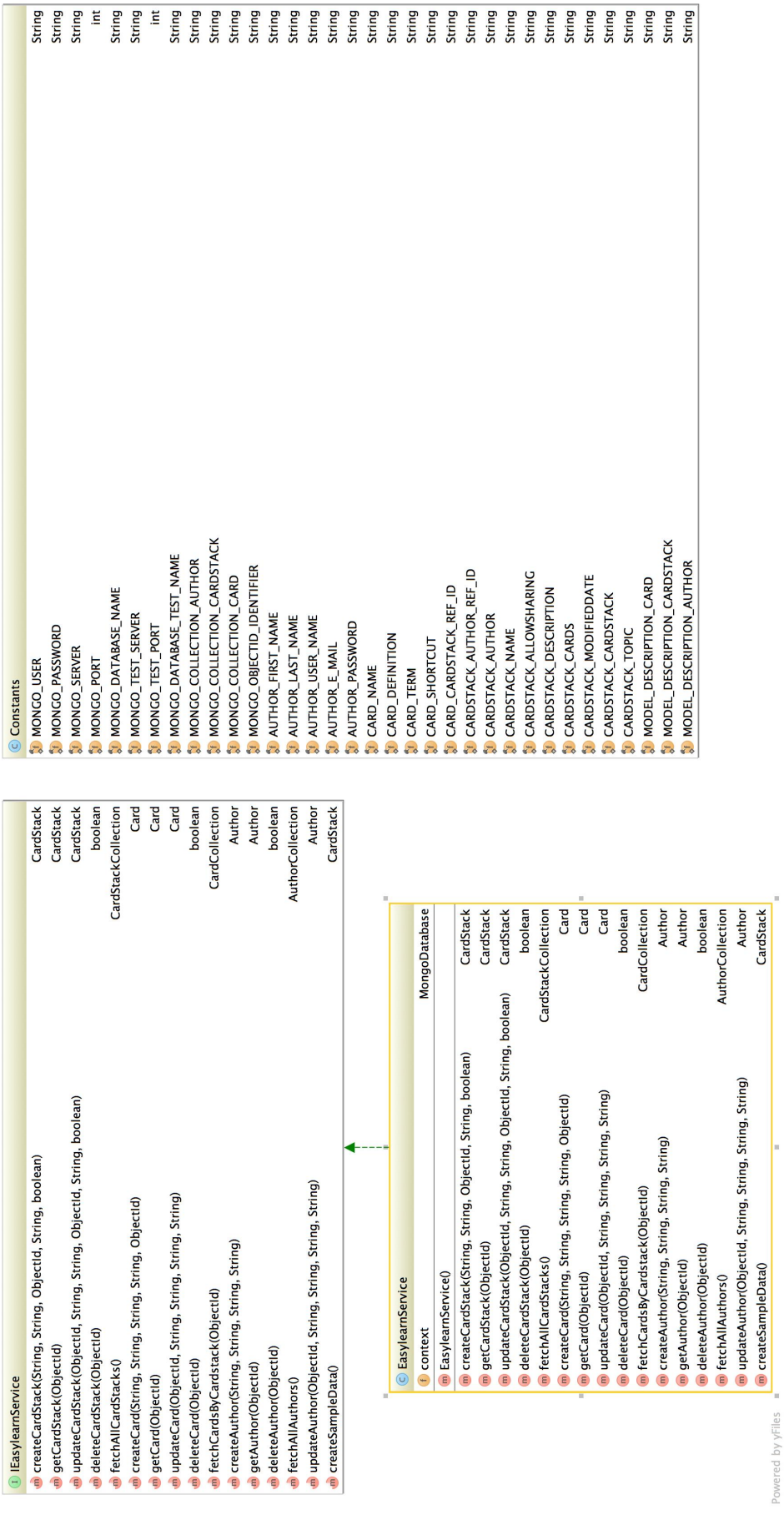


Abbildung A.2: Klassendiagramm - Package easylearnServices.Core.Common

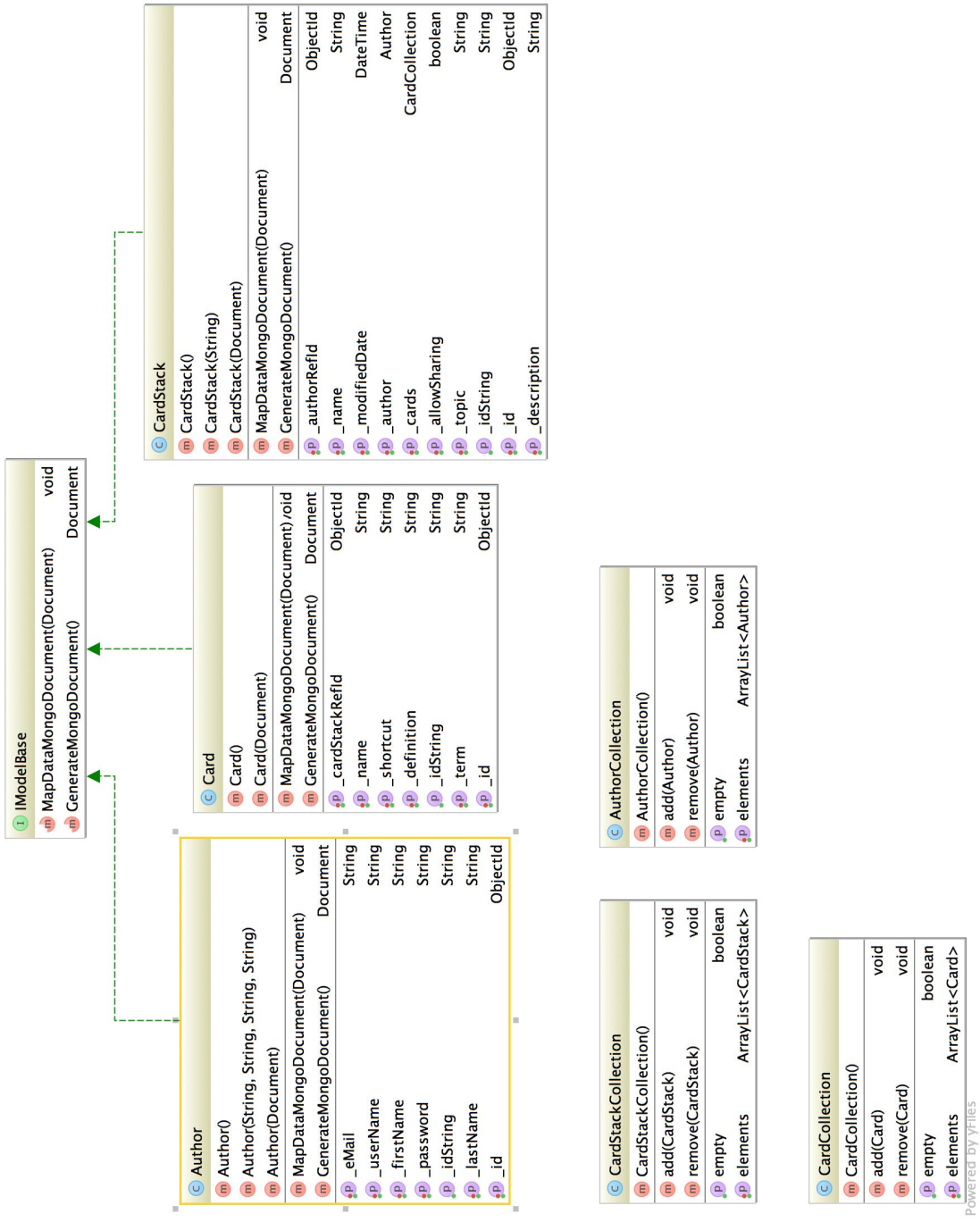


Abbildung A.3: Klassendiagramm - Package easylearnServices.Core.Models

Abbildung A.4: Klassendiagramm - Package easylearnServices.Rest

# Anhang B: Datenbank Diagramm

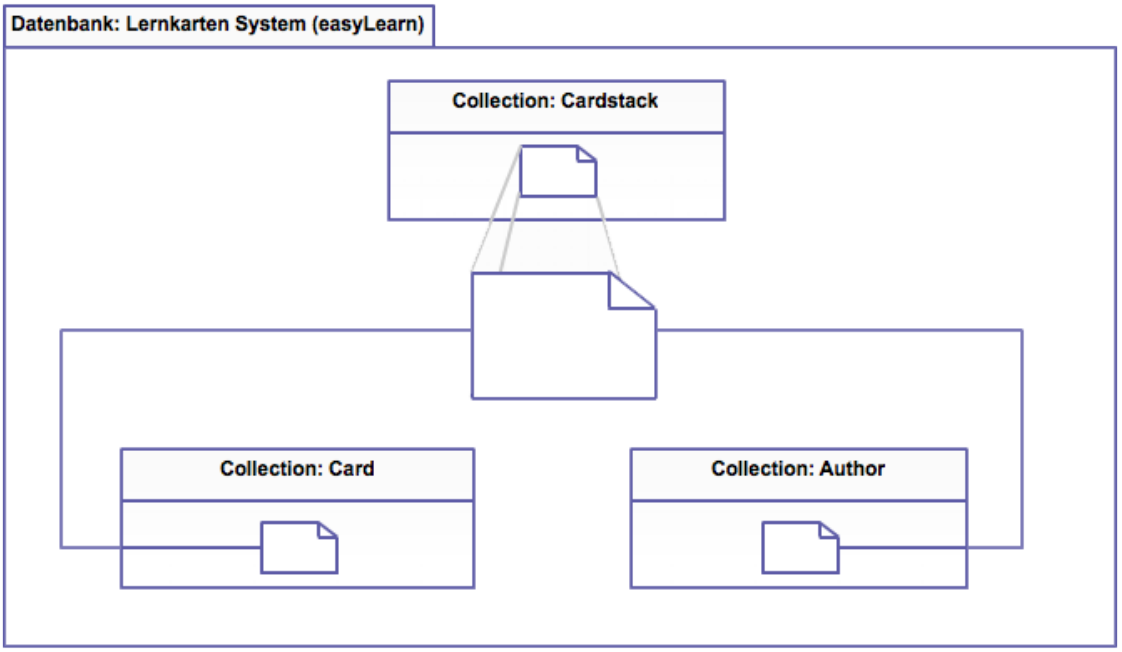


Abbildung B.1: Datenbank Diagramm - Übersicht

# Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 16. März 2015