Software-Architektur: Praktikumsaufgabe 2

ShareIt Teil I

Prof. Dr.-Ing. Axel Böttcher

1 Ausgangsbasis: Projektidee “ShareIt”

Dieser Abschnitt beschreibt eine Projektidee, die sich durch die kommenden Aufgaben hindurch ziehen wird. Die dabei entstehende Anwendung soll von Ihnen schrittweise erweitert werden. Die konkrete Aufgabenstellung für die kommenden drei Kalenderwochen folgt dann in Abschnitt 4.
Als Grundlage für das durchzuführende Systementwicklungsprojekt wird zunächst die Kontextsituation der realen Welt beschrieben, in welche diese Fallstudie eingebettet ist.
Danach kommt ein grober Überblick über die wesentlichen Fachdatenbestände und Nutzungsweisen des zu erstellenden Systems. Des Weiteren gibt eine erste Skizze für Schichten- und Komponentenstruktur die groben Randbedingungen für die Architektur des zu erstellenden Systems vor.
Ferner finden Sie diejenigen Anforderungen und Entwurfsentscheidungen, die von Ihnen in dieser ersten Iteration der Systementwicklung umzusetzen sind.

1.1 Problemstellung

Einige miteinander befreundete Studierende haben im Verlauf des ersten Studienjahres festgestellt, dass bei vielen Lehrveranstaltungen die angegebene Begleitliteratur hilfreich ist für den schnellen und intensiven Lernerfolg. Daher haben die Studierenden diejenigen Bücher, die nicht in der Bibliothek der Hochschule verfügbar sind, selbst erworben und diese Bücher anschließend durchgearbeitet, sowohl als Begleitlektüre als auch zur Vorbereitung auf die Prüfung.
Nachdem die Studierenden die Bücher durchgearbeitet und die Prüfung erfolgreich bestanden hatten, haben sie viele dieser Bücher im kommenden Semester nicht mehr benötigt. Stattdessen wurde für die neuen Fächer neue Begleitliteratur empfohlen.
Bald haben die Studierenden erkannt, dass die kontinuierliche Beschaffung der empfohlenen Begleitlektüre zwar inhaltlich sinnvoll wäre, diese Vorgehen jedoch aufgrund der zum Teil hohen Preise der Fachbücher sehr schnell das Budget der meisten Studierenden sprengen würde. Da viele der Bücher von den Studierenden ohnehin nur für einen relativ kurzen Zeitraum genutzt werden, entstand die Idee, die erworbenen Fachbücher im Kreis der befreundeten Kommilitonen auszutauschen. Dadurch erhalten alle beteiligten Studierenden Zugriff auf eine große Anzahl von Fachbüchern, ohne dass jede bzw. jeder Einzelne alle Bücher selbst erwerben muss.
In der Anfangsphase der gemeinschaftlichen Nutzung dieses Fundus an Fachbüchern haben die beteiligten Studierenden sich auf sehr informelle Weise darüber verständigt, wer welches Werk gerade benötigt, ausleihen möchte bzw. neu angeschafft hat und der Gemeinschaft zur Verfügung stellt. Mit zunehmender Anzahl der beteiligten Studierenden und der zu verwaltenden Bücher wurde diese informelle Abstimmung zunehmend unübersichtlich und ineffizient. Daraufhin haben die Studierenden beschlossen, die aktuellen Tausch- bzw. Verleihbeziehungen mit Hilfe eines geeigneten Softwaresystems zu verwalten – und in den Austausch den Bestand ihrer sonstigen Medien, wie CDs/DVDs/Bluerays, mit einzubeziehen.

1.2 Systemidee und Skizze der Anforderungen

Als Grundlage für die Entwicklung des Softwaresystems haben einige der beteiligten Studierenden gemeinsam Anforderungen an das System gesammelt und schriftlich dokumentiert. Dieses Anfoderungsdokument ist als erste Skizze der Anforderungen zu verstehen. Es dient als Diskussionsgrundlage für die systematische Anforderungsanalyse und den Entwurf. Entsprechend ist davon

auszugehen, dass die Anforderungsskizze noch unvollständig ist und möglicher Weise auch einige widersprüchliche Anforderungen enthält. Die gesammelten Anforderungen wurden bisher noch nicht priorisiert.

Arbeitsname Das zu erstellende System wird auf den Arbeitsnamen *ShareIt* getauft.

Benutzerverwaltung Studierende, die ShareIt nutzen wollen, müssen sich vorab als Benutzer registrieren. Geplant sind zwei Arten von registrierten Benutzern, nämlich Administratoren und "normale" Benutzer.
Ein registrierter Benutzer kann sich am System an- und abmelden. Ist ein Benutzer angemeldet, so kann er selbst die von ihm gespeicherten Stammdaten bearbeiten. Außerdem kann der Benutzer sich die von ihm zur Verfügung gestellten Leihgaben sowie die von ihm aktuell entliehenen Medien anzeigen lassen.
Zusätzlich zu den Funktionalitäten für "normale" Benutzer können Administratoren einen neu registrierten Benutzer freigen. Benutzer, die mehrfach gegen die selbstdefinierten Regeln der ShareIt-Community verstoßen haben, können von einem Administrator von der weiteren Nutzung des Sytems ausgeschlossen werden.

Medien-Verwaltung ShareIt verwaltet den Bestand der ausleihbaren Medien. Hierfür bietet es die Möglichkeit, Medien neu anzulegen bzw. zu löschen sowie den Datensatz eines bestehenden Mediums zu bearbeiten. Darüber hinaus ist es möglich, eine Liste der ausleihbaren Medien anzuzeigen oder nach einem bestimmten Medium zu suchen.
Bei der Medienverwaltung berücksichtigt ShareIt, dass von einer bestimmten Art von Medium mehrere inhaltlich gleichwertige Exemplare in Umlauf sein können. Beispielsweise können von einem bestimmten Buch mehrere Exemplare verfügbar sein. Das System soll die Basisinformationen zu diesen gleichwertigen Exemplaren so verwalten, dass der Datenbestand möglichst frei von Redundanzen ist und dass inhaltlich gleiche Exemplare auch als gleichwertig behandelt werden. Sucht ein Benutzer nach einem bestimmten Buch, von dem mehrere Exemplare in Umlauf sind, so kann diese Suche durch ein beliebiges Exemplar dieses Buches bedient werden.
Bei der Gestaltung des Systemes ist einzuplanen, dass mittelfristig nicht nur Bücher über ShareIt ausgetauscht werden können, sondern je nach Bedarf auch andere Arten von Medien (Beispielsweise DVDs, BlueRay, Audiobooks ...).

Verleih und Rückgabe Angemeldete Benutzer können Medien bzw. deren Instanzen über ShareIt ausleihen bzw. zurückgeben. ShareIt dokumentiert dabei, welches Medium gerade bei welchem Benutzer ist. Ein genaues Verfahren muss hierfür noch festgelegt werden.
Beispielsweise wäre es denkbar, dass ein Benutzer sich das gewünschte Medium zunächst über ShareIt reserviert und anschließend mit dem aktuellen Besitzer die Details der Übergabe des Mediums klärt. Nachdem das Medium den Besitzer gewechselt hat, wird das Medium aus dem Benutzerkonto des bisherigen Besitzers gelöscht und beim neuen Empfänger eingetragen.
Ist ein Medium gerade entliehen, so kann dessen Eigentümer dieses bei Bedarf vom aktuellen Entleiher zurückfordern.

Mobilität Die über ShareIt verbundenen Studierenden studieren nicht notwendigerweise alle am selben Ort. Darüber hinaus pendeln viele Studierende zwischen Heimat- und Studienort oder wohnen während eines Praxis- oder Auslandssemesters in einer anderen Stadt. Auch derartig mobile Studierenden sollen ShareIt nutzen bzw. entliehene Medien mitnehmen können. Um den unkomplizierten Austausch von Medien auch für mobile Studierende effizient zu halten, müssen entsprechende Informationen zum Aufenthaltsort der Studierenden bzw. der von diesen genutzten Medien in ShareIt verwaltet werden.
Damit das System universell verfügbar ist, soll es als Webapplikation realisiert werden.

2 Überblick über das Gesamtsystem

2.1 Ziele

Ziel der Verleihplattform ShareIt ist es, den wechselseitigen Verleih von Lehr- und Lernmaterialien innerhalb einer eingegrenzten Gruppe von Studierenden zu organisieren und zu verwalten. Dieses Gesamtziel lässt sich gliedern in die folgenden drei Hauptaufgabenbereiche:

- Verwaltung von Benutzerdaten
- Verwaltung von Exemplaren
- Ausleihe und Rückgabe der Exemplare

2.1.1 Zwingend umzusetzende Anforderungen

Die folgenden Funktionalitäten bzw. sonstigen Systemeigenschaften muss das zu erstellende Softwaresystem zwingend umsetzen.

- Registrieren eines neuen Benutzers
- Anmelden eines bereits registrierten Benutzers am System
- Abmelden eines angemeldeten Benutzers vom System
- Bereitstellen eines Exemplares
- Ausleihen eines Exemplares
- Zurückgeben eines entliehenen Exemplares

2.1.2 Wünschenswerte Anforderungen

Um die Nutzung der Anwendung komfortabler zu gestalten sind die folgenden Funktionalitäten hilfreich. Diese sind nicht zwingend erforderlich, sollten aber dennoch so weit wie möglich umgesetzt werden.

- Auflisten aller verfügbaren Exemplare
- Suchen nach einer bestimmten Art von Exemplar

2.1.3 Anforderungen an spätere Produktversionen

Die folgenden Funktionalitäten und Systemeigenschaften sind nicht im Fokus der aktuellen Systementwicklung, sollen jedoch aus heutiger Sicht mit hoher Wahrscheinlichkeit in zukünftig geplanten Versionen des Softwaresystems realisiert werden.

- Verwaltung der Aufenthaltsorte von Exemplaren, aktuellen Besitzern und Ausleihern, um ggf. eine unmittelbare persönliche Übergabe der Exemplare zu ermöglichen
- Einfache Erweiterbarkeit auf beliebige Arten von Medien

2.1.4 Ausgeschlossene Anforderungen

Die folgenden Eigenschaften sind explizit nicht vom System umzusetzen und auch aus heutiger Sicht für die zukünftige Weiterentwicklung des Systems nicht geplant.

- Die Einführung von Leihgebühren ist derzeit nicht geplant.

3 Architektur-und Design-Vorgaben

Um eine iterative Entwicklung des Softwaresystems zu ermöglichen und das System auch mittelfristig wartbar und erweiterbar zu erhalten, soll die logische Architektur der Software modular aufgebaut sein. Grundlage dafür ist eine sukzessive Gliederung in fachliche Domänen bzw. Subsysteme (Abschnitt 3.1) sowie in Schichten (Abschnitt 3.2).

3.1 Subsysteme

Bedeutung Subsysteme gliedern ein Softwaresystem vertikal nach fachlichen Gesichtspunkten. Dazu werden diejenigen Anwendungsfälle, die thematisch eng miteinander gekoppelt sind (in einer fachlichen Domäne liegen), in ein Subsystem gebündelt. Alternativ wäre es auch denkbar, das Softwaresystem auf der Grundlage des fachlichen Klassenmodells in Subsysteme zu gliedern.

Subsysteme in ShareIt Die vertikale Gliederung von ShareIt nach thematisch zusammenhängengen Anwendungsfällen ergibt zunächst die folgenden drei fachlichen Domänen:

- Benutzerverwaltung
- Verwaltung von Medien und Exemplaren
- Ausleihe und Rückgabe

3.2 Schichten

Durch eine Einteilung in Schichten wird das Softwaresystem horizontal gegliedert in verschiedene Abstraktionsebenen (vergleiche Abbildung 1). Jede Schicht ist dabei Dienstanbieter für die darüber liegende Schicht und nutzt selbst nur diejenigen Dienste der unmittelbar darunter liegenden Schicht. Die Grenze zwischen den Schichten ist so zu ziehen, dass möglichst wenige Abhängigkeiten über die Schichtgrenzen hinweg bestehen. Ein wesentliches Ziel der Gliederung in Schichten besteht darin, die

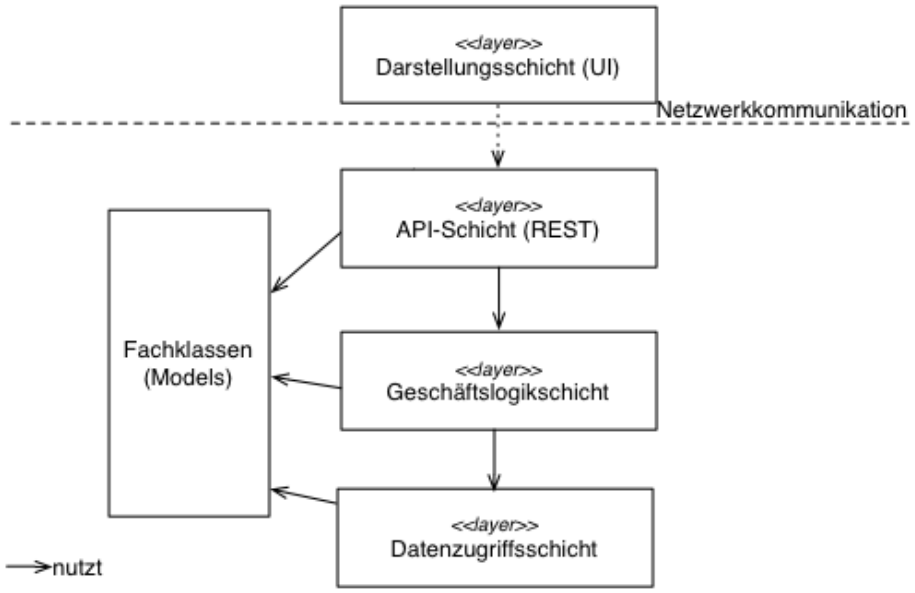

Abbildung 1: Typische Schichtenarchitektur einschließlich schichtenübergreifender Fachklassen.

Implementierung einer einzelnen Schicht auszutauschen. Die Schnittstellen dieser Schicht bleiben dabei unverändert.
Die Abhängigkeiten zwischen den Schichten werden über Schnittstellen gekapselt, sodass die Implementierung jeder einzelnen Schicht austauschbar ist.
Modelklassen (Fachklassen) dürfen von allen Schichten, also auch über Schichtengrenzen hinweg, verwendet werden.

Datenzugriffsschicht Die Datenzugriffsschicht (Persistence Layer) realisiert die Fachklassen sowie den Zugriff auf diejenige Technologie, die zur persistenten Datenhaltung für diese Fachklassen verwendet wird, beispielsweise eine relationale Datenbank. (Die Technologie selbst ist dabei nicht Bestandteil der Datenzugriffsschicht.) Ebenfalls in der Datenzugriffsschicht angesiedelt ist die Abbildung von Objekten auf Strukturen einer Datenbank. Das umfasst beispielsweise das OR-Mapping, also die Abbildung der Objektstruktur der Fachklassen auf relationale Datenbanktabellen, sofern eine relationale Datenbank eingesetzt wird.

Geschäftslogikschicht Auf der nächsten Abstraktionsebene liegt die Geschäftslogikschicht (Business Layer), die sich auf der Datenzugriffsschicht abstützt. Die Geschäftslogikschicht beinhaltet die grundlegende Geschäftslogik zur Verarbeitung der Fachklassen sowie eine Implementierung der Workflows bzw. Arbeitsabläufe des Systems.

API-Schicht Die API-Schicht stellt die Kommunikations-Endpunkte zur Verfügung für Mensch-Maschine (Browser-basiert) und Maschine-Maschine-Interaktionen. Hier werden insbesondere Objekte serialisiert und deserialisiert.

Darstellungsschicht Die Darstellungsschicht (Presentation Layer) kapselt die Benutzungsschnittstelle des Softwaresystems. Sie bildet für den Benutzer den Einstiegspunkt in die Bedienung des Softwaresystems. Ausgehend von den Aktionen des Benutzers auf dieser Benutzungsschnittstelle wie z.B. der Auswahl eines bestimmten Anwendungsfalles werden die dafür erforderlichen Funktionalitäten der API-Schicht angestoßen. Anfallende Ergebnisdaten werden von der API-Schicht an die Darstellungsschicht übergeben, von dieser in geeigneter Form aufbereitet und anschließend dargestellt.
Das Softwaresystem ShareIt wird ebenfalls gemäß dieser Konvention als Mehrschichtarchitektur realisiert (siehe Abbildung 1).

4 Ihre Aufgabenstellung, ShareIt Teil I

4.1 Lernziele dieser Aufgabe

- Sie definieren ein einfaches REST-API
- Sie definieren eine eigene Vorgehensweise für die Behandlung von Fehlern in einem REST-API
- Sie implementieren und testen einen einfachen REST-Service auf Basis verschiedener gegebener Java-Bibliotheken
- Sie deployen eine Web-Applikation bei einem PaaS-Provider

4.2 Vorbereitung

Zur Bearbeitung der Aufgabe benötigen Sie einen github-Account. Ferner müssen Sie sich für den Kurs registrieren. Dazu können Sie diesen Link verwenden. Letzter Commit bis spätestens 30.4.2017, 23:59Uhr. Denken Sie daran, dass ich je Aufgabe einen individuellen Commit je Teilnehmender Person fordere!
Sie finden in Ihrem Repository zu dieser Aufgabe ein vorbereitetes Maven-Projekt. Dieses Projekt basiert auf dem Maven Archetyp `webapp`. Lassen Sie sich von Maven die Metadaten für Ihre IDE erzeugen (z.B. `mvn eclipse:eclipse`, `maven idea:idea` oder mit entsprechendem IDE-Plugin).
Die in der Bezeichnung der bereit gestellten Packages enthaltenen Domain-Namen dürfen Sie nach Ihren eigenen Vorstellungen anpassen.
Zum Ausführen muss das Projekt eigentlich auf einem Servlet-Container deployed werden (siehe Abschnitt 4.4). Für die lokale Ausführung verwenden wir Jetty, der sich ohne Installation einfach aus einem `jar` heraus starten lässt. Dazu dient die Klasse `JettyStarter`, die eine entsprechende `main`-Methode bereit stellt. Sie können den Jetty-Starter sofort ausführen und sollten lokal eine vorgegebene einfache Seite zur Eintragung von Büchern angezeigt bekommen.

4.3 Implementierung und Test

Implementieren Sie das Subsystem (die Domäne) “Medien- und Leihobjekte bereitstellen”. Da die Benutzerverwaltung und damit die Prüfung, ob ein Benutzer angemeldet ist und die entsprechenden

Rechte hat, später implementiert werden wird, brauchen Sie darauf bei **dieser** Aufgabe noch keine Rücksicht zu nehmen. Jeder, der die URI der Anwendung kennt, darf also alles.
Achten Sie wieder auf eine Testabdeckung von mindestens 85% und testen Sie immer auch die möglichen Fehlerfälle mit ab.

4.3.1 Schichten

Die Anwendung wird als Schichten-Architektur mit einem REST-API zwischen Web-Browser und Server-Infrastruktur implementiert (siehe Abbildung 1).
Jede Schicht erhält ein eigenes Java-Package. Sie darf nur Dienste der unmittelbar darunter liegenden Schicht nutzen; an der Schnittstelle befindet sich stets ein Java-Interface.
Eine Benutzeroberfläche (User Interface, kurz UI) wird heute sehr oft mit JavaScript realisiert. Wenn überhaupt vorhanden, dann halten wir das in diesem Praktikum sehr sehr einfach und nutzen dafür keine komplexen Frameworks. Ein einfaches Beispiel finden Sie in dem vorgegebenen Projekt auf Github. Dieses Beispiel dient nur und ausschließlich zur Demonstration; es ist weder schön implementiert noch ausgiebig getestet!

4.3.2 Fachklassen

Die erforderlichen Fachklassen (Models) sehen Sie im Diagramm in Abbildung 2. Die Klasse `Medium` ist eine abstrakte Darstellung eines Mediums (Book oder Disc). Diese Darstellung muss existieren, bevor ein konkretes Exemplar (`Copy`) dieses Mediums eingestellt werden kann. Jedes Objekt der Klasse `Book` enthält eine valide und systemweit eindeutige ISBN-13; jedes Objekt der Klasse `Disc` enthält einen systemweit eindeutigen Barcode.
Ein Benutzer kann dann beliebig viele Exemplare ein und desselben Mediums einstellen, die sich in seinem Eigentum befinden. Ein Benutzer wird dabei über einen systemweit eindeutigen Namen identifiziert.
Diese Klassen sind Standard-Java-Klassen, ohne Annotationen, die Sie einfach implementieren können. Die Prüfung der Gültigkeit von ISBN und Barcode wird innerhalb dieser Klassen nicht geprüft.

4.3.3 Resourcenschicht und REST-Schnittstelle

An der REST-Schnittstelle soll zunächst nur das API für die Medien zur Verfügung gestellt werden[1]:

[1] Bei Bedarf können Sie weitere mögliche Fehler aufnehmen

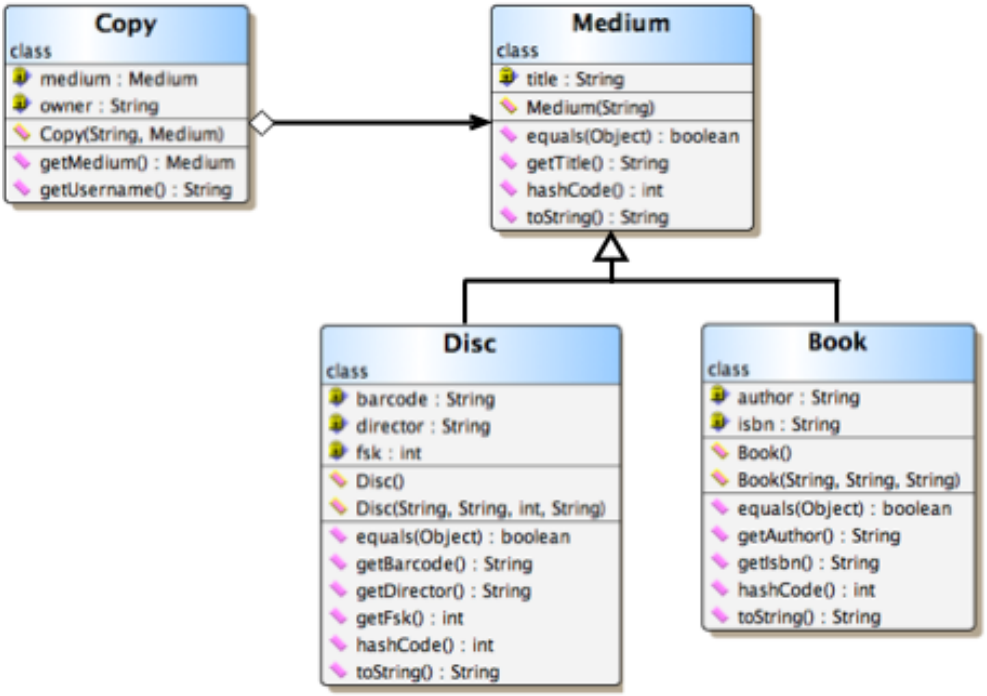

Abbildung 2: Die Modellklassen für die Domäne "Medien- und Leihobjekte bereitstellen". Alle Klassen-Diagramme wurden mit dem ydoc Javadoc-Plugin erstellt.

URI-Template	Verb	Wirkung
`/media/books`	POST	Neues Medium Buch anlegen Möglicher Fehler: Ungültige ISBN Möglicher Fehler: ISBN bereits vorhanden Möglicher Fehler: Autor oder Titel fehlt
`/media/books/{isbn}`	GET	Eine JSON-Repräsentation eines gespeicherten Buches liefern, falls vorhanden
`/media/books`	GET	Alle Bücher auflisten
`/media/books/{isbn}`	PUT	Daten zu vorhandenem Buch modifizieren (JSON-Daten enthalten nur die zu modifizierenden Attribute) Möglicher Fehler: ISBN nicht gefunden Möglicher Fehler: ISBN soll modifiziert werden (also die JSON-Daten enthalten eine andere ISBN als die URI) Möglicher Fehler: Autor und Titel fehlen
`/media/discs`	POST	analog zu oben
`/media/discs`	GET	analog zu oben
`/media/discs/{barcode}`	GET	analog zu oben
`/media/discs/{barcode}`	PUT	analog zu oben

Implementieren und testen Sie eine Resourcen-Klasse `MediaResource`, sowie die dazugehörige

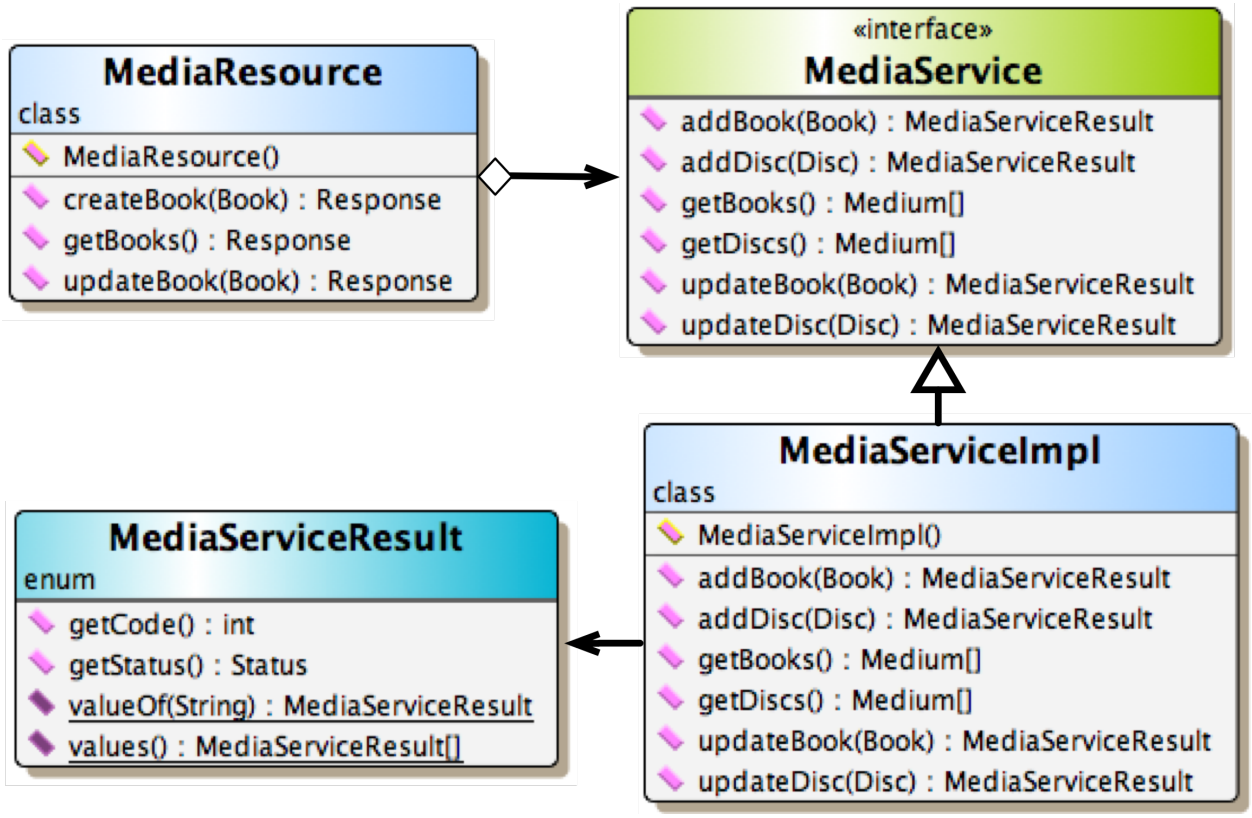

Abbildung 3: Mögliche Klassenstruktur für Resurcen und Services.

Geschäftslogik in einer Service-Klasse zur Verwaltung der Medien (mit einem entsprechenden Interface dazwischen).
Die Resourcenklasse nimmt Pfadparameter und JSON entgegen und leitet de-serialisierte Objekte an den Service weiter. Ferner serialisiert sie Ergebnisse von Anfragen. Erst im Service wird geprüft, ob die Objekte vollständig und valide sind (z.B. gültige ISBN bzw. Barcode).
Die Implementierung der Datenzugriffsschicht folgt in der nächsten Praktikumsaufgabe. Sie können in dieser Aufgabe die Medien-Daten in geeigneten Objekten des Java Collection-API im Service "speichern".
In Abbildung 3 finden Sie eine Skizze der beteiligten Klassen einschließlich des `enum`Typs `MediaSrviceresult` um Fehler vom Service zur resourcen-Klasse transportieren zu können, falls erforderlich. Sie müssen sich aber nicht zwingend an dieses Diagramm halten.

4.3.4 Response-Objekte und Fehlerbehandlung

Wenn kein Fehler auftritt, dann reicht es aus, wenn die aufgerufene Methode einer Resourcen-Klasse den entsprechenden 2xx-Code zurück gibt. Tritt ein Fehler auf, so muss ein angemessener HTTP-Fehlercode an den Client geliefert werden und sinnvollerweise ein Hinweis auf die Fehlerursache. Liefern Sie in diesem Fall ein JSON-Objekt mit denn Attributen `code` (Zahl) und `detail` (Zeichenkette). Der Fehlercode wird dann also redundant übertragen.

4.4 Deployment

Machen Sie ein Deployment Ihrer Anwendung auf Heroku. Sie brauchen dazu einen Heroku-Account. Die Gefahr ist zwar gering, aber passen Sie dabei bitte auf, dass Sie nicht aus Versehen kostenpflichtige Services buchen — hinterlegen Sie insbesondere keine Kreditkareninformationen!
Sobald Sie den Account erzeugt haben, müssen Sie noch gleich eine `app` anlegen.
Sie benötigen ferner das Heroku command line interface (cli). Dieses können Sie leider nur auf Ihren eigenen Rechnern installieren. Deshalb können Sie diesen Aufgabenteil nicht bearbeiten, wenn Sie auf die Laborrechner angewiesen sind.
Es gibt dann mehrere Möglichkeiten, das Heroku-Deployment durchzuführen:

1. Mittels `git`
   Dazu müssen Sie Ihre Heroku-app als so genanntes remote in `.git/config` eintragen:

   ```
   [remote "heroku"]
       url = https://git.heroku.com/IHRE-APP.git
       fetch = +refs/heads/*:refs/remotes/heroku/*
   ```

   Der Nachteil dieser Lösung ist, dass Sie ein generiertes Produkt, nämlich das `.war`-File, in Ihr Repository committen müssen:

   `mvn package` ausführen, das Archiv committen und das Repo pushen:

   `git push heroku master`

2. Mit maven: `mvn heroku:deploy-war`. Dieser Aufruf integriert das Packen sowie das Deployment. Sie müssen dazu im `pom.xml` den Namen Ihrer Heroku-App im Abschnitt zum `heroku-maven-plugin` eintragen. Vorteil: volle Maven-Integration.

3. Mit dem Heroku-cli: `heroku war:deploy target/<name>.war --app IHRE-APP`.
   Nachteil: nicht in Maven integriert

Wenn Sie das geschafft haben, dann haben Sie Ihren ersten Micro-Service gebaut und deployed. Die Anwendung steht nun unter `https://IHRE-APP.herokuapp.com/` zur Verfügung.
Hinterlassen Sie bitte den Link zu Ihrer Anwendung bei Heroku in einer Datei namens `Readme.md` im Projekt-Root.

5 Erste Erweiterung: Anlegen von Exemplaren

Entwerfen und implementieren Sie einen REST-Endpoint für das Anlegen von Exemplaren. Dazu benötigen Sie wieder eine Resourcen-Klasse und einen Service.
Zum Eintragen muss eine ISBN/BArcode und ein Benutzername angegeben werden. Geprüft werden muss lediglich, ob das Medium zur gegebenen ISBN/Barcode vorhanden ist.