Software-Architektur

Sommersemester 2017
Hartl, Sebastian
shartl@hm.edu

Einführung und Ziele

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/Blu-rays, mit einzubeziehen.

Aufgabenstellung

Inhalt.

Es soll ein Programm implementiert werden, mit dem Lehrmedien von Studenten angelegt und ausgeliehen werden können. Es soll zwischen Büchern und Discs unterschieden werden.

Von einem Medium sollen beliebig viele Kopien angelegt werden können. Der Besitzer einer Kopie soll sein Exemplar nach der Ausleihe auch wieder zurückfordern können, falls er es wieder benötigt.

Das Programm soll eine Benutzerverwaltung haben, mit verschiedenen Rollen.

Der Besitzer einer Kopie soll immer kontrollieren können, welcher Benutzer sich sein Medium ausgeliehen hat und wo sich der Ausleiher bzw. seine Kopie aktuell befinden.

Durch die Anwendung sollen sich Studenten Geld sparen können, indem sie nicht mehr selber Lehrmedien kaufen müssen. Diese Lehrmedien werden nach erfolgreichem Bestehen der Prüfung nicht mehr benötigt.

Durch die Möglichkeit, nicht mehr benötigte Medien verleihen zu können und ausleihen zu können, würden sich viele Studenten die teure Beschaffung der Medien ersparen.

Qualitätsziele

Inhalt.

1. Das System soll online sein und jeder Zeit verfügbar
2. Das Anlegen von Medien soll funktionieren
3. Ein angemeldeter Benutzer soll sich an- und ausloggen können
4. Das System soll nach dem Schichtenmodell implementiert werden (REST, Logic, Persistence)
5. Die Software muss auf dem PaaS Anbieter Heroku deploybar sein
6. Das System soll als Microservice deployed werden
7. Die Programmiersprache soll Java sein

Motivation.

Ein System für Studenten zum Ausleihen von Lehrmedien, ist vor allem vor der Prüfungszeit und am Wochenende wichtig, wenn eingeschränkter Zugang zu der Bibliothek der Universität besteht oder das Medium bereits vergriffen ist. Wenn dann die Applikation

nur an bestimmten Tagen zur Verfügung steht oder nur im Netz der Universität, besteht für die Studenten kein Mehrwert.

Genauso sollte das Anlegen der Medien und deren Kopien im Vordergrund stehen. Ansonsten kann natürlich nicht ver- und entliehen werden.

Die Sicherheit steht auch im Fokus und um entliehene Medien nachverfolgen zu können, ist es notwendig, dass nur registrierte Benutzer Zugriff auf das System bekommen und mit ihm interagieren können.

Um zu einem späteren Zeitpunkt der Entwicklung leicht innerhalb des Programmes Teile austauschen zu können, um z.B. auf eine andere Datenbank bzw. ein anderes Persistierungsframework zu wechseln. Dafür bietet sich das Schichtenmodell an, da gegen Schnittstellen programmiert wird.

Ebenso erleichtert das entwickeln als Microservice das Deployment, da so jedes Teil des Systems separat deployed wird und so die Ausfallzeiten möglichst gering sind. Wichtig hierbei ist allerdings, dass eine geeignete Fehlerbehandlung implementiert wird, um während einem Deployment keine Inkonsistenz der Daten zu riskieren und die Anwendung trotz kurzweiligem Ausfall mancher Subsysteme online und verfügbar ist.

Die Wahl der Programmsprache viel auf Java, da diese Sprache an Berufs- und Hochschulen gelehrt wird und niedrige Semester dann bereits in die Weiterentwicklung eingebunden werden können.

Stakeholder

Rolle	Beschreibung	Abnahmerelevanz	Erwartungen	Klassifizierung
Prof. Dr. Axel Böttcher	Professor, Fachgebiet der Softwaretechnik und Rechnerarchitektur	Hoch, begutachtet und bewertet den Code, gibt Tipps für die Entwicklung	Software soll gut programmiert und dokumentiert sein	Intensiv betreuen
Studenten von ShareIt	Studenten, hatten die Ursprüngliche Idee der Software	Hoch, haben konkrete Vorstellungen was die Software am Ende leisten soll	Funktionen sollen implementiert und funktionstüchtig sein	Intensiv betreuen
Nachfolgende Studenten	Studenten, die zu einem späteren Zeitpunkt ins Projekt einsteigen	Niedrig, arbeiten sich mit der Codedokumentation und diesem Dokument ein	Gut dokumentierter Code, aussagekräftige Methoden- und Variablennamen und eine gute Architektur-dokumentation	Informieren und involvieren

Randbedingungen

Die Software wird nach CC-Lizenz veröffentlicht. Die Rahmenbedingungen der Lizenz sind:

- Die Namen der Urheber müssen genannt werden
- Die Software darf nicht kommerziell verwendet werden
- Bei Bearbeitung oder Veränderung der Software muss diese unter den gleichen Bedingungen weitergegeben werden

Die Programmiersprache ist Java Version 8.

Im Code muss vor dem einchecken in das Sourcecode Repository das Tool Checkstyle durchlaufen, ohne Fehler anzuzeigen. Dies hat den Vorteil, dass der Code von allen Entwicklern Standardisiert ist. Die Konfiguration des Checkstyle ist über Moodle abrufbar. Sie beinhaltet, dass Methoden ein JavaDoc besitzen müssen, die Formatierung des Codes einheitlich ist und dass es keine „magischen“ Zahlen im Code gibt (hartgecodete Zahlen müssen als Klassenvariable deklariert und aussagekräftig benannt werden).

Es müssen, da wo es möglich ist, Unit Tests geschrieben werden, um die einzelnen Komponenten und Klassen auf ihre korrekte Funktionsweise zu testen. Integrationstest sollen von den Entwicklern auf eigene Verantwortung selbst durchgeführt werden und werden (zum jetzigen Zeitpunkt) noch nicht zentralisiert verwaltet. Allerdings halten sich die Stakeholder vor, selber Integrationstest zu verwenden, um Abnahmen besser bewerten zu können.

Die Software muss auf Heroku deployed werden können. Dieser Dienst wurde ausgewählt, da der Standard Account kostenfrei ist und bei Bedarf nach oben skaliert werden kann.

Um neuen Programmieren im Team die Einarbeitung leicht zu machen, ist darauf zu achten, dass Methoden, Variablen und Parameter aussagekräftige Namen bekommen. Es gilt das Single-Responsibility-Prinzip zu beachten, soweit möglich. So wird die Lesbar- und Wartbarkeit des Codes erhöht und Redundanzen im Code können womöglich vermieden werden.

Kontextabgrenzung

Durch die Entwicklung als Mikroservice sind derzeit zwei verschiedene Module zu betrachten. Authentication- und MediaService. Der Service für die Ausleihe und erstellen der Kopien ist derzeit in der Planungsphase und muss zu einem späteren Zeitpunkt hier noch ergänzt werden.

Die Kommunikation intern sowie extern erfolgt über REST Schnittstellen.

AuthenticationService

Zuständig für die Benutzer Authentifizierung sowie die Autorisierung. Hält als einziges Modul die Anmeldedaten der Benutzer vor und kennt auch als einziges System die Benutzerrolle.

Methode	HTTP Methode	Beschreibung	Mögliche Fehler
/login	POST	Überprüft die übergebenen Anmeldedaten des Benutzers und gibt einen Token zurück	Anmeldedaten falsch; Benutzer nicht im System
/logout	POST	Meldet einen Benutzer vom System ab und löscht den Token.	--
/validate	POST	Validiert ob der übergebene Token die Methode ausführen darf. Es wird der Token und der Methodennamen übergeben.	Token nicht (mehr) im System gespeichert; Token nicht valide; Methode für Benutzer nicht erlaubt

MediaService

Modul ist zuständig für das Anlegen neuer Medien, sowie der Suche und Update bestehender. Zur Authentifizierung des Benutzers soll ein valider Token als Header Parameter im den jeweiligen Requests mitgegeben werden.

Methode	HTTP Methode	Beschreibung	Mögliche Fehler
/books	GET	Suche nach allen gespeicherten Büchern	--
/books	POST	Speichert ein neues Buch in der Datenbank ab	Titel, ISBN oder Author fehlt; ISBN existiert bereits oder ist nicht valide

/books/{isbn}	GET	Suche nach einem Buch mit bestimmter ISBN	--
/books/{isbn}	PUT	Updated ein bestimmtes Buch	Titel, ISBN oder Author fehlt; ISBN soll geändert werden
/discs	GET	Suche nach allen gespeicherten Discs	--
/discs	POST	Speichert eine neue Disc in der Datenbank	Titel, Barcode, oder Direktor fehlt; Barcode existiert bereits oder ist nicht valide
/discs/{barcode}	GET	Sucht eine bestimmte Disc mit dem Barcode	--
/discs/{barcode}	PUT	Updated eine Disc	Titel, Barcode, oder Direktor fehlt; Barcode soll geändert werden

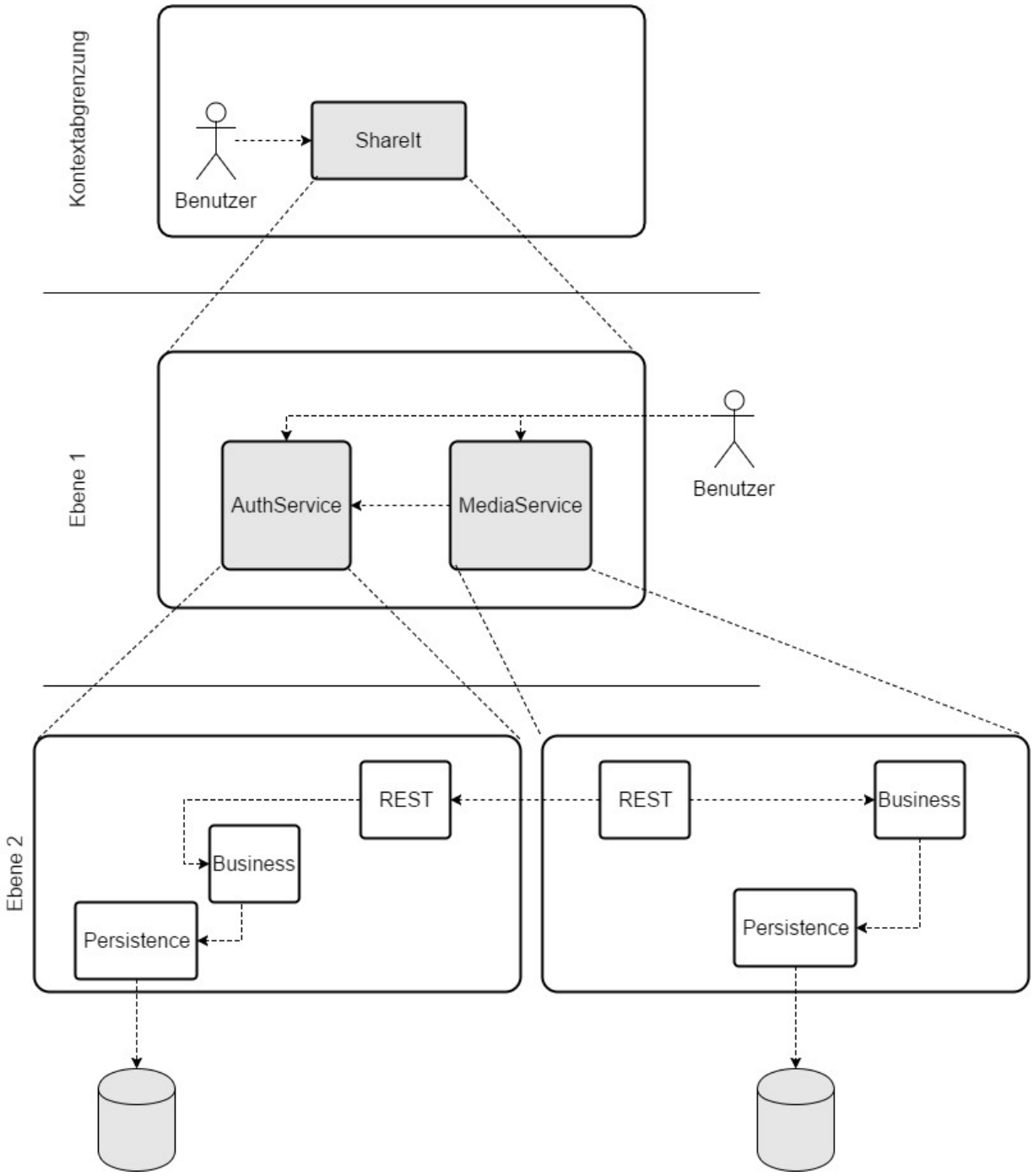
Lösungsstrategie

Hier werden die Grundlegenden Entscheidungen und Lösungsansätze beschrieben, die den Entwurf und die Implementierung des Systems prägen:

Aufgabe / Problem	Lösungsansatz
Leichte Wartbarkeit und Austauschbarkeit von Programmteilen	Implementierung als Schichtenmodell. Jede Schicht (REST, Logik, Persistierung) kennt nur die Schicht, die direkt unter ihr selber liegt. Jede Schicht wird als Interface definiert und gegen diese wird programmiert. Durch den Einsatz von CDI entfällt zudem die Auswahl der zu initialisierenden Klasse auf die JVM. Diese Klassen werden zentral konfiguriert.
Einzelne Module sollen leicht und unabhängig voneinander geupdated werden können	Durch die Implementierung als Microservice sind die einzelnen Programmteile verteilt auf dem Server deployed. Wenn sich Code in einem Modul ändert, muss nur dieses neu deployed werden und alle anderen können online bleiben und sind weiterhin (zumindest teilweise) funktionsfähig. Für die interne Kommunikation zwischen den einzelnen Modulen werden REST Schnittstellen verwendet.

Daten sollen dauerhaft persistiert werden	Es wird eine Relationale Datenbank verwendet um die Daten dauerhaft zu persistieren. Jedes Modul hat für seine Daten eine eigene Datenbank. Es gilt zu beachten, dass Daten so wenig wie möglich bzw. gar nicht redundant gespeichert werden sollen.
Die Schnittstellen sollen funktionieren und das erwartete Ergebnis zurückliefern	Für interne Komponenten/Klassentests sind Unit Tests zu verwenden. Für den Test, ob das gesamte System, wie geplant funktioniert sind Integrationstests durchzuführen. Diese können entweder automatisch oder manuell durchgeführt werden.

Bausteinsicht



Whitebox Gesamtsystem

Der Benutzer „sieht“ eine Schnittstelle mit der er interagieren kann. Diese Schnittstelle dient für die Kommunikation des Systems in die Außenwelt und stellt alle Services zur Verfügung, die ein Benutzer benötigt um sich zu authentifizieren zu können und um Medien anzulegen, ändern und suchen zu können.

Durch diese zusätzliche Schicht bleiben interne Arbeitsabläufe für den Benutzer unsichtbar und alle Module sind über eine URL ansprechbar. Weiter kann in dieser Schicht bereits die Fehlerbehandlung im Falle eines Modulausfalls implementiert werden. So könnten zum Beispiel für den Fall, dass der AuthentifizierungService ausfällt, die abgerufenen Token für eine gewisse Zeit gecached werden.

Ebene 1

Diese Ebene enthält die aktuell zwei Module und beschreibt die eigentliche Interaktion zwischen den Modulen.

Ein Benutzer muss sich zuerst über den AuthService anmelden um mit dem MediaService arbeiten zu können.

Der MediaService übergibt den vom Benutzer im Request mitgegebenen Token um zu verifizieren, ob dieser die beabsichtigte Methode ausführen darf.

So hat der MediaService keinerlei Informationen über den Benutzer und seine Rechte und frägt lediglich beim AuthService nach Erlaubnis um die Methode ausführen zu dürfen.

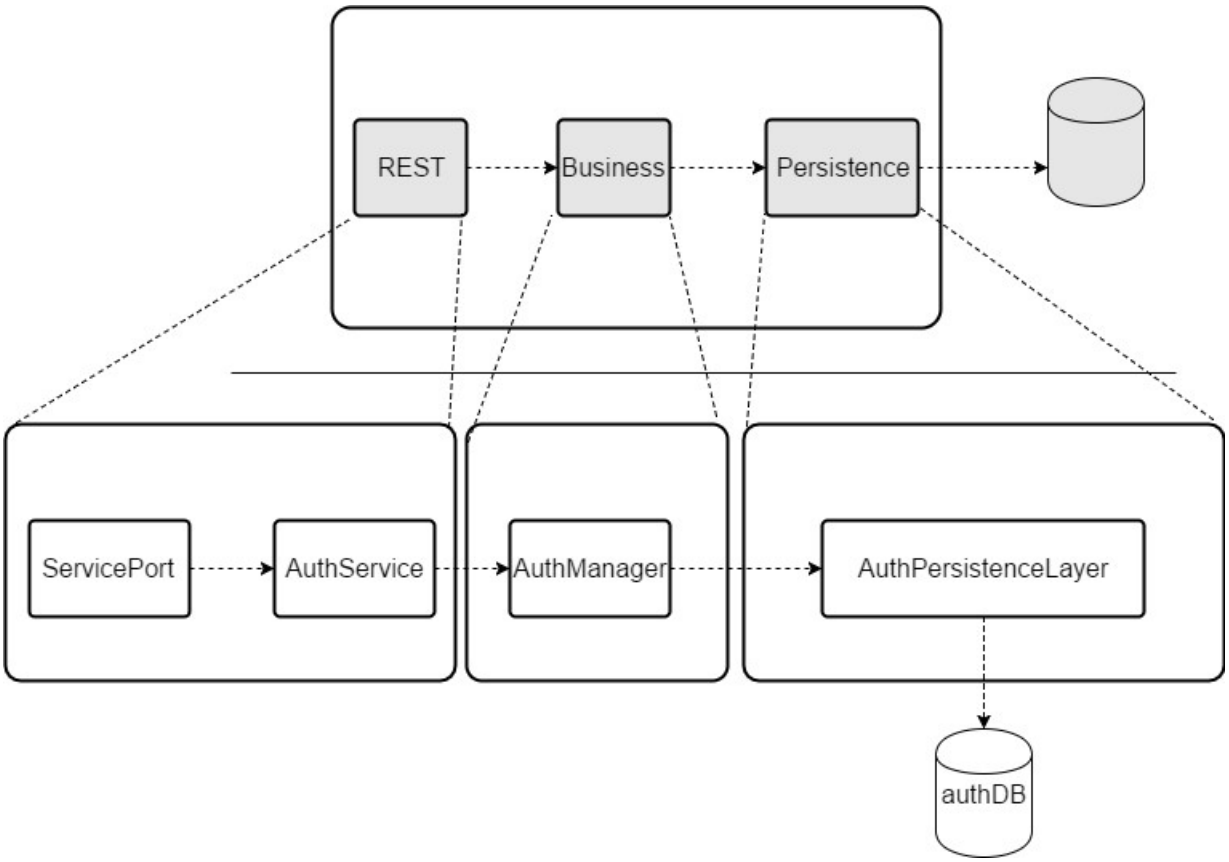
Ebene 2

Hier werden die einzelnen Module und ihre internen Abläufe aufgezeigt. In ihnen läuft die eigentliche Programmlogik ab. Jedes Modul hat seine eigene Datenbank und speichert auch nur die für ihn benötigten Daten ab.

Whitebox AuthService

Im AuthentifizierungService werden die Benutzer an- und abgemeldet vom System und es erfolgt die Autorisierung für Methoden.

Die Kommunikation nach außen erfolgt über eine REST-Schnittstelle mit drei definierten Methoden. Für die Kommunikation intern wird gegen das Interface der jeweilig niedrigeren Schicht implementiert.
So ist sind der Schicht REST nur die Methoden der Schicht Business bekannt und der wiederrum nur die Methoden der Schicht Persistence.

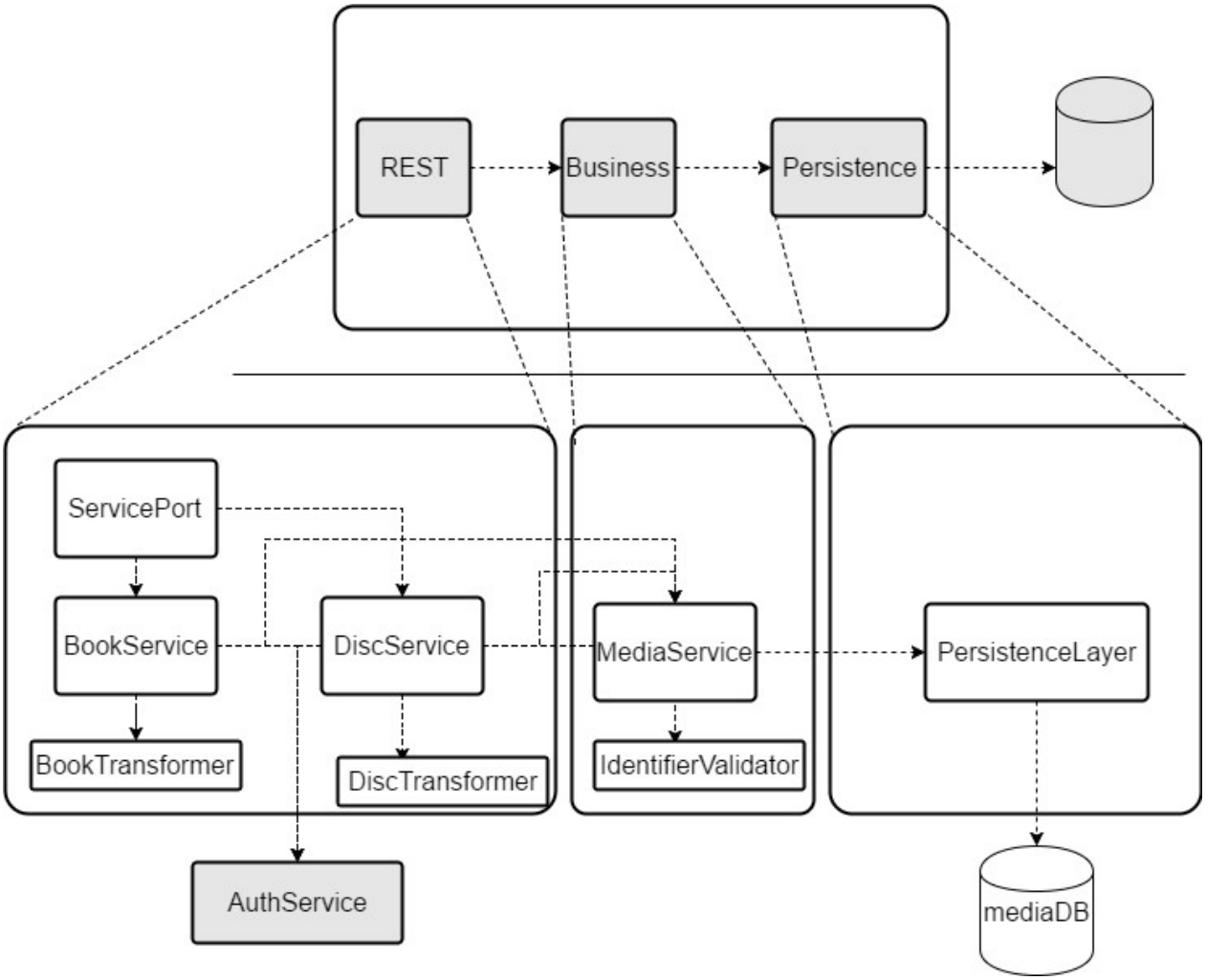


Whitebox MediaService

In diesem Modul findet die Verwaltung der Medien und ihrer Informationen statt. Es können neue Bücher und Discs angelegt, verändert und gesucht werden. Für die Autorisierung kommuniziert dieses Modul mit dem Authentifizierungsmodul.

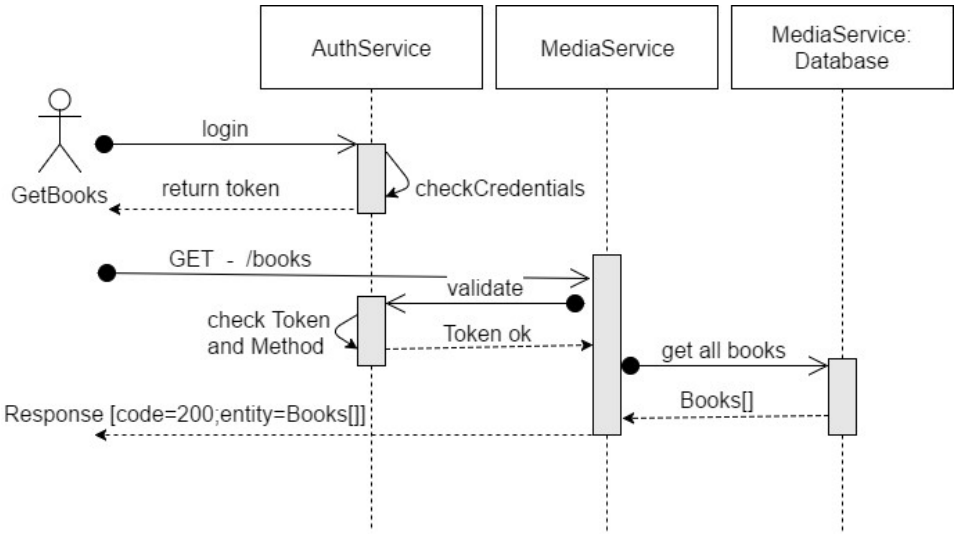
Die Logik für Bücher und Discs ist auf Schicht der REST Schnittstelle in zwei separate Klassen geteilt um die Lesbarkeit zu verbessern und um dem Single-Responsibility-Prinzip zu folgen. In den zwei Klassen wird der MediaService für die Backend Arbeit, der Autorisierungsservice für die Autorisierung des Benutzers und der Methode und am Ende der Response zusammengebaut, mit dem richtigen Errorcode und -message bzw. im Erfolgsfall das vom MediaService zurückgegebene Objekt.

Die Validierung der ISBN und der Barcodes erfolgt im MediaService in der Businesslogik mittels einer eigenen Klasse *IdentifierValidator*.


REST
Business
Persistence
ServicePort
BookService
DiscService
MediaService
PersistenceLayer
BookTransformer
DiscTransformer
IdentifierValidator
AuthService
mediaDB

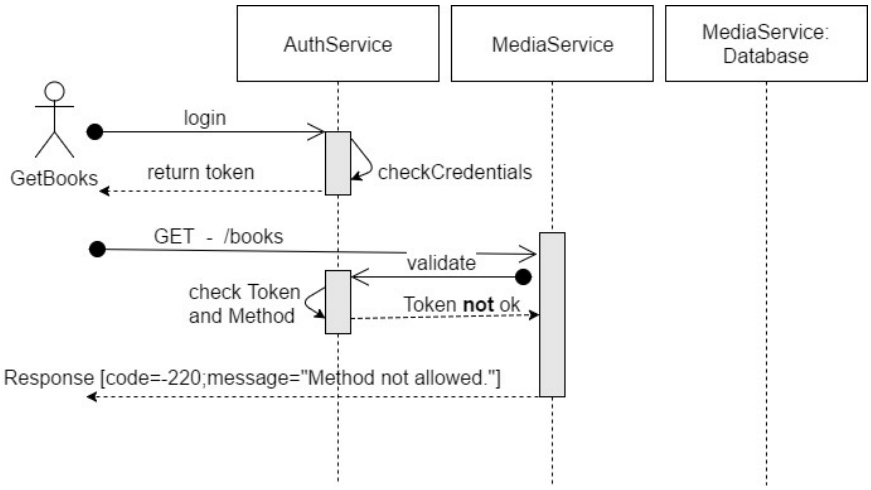
Laufzeitsicht

Ablauf Autorisierung durch AuthService



Als Beispiel der Autorisierung durch den Autorisierungsservice wird der Ablauf der Suche nach allen gespeicherten Bücher gezeigt.

Diese Darstellung ist stellvertretend für alle anderen Services des MediaService was die Autorisierung des Benutzers betrifft.



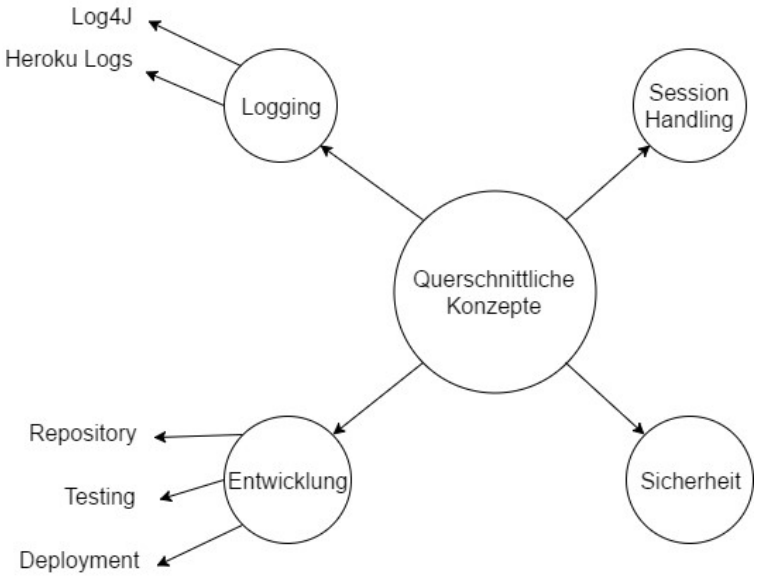
Falls der Token ungültig sein sollte oder der Benutzer nicht autorisiert ist, würde der Ablauf so aussehen. Auch hier ist die Abbildung stellvertretend für alle Services.

Verteilungssicht

Die Anwendung wird auf dem PaaS Anbieter Heroku deployed. Jedes Modul hat sein eigenes Dyno und seine eigene URL.

Für die Kommunikation der einzelnen Module muss die Adresse des jeweilig anderen Modules bekannt sein.

Querschnittliche Konzepte



Logging

Um die allgemeine Nutzung der Software auf Produktionsumgebung zu überprüfen werden verschiedene Logging Verfahren verwendet. Zum einen wird für Hibernate, zur Persistierung der Daten in den Datenbanken, Log4J verwendet. Für alle anderen Logausgaben wird auf das native Logsystem von Heroku zurückgegriffen. Hier werden unter anderem auch Fehler beim Deployment aufgezeichnet.

Session Handling

Da es sich bei der Kommunikation nach innen und nach außen um REST Services handelt, werden unter den Services keine Sessioninformationen weitergegeben.

Das gebietet die Definition von REST – die Services sollten Stateless, also zustandslos sein und somit auch keinerlei Informationen über den bisherigen Ablauf besitzen.

Sicherheit

Da für die Anmeldung und für die Informationen zum Aus- und Verleihen von Medien Benutzerdaten benötigt werden, ist Sicherheit ein wichtiger Punkt. Nichts ist schlimmer als wenn personenbezogene Daten kompromittiert werden.

Aus diesem Grund wird zum einen eine SSL Verschlüsselte Verbindung verwendet. Dies erledigt Heroku standardmäßig und auf unserer Seite entfällt der Konfigurationsaufwand.

Als zweites wird bei jedem Methodenaufruf der übergebene Token validiert. Die Token haben zudem ein maximales TTL von 30 Minuten. Das TTL wird bei jedem Methodenaufruf automatisch erneuert. Bei abgelaufenen TTL wird der Token aus dem System entfernt und der Benutzer muss sich erneut anmelden.

Ein weiteres Sicherheitskonzept ist, dass der Autorisierungsservice zwar die Anmeldedaten der Benutzer speichert, allerdings nicht personenbezogene Daten wie Adresse, Geburtsdatum, Studienfach, etc. Hier soll ein weiterer Service für die Stammdaten implementiert werden. Durch diese getrennte Datenhaltung ist sichergestellt, dass wenn wirklich ein Angreifer ins System eindringen kann, die Daten verteilt sind und nicht alle Daten auf einmal abgreifbar sind. So muss für jedes Modul ein eigener Angriff erfolgen.

Entwicklung

Als Repository wurde Github ausgewählt, da es frei verfügbar und leicht bedienbar ist. Es gewährleistet, dass mehrere Personen gleichzeitig am Sourcecode arbeiten können ohne sich gegenseitig zu blockieren. Jeder Entwickler arbeitet in seinem eigenen Branch und wenn er mit seiner UserStory fertig ist, wird sein Branch mit dem Master gemerged.

Für das Testen der Klassen sollen Unit Tests geschrieben werden. Zur Unterstützung wird Mockito verwendet um vorhersagbare Ergebnisse zu bekommen. Für die Integrationstests werden Beispieldaten in die Datenbank geladen und verschiedene Methoden aufgerufen. Das Ergebnis wird nun mit den Erwartungen verglichen.
Für manuelle Integrationstests kann das frei Tool *SoupUI* verwendet werden. Für automatisierte Tests *Tosca*. Wobei dieses Programm lediglich 14 Tage kostenlos getestet werden kann bevor man eine Lizenz käuflich erwerben muss.

Das Deployment auf Heroku kann entweder über das Github Repository erfolgen oder über das Build Tool *Maven*.
Der Nachteil der Lösung über Github ist allerdings, dass in ein Sourcecode Repository kompilierte Dateien hochgeladen werden müssen. Dies wiederspricht eigentlich dem Gedanken eines Repositories, da hier nur Sourcecode hochgeladen werden sollte.

Deshalb verwenden wir das Maven Plugin für Heroku. Es wird in der Plugin Konfiguration der Name der Heroku App eingetragen und mit dem Befehl

```
mvn clean install heroku:deploy
```

wird das Programm gebaut, Unittests ausgeführt und auf Heroku deployed.

Entwurfsentscheidungen

Auswahl der zu verwendeten Programmiersprache

Kriterien:

- Entwicklungsumgebungen frei verfügbar
- Geeignet für Programmieranfänger
- Lauffähig auf verschiedenen Betriebssystemen bei Umzug auf anderes Server Betriebssystem
- Für Entwicklung von REST Service geeignet

Getroffene Wahl: Java

Artefakt / Deployment Strategie

Kriterien:

- Ausführung der Applikation auf Anwendungsserver
- Bei Änderung des Codes möglichst geringer Aufwand bei Deployment

Getroffene Wahl: Deployment als Mikroservice auf Heroku

Glossar

Begriff	Definition
PaaS	*Plattform-as-a-Service*
TTL	*Time to live*
Deployment	*Veröffentlichung des kompilierten Programmes*